The Mechatronics Handbook

Second Edition

MECHATRONIC SYSTEM CONTROL, LOGIC, AND DATA ACQUISITION

Edited by

Robert H. Bishop

CRC Press
Taylor & Francis Group

How to go to your page

In this eBook, each chapter has its own page numbering scheme, consisting of a chapter number and a page number, separated by a hyphen.

For example, to go to page 5 of Chapter 1, type 1-5 in the "page #" box at the top of the screen and click "Go." To go to page 5 of Chapter 2, type 2-5… and so forth.

The Mechatronics Handbook

Second Edition

Edited by

Robert H. Bishop

Mechatronic Systems, Sensors, and Actuators

Fundamentals and Modeling

Mechatronic System Control, Logic, and Data Acquisition

The Electrical Engineering Handbook Series

Series Editor

Richard C. Dorf
University of California, Davis

Titles Included in the Series

The Handbook of Ad Hoc Wireless Networks, Mohammad Ilyas
The Avionics Handbook, Second Edition, Cary R. Spitzer
The Biomedical Engineering Handbook, Third Edition, Joseph D. Bronzino
The Circuits and Filters Handbook, Second Edition, Wai-Kai Chen
The Communications Handbook, Second Edition, Jerry Gibson
The Computer Engineering Handbook, Second Edition, Vojin G. Oklobdzija
The Control Handbook, William S. Levine
The CRC Handbook of Engineering Tables, Richard C. Dorf
The Digital Avionics Handbook, Second Edition Cary R. Spitzer
The Digital Signal Processing Handbook, Vijay K. Madisetti and Douglas Williams
The Electrical Engineering Handbook, Second Edition, Richard C. Dorf
The Electric Power Engineering Handbook, Second Edition, Leonard L. Grigsby
The Electronics Handbook, Second Edition, Jerry C. Whitaker
The Engineering Handbook, Third Edition, Richard C. Dorf
The Handbook of Formulas and Tables for Signal Processing, Alexander D. Poularikas
The Handbook of Nanoscience, Engineering, and Technology, Second Edition
William A. Goddard, III, Donald W. Brenner, Sergey E. Lyshevski, and Gerald J. Iafrate
The Handbook of Optical Communication Networks, Mohammad Ilyas and
Hussein T. Mouftah
The Industrial Electronics Handbook, J. David Irwin
The Measurement, Instrumentation, and Sensors Handbook, John G. Webster
The Mechanical Systems Design Handbook, Osita D.I. Nwokah and Yidirim Hurmuzlu
The Mechatronics Handbook, Second Edition, Robert H. Bishop
The Mobile Communications Handbook, Second Edition, Jerry D. Gibson
The Ocean Engineering Handbook, Ferial El-Hawary
The RF and Microwave Handbook, Second Edition, Mike Golio
The Technology Management Handbook, Richard C. Dorf
The Transforms and Applications Handbook, Second Edition, Alexander D. Poularikas
The VLSI Handbook, Second Edition, Wai-Kai Chen

MECHATRONIC SYSTEM CONTROL, LOGIC, AND DATA ACQUISITION

Edited by

Robert H. Bishop

The University of Texas at Austin
U.S.A.

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-0-8493-9260-3 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Mechatronic system control, logic, and data acquisition / editor, Robert H. Bishop.
p. cm.
"A CRC title."
Includes bibliographical references and index.
ISBN 978-0-8493-9260-3 (alk. paper)
1. Programmable controllers. 2. Mechatronics. I. Bishop, Robert H., 1957-

TJ223.P76M43 2008
621--dc22 2007025442

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Preface

According to the original definition of mechatronics proposed by the Yasakawa Electric Company and the definitions that have appeared since, many of the engineering products designed and manufactured in the last 30 years integrating mechanical and electrical systems can be classified as *mechatronic systems*. Yet many of the engineers and researchers responsible for those products were never formally trained in mechatronics per se. The *Mechatronics Handbook, 2nd Edition* can serve as a reference resource for those very same design engineers to help connect their everyday experience in design with the vibrant field of mechatronics.

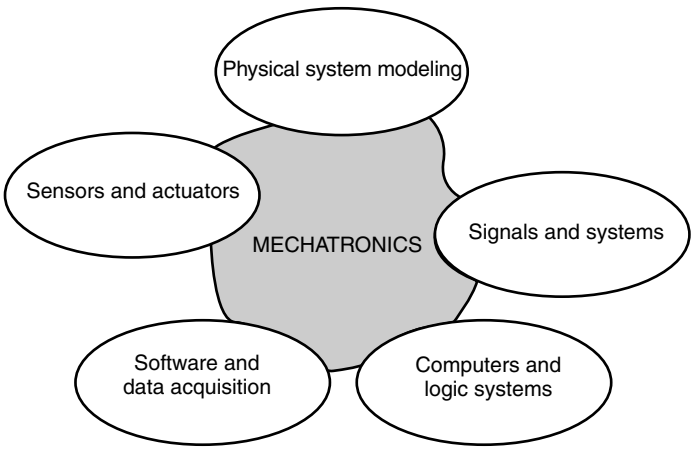
The *Handbook of Mechatronics* was originally a single-volume reference book offering a thorough coverage of the field of mechatronics. With the need to present new material covering the rapid changes in technology, especially in the area of computers and software, the single-volume reference book quickly became unwieldy. There is too much material to cover in a single book. The topical coverage in the *Mechatronics Handbook, 2nd Edition* is presented here in two books covering *Mechatronic Systems, Sensors, and Actuators: Fundamentals and Modeling* and *Mechatronic System Control, Logic, and Data Acquisition*. These two books are intended for use in research and development departments in academia, government, and industry, and as a reference source in university libraries. They can also be used as a resource for scholars interested in understanding and explaining the engineering design process.

As the historical divisions between the various branches of engineering and computer science become less clearly defined, we may well find that the mechatronics specialty provides a roadmap for nontraditional engineering students studying within the traditional structure of most engineering colleges. It is evident that there is an expansion of mechatronics laboratories and classes in the university environment worldwide. This fact is reflected in the list of contributors to these books, including an international group of academicians and engineers representing 13 countries. It is hoped that the books comprising the *Mechatronics Handbook, 2nd Edition* can serve the world community as the definitive reference source in mechatronics.

Organization

The *Mechatronics Handbook, 2nd Edition* is a collection of 56 chapters covering the key elements of mechatronics:

a. Physical Systems Modeling
b. Sensors and Actuators
c. Signals and Systems
d. Computers and Logic Systems
e. Software and Data Acquisition



Key Elements of Mechatronics

Mechatronic System Control, Logic, and Data Acquisition

An overview of signals and system control, computers, logic systems, software, and data acquisition is presented in this book. These are most rapidly changing areas of mechatronics.

Section I—Mechatronic System Control

Since there is a significant body of readily available material to the reader on the general subject of signals and systems, there is no overriding need to repeat that material here. Instead, the goal of this book is to present the relevant aspects of signals and systems of special importance to the study of mechatronics. The book begins with chapters on the role of control in mechatronics and on the role of modeling in mechatronic design. These chapters set the stage for the more fundamental discussions on signals and systems comprising the bulk of the material in this section. Modern aspects of control design using optimization techniques from H_2 theory, adaptive and nonlinear control, neural networks, and fuzzy systems are also included as they play an important role in modern engineering system design. The book includes chapters on design optimization for mechatronic systems, and real-time monitoring and control. The chapters, listed in order of appearance, are

1. The Role of Controls in Mechatronics
2. The Role of Modeling in Mechatronics Design
3. Signals and Systems
   - 3.1 Continuous- and Discrete-Time Signals
   - 3.2 *z* Transforms and Digital Systems
   - 3.3 Continuous- and Discrete-Time State-Space Models
   - 3.4 Transfer Functions and Laplace Transforms
4. State Space Analysis and System Properties
5. Response of Dynamic Systems
6. The Root Locus Method
7. Frequency Response Methods
8. Kalman Filters as Dynamic System State Observers
9. Digital Signal Processing for Mechatronic Applications
10. Control System Design via H^2 Optimization
11. Adaptive and Nonlinear Control Design
12. Neural Networks and Fuzzy Systems
13. Advanced Control of an Electrohydraulic Axis
14. Design Optimization of Mechatronic Systems
15. Motion Control
16. Real-Time Monitoring and Control
17. Micromechatronics and Microelectromechanical Motion Devices

Section II—Computers and Logic Systems

The development of the computer, and then the microcomputer, embedded computers, and associated information technologies and software advances, has impacted the world in a profound manner. This is especially true in mechatronics where the integration of computers with electromechanical systems has led to a new generation of smart products. The future is filled with promise of better and more intelligent products resulting from continued improvements in computer technology and software engineering. In this section, the focus is on computer hardware and associated issues of logic, communication, networking, architecture, fault analysis, embedded computers, and programmable logic controllers. The chapters, listed in order of appearance, are

18. Introduction to Computers and Logic Systems
19. Digital Logic Concepts and Combinational Logic Design
20. System Interfaces
21. Communications and Computer Networks
22. Fault Analysis in Mechatronic Systems
23. Logic System Design

24. Architecture
25. Control with Embedded Computers and Programmable Logic Controllers
26. Graphical System Design for Embedded Systems
27. Field-Programmable Gate Arrays
28. Graphical Programming for Field-Programmable Gate Arrays: Applications in Control and Mechatronics

Section III—Software and Data Acquisition

Given that computers play a central role in modern mechatronics products, it is very important to understand how data is acquired and how it makes its way into the computer for processing and logging. The final section of this book is devoted to the issues surrounding computer software and data acquisition. The chapters, listed in order of appearance, are

29. Introduction to Data Acquisition
30. Measurement Techniques: Sensors and Transducers
31. A/D and D/A Conversion
32. Signal Conditioning
33. Virtual Instrumentation Systems
34. Software Design and Development
35. Data Recording and Logging

Acknowledgments

I wish to express my heartfelt thanks to all the contributing authors. Taking time in otherwise busy and hectic schedules to author the excellent chapters appearing in this book is much appreciated.

This handbook is a result of a collaborative effort expertly managed by CRC Press. My thanks to the editorial and production staff:

Nora Konopka	Acquisitions Editor
Theresa Delforn	Project Coordinator
Joette Lynch	Project Editor

Thanks to my friend and collaborator Professor Richard C. Dorf for his continued support and guidance. And finally, a special thanks to Lynda Bishop for managing the incoming and outgoing draft manuscripts. Her organizational skills were invaluable to this project.

Editor

Robert H. Bishop is a professor of aerospace engineering and engineering mechanics at The University of Texas at Austin and holds the Joe J. King Professorship. He received his BS and MS from Texas A&M University in aerospace engineering, and his PhD from Rice University in electrical and computer engineering. Prior to coming to The University of Texas at Austin, he was a member of the technical staff at the MIT Charles Stark Draper Laboratory. Dr. Bishop is a specialist in the area of planetary exploration with emphasis on spacecraft guidance, navigation and control. He is a fellow of the American Institute of Aeronautics and Astronautics. Currently, Dr. Bishop is currently working with the NASA Johnson Space Center on techniques for achieving precision landing on the moon and Mars. He is an active researcher authoring and co-authoring over 100 journal and conference papers. He was twice selected a faculty fellow at the NASA Jet Propulsion Laboratory and as a Welliver faculty fellow by The Boeing Company. Dr. Bishop co-authors *Modern Control Systems* with Professor R. C. Dorf, and he has authored two other books entitled *Learning with LabView* and *Modern Control System Design and Analysis Using Matlab and Simulink*. He received the John Leland Atwood Award by the American Society of Engineering Educators and the American Institute of Aeronautics and Astronautics that is given periodically to "a leader who has made lasting and significant contributions to aerospace engineering education." Dr. Bishop is a member of the Academy of Distinguished Teachers at The University of Texas at Austin.

List of Contributors

Maruthi R. Akella
Department of Aerospace Engineering and Engineering Mechanics
The University of Texas at Austin
Austin, Texas

Craig Anderson
National Instruments, Inc.
Austin, Texas

Dragos Arotaritei
Department of Computer Science and Engineering
Aalborg University
Esbjerg, Denmark

Brian Betts
Data Acquisition and Signal Conditioning
National Instruments, Inc.
Austin, Texas

Tomas Brezina
Technical University of Brno
Brno, Czech Republic

George I. Cohn
California State University
Los Angeles, California

Daniel A. Connors
Department of Electrical and Computer Engineering
University of Colorado at Boulder
Boulder, Colorado

Kevin C. Craig
Department of Mechanical, Aerospace, and Nuclear Engineering
Rensselaer Polytechnic Institute
Troy, New York

Timothy P. Crain II
Department of Aerospace Engineering and Engineering Mechanics
NASA Johnson Space Center
Houston, Texas

Raymond A. de Callafon
Department of Mechanical and Aerospace Engineering
University of California, San Diego
La Jolla, California

Darcy Dement
National Instruments, Inc.
Austin, Texas

Santosh Devasia
Department of Mechanical Engineering
University of Washington
Seattle, Washington

C. Nelson Dorny
Moore School of Electrical Engineeering
University of Pennsylvania
Philadelphia, Pennsylvania

Stephen A. Dyer
Electrical and Computer Engineering
Kansas State University
Manhattan, Kansas

Jeannie Sullivan Falcon
National Instruments, Inc.
Austin, Texas

Daniel R. Fay
University of Colorado at Boulder
Boulder, Colorado

Gerardo Garcia
National Instruments, Inc.
Austin, Texas

Shelley Gretlein
National Instruments, Inc.
Austin, Texas

Margaret H. Hamilton
Hamilton Technologies, Inc.
Tucson, Arizona

Cecil Harrison
University of Southern Mississippi
Hattiesburg, Mississippi

Bonnie S. Heck
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia

Wen-Mei W. Hwu
University of Illinois
Urbana, Illinois

Mohammad Ilyas
Florida Atlantic University
Boca Raton, Florida

Florin Ionescu
University of Applied Sciences
Konstanz, Germany

Hugh Jack
Product Design and Manufacturing Engineering
Grand Valley State University
Grand Rapids, Michigan

Jeffrey A. Jalkio
University of St. Thomas
St. Paul, Minnesota

Rolf Johansson
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Jayantha Katupitiya
The University of New South Wales
Sydney, New South Wales, Australia

Ctirad Kratochvil
Technical University of Brno
Brno, Czech Republic

Rahul Kulkarni
Industrial Data Acquisition and Control
National Instruments, Inc.
Austin, Texas

Thomas R. Kurfess
Department of Mechanical Engineering
Clemson University
Clemson, South Carolina

Kam K. Leang
Department of Mechanical Engineering
University of Washington
Seattle, Washington

Sergey Edward Lyshevski
Department of Electrical Engineering
Rochester Institute of Technology
Rochester, New York

Thomas N. Moore
Department of Mechanical Engineering
Queen's University
Kingston, Ontario, Canada

Leila Notash
Department of Mechanical Engineering
Queen's University
Kingston, Ontario, Canada

Cestmir Ondrusek
Technical University of Brno
Brno, Czech Republic

Hitay Özbay
Department of Electrical Engineering
The Ohio State University
Columbus, Ohio

M. K. Ramasubramanian
Department of Mechanical and Aerospace Engineering
North Carolina State University
Raleigh, North Carolina

Armando A. Rodriguez
Department of Electrical Engineering
Arizona State University
Tempe, Arizona

Momoh-Jimoh Eyiomika Salami
International Islamic University of Malaysia
Kuala Lumpur, Malaysia

Mario E. Salgado
Departamento de Electrónica
Universidad Tećnica Federico Santa Maria
Valparaíso, Chile

Jyh-Jong Sheen
Department of Mechanical Engineering and Marine Engineering
National Taiwan Ocean University
Keelung, Taiwan

Andrew Sterian
Padnos College of Engineering and Computing
Grand Valley State University
Grand Rapids, Michigan

Fred Stolfi
Xerox Mechanical Engineering Sciences Laboratory
Rensselaer Polytechnic Institute,
Troy, New York

Michael J. Tordon
The University of New South Wales
Sydney, New South Wales, Australia

Michael Trimborn
National Instruments, Inc.
Austin, Texas

Job van Amerongen
University of Twente
Enschede, The Netherlands

Crina Vlad
Politehnica University of Bucharest
Bucharest, Romania

Bogdan M. Wilamowski
Department of Electrical and Computer Engineering
Auburn University
Auburn, Alabama

Juan I. Yuz
Universidad Tećnica Federico Santa Maria
Valparaíso, Chile

Qingze Zou
University of Washington
Seattle, Washington

Contents

SECTION I Mechatronic System Control

SECTION II Computers and Logic Systems

SECTION III Software and Data Acquisition

I

Mechatronic System Control

1

The Role of Controls in Mechatronics

Job van Amerongen
University of Twente

1.1 Introduction

"Mechatronic design deals with the integrated and optimal design of a mechanical system and its embedded control system." This definition implies that the mechanical system is enhanced with electronic components in order to achieve a better performance, a more flexible system, or just reduce the cost of the system. In many cases the electronics are present in the form of a computer-based embedded (control) system. This does not imply that every controlled mechanical system is a mechatronic system because in many cases the control is just an add-on to the mechanical system in a sequential design procedure. A real mechatronics approach requires that an optimal choice be made with respect to the realization of the design specifications in the different domains. In control engineering the design of an optimal control system is well understood and for linear systems standard methods exist. The optimization problem is formulated as: given a process to be controlled, and given a performance index (cost function), find optimal controller parameters such that the cost function is minimized. With a state feedback controller and a quadratic cost function, solutions for the optimal controller gains can be found with standard controller design software, such as MATLAB[1] (Figure 1.1).

Mechatronic design on the contrary requires that not only the controller be optimized. It requires optimization of the system as a whole. In the ideal case all the components in the system: the process itself, the controller, as well as the sensors and actuators, should be optimized simultaneously (Figure 1.2).

In general this is not feasible. The problem is ill defined and has to be split into smaller problems that can be optimized separately. Later on the partial solutions have to be combined and the performance of the complete system has to be evaluated. After eventually readjusting some parts of the system this leads to a sub-optimal solution.

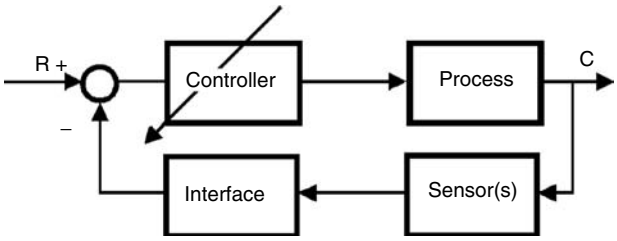


FIGURE 1.1 Optimization of the controller.

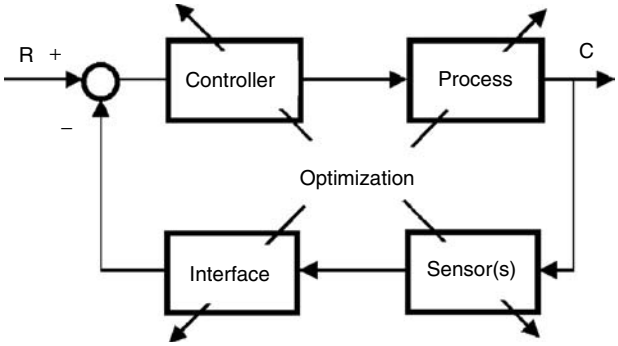


FIGURE 1.2 Optimization of the all system components simultaneously.

In the initial conceptual design phase it has to be decided which problems should be solved mechanically and which problems electronically. In this stage decisions about the dominant mechanical properties have to be made, yielding a simple model that can be used for controller design. Also a rough idea about the necessary sensors, actuators, and interfaces has to be available in this stage. When the different partial designs are worked out in some detail, information about these designs can be used for evaluation of the complete system and be exchanged for a more realistic and detailed design of the different parts.

Although the word mechatronics is new, mechatronic products have been available for some time. In fact, all electronically controlled mechanical systems are based on the idea of improving the product by adding features realized in another domain. Good mechatronic designs are based on a *real systems approach*. But mostly, control engineers are confronted with a design in which major parameters are already fixed, often based on static or economic considerations. This prohibits optimization of the system as a whole, even when optimal control is applied.

In the last days of gramophones, the more sophisticated designs used tacho feedback in combination with a light turntable to achieve a constant number of revolutions. But a really new design was the compact disc player. Instead of keeping the number of revolutions of the disc constant, it aims for a constant speed of the head along the tracks of the disc. This means that the disc rotates slower when tracks with a greater diameter are read. The bits read from the CD are buffered electronically in a buffer that sends its information to the DA converter, controlled by a quartz crystal. This enables the realization of a very constant bit rate and eliminates all audible speed fluctuations. Such a performance could never be obtained from a pure mechanical device only, even if it were equipped with a good speed control system. In fact, the control loop for the disc speed does not need to have very strict specifications. It should only prevent overflow or underflow of the buffer. The high accuracy is obtained in an open loop mode, steered by a quartz crystal (Figure 1.3).

The flexibility introduced by the combination of precision mechanics and electronic control has allowed the development of CD-ROM players, running at speeds more than 50 times faster than the original audio CDs. A new way of thinking was necessary to come to such a new solution. On the other

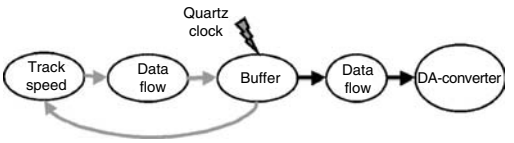


FIGURE 1.3 Combination of closed-loop and open-loop control in a CD player.

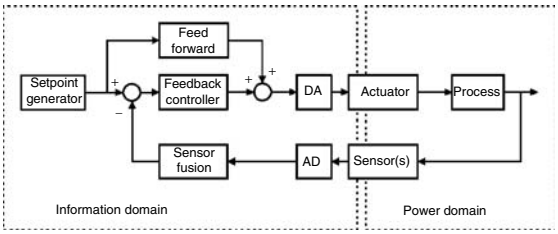


FIGURE 1.4 Mechatronic system.

hand, the CD player is still a sophisticated piece of precision mechanics. No solid-state electronic memory device can compete yet economically with the opto-mechanical storage capabilities of the CD and its successor the DVD. But this may change rapidly.

1.2 Key Elements of Controlled Mechatronic Systems

A mechatronic system consists by definition of a mechanical part that has to perform certain motions and an electronic part (in many cases an embedded computer system) that adds intelligence to the system. In the mechanical part of the system power plays a major role. This in contrast to the electronic part of the system where information processing is the main issue. Sensors convert the mechanical motions into electrical signals where only the information content is important or even into pure information in the form of numbers (if necessary, through an AD converter). Power amplifiers convert signals into modulated power. In most cases the power supply is electrical, but other sources such as hydraulic and pneumatic power supplies are possible as well. A controlled mechanical motion system thus typically consists of a mechanical construction, one or more actuators to generate the desired motions, and a controller that steers the actuators based on feed-forward and sensor-based feedback control (Figure 1.4).

1.3 Integrated Modeling, Design and Control Implementation

1.3.1 Modeling

During the design of mechatronic systems it is important that changes in the construction and the controller be evaluated simultaneously. Although a proper controller enables building a cheaper construction, a badly designed mechanical system will never be able to give a good performance by adding a sophisticated controller. Therefore, it is important that during an early stage of the design a proper choice can be made with respect to the mechanical properties needed to achieve a good performance of the controlled system. On the other hand, knowledge about the abilities of the controller to compensate for mechanic imperfections may enable that a cheaper mechanical construction be built. This requires that in an early stage of the design a simple model is available that reveals the performance limiting factors of the system. Still there is a gap between modeling and simulation software used for evaluation of mechanical constructions and software used for controller design. Mechanical engineers are used to

finite element packages to examine the dynamic properties of mechanical constructions. It is only after reduction to low-order models (modal analysis) that these models can be used for controller design. On the other hand, typical control-engineering software does not directly support the mechatronic design process either; in the modeling process the commonly used transfer functions and state space descriptions often have lost the relation with the physical parameters of the mechanical construction. Tools are required that allow modeling of mechanical systems in a way that the dominant physical parameters (like mass and dominant stiffness) are preserved in the model and simultaneously provide an interface to the controller design and simulation tools control engineers are used to (Coelingh;[2] Coelingh et al.[3]).

Simulation is an important tool to evaluate the design of mechatronic systems. Most simulation programs like Simulink[1] use block diagram representations and do not support physical modeling in a way that direct tuning of the physical parameters of the mechanical construction and those of the controller is possible as required in the design of mechatronic systems. Recently, programs that allow physical modeling in *various physical domains* became available. They use an object-oriented approach that allows hierarchical modeling and reuse of models. The order of computation is only fixed after combining the subsystems. Examples of these programs are 20-sim,[4] described by Broenink[5] as CAMAS and Dymola.[6]

In this section the modeling and simulation program 20-sim (pronounced Twente Sim) will be used to illustrate the simultaneous design of construction and controller in a mechatronic system. 20-sim supports object-oriented modeling. Power and signal ports to and from the outside world determine each object[7]. Inside the object there can be other objects or, on the lowest level, equations. Various *realizations* of an object can contain different or more detailed descriptions as long as the interface (number and type of ports) is identical. Modeling can start by a simple interconnection of (empty) submodels. Later they can be filled with realistic descriptions with various degrees of complexity. De Vries[8] refers to this as *polymorphic* modeling. Submodels can be constructed from other submodels in hierarchical structures. Proper physical modeling is achieved by coupling the submodels by means of the *flow of energy*, rather than by *signals* such as voltage, current, force, and speed. This way of modeling is well suited for mechatronics system design. It will be illustrated with an example. We want to consider the design of a simple servo system, considering the use of a voltage source, a DC motor, and a mechanical load driven through a transmission (Figure 1.5).

The transmission is disregarded for the time being. The belt is considered as infinitely stiff and the transformation ratio is taken care of by changing the motor constant. If a power amplifier driven by a signal generator describes the voltage source, we can draw the iconic diagram of Figure 1.6. At this stage the different *components* in this model are still empty. But all components have electrical and/or mechanical "ports." With the proper interfaces (ports) defined, the components can be connected to each other.

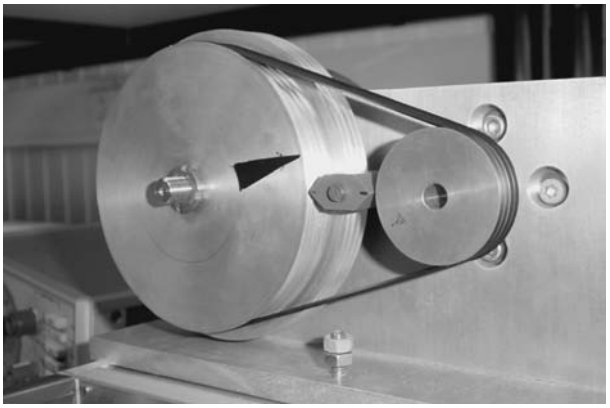


FIGURE 1.5 Simple DC-servo system.

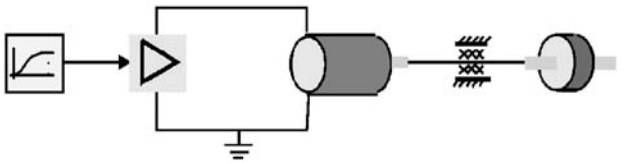


FIGURE 1.6 Iconic diagram of the simple servo system.

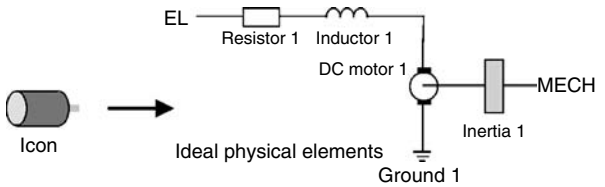


FIGURE 1.7 Icon of the motor expanded to ideal physical elements.

In the next step we can detail the description of the DC motor. One solution could be the description given in Figure 1.7. The motor is now described by a number of *ideal physical elements*, each representing a basic physical relation. The motor has an electrical (EL) as well as a mechanical port (MECH).

Each of the *elements* in this figure can be described as an element with an electrical and/or mechanical port. The idea of ports is made more explicit in so-called bond graphs.[9–12] For the electrical elements these are the voltage difference over the element and the current through the element. For the mechanical elements these are the torque and the (angular) velocity. The products of these conjugated variables ($P = ui$ or $P = T\omega$) represent power.

If we go down a step further into the hierarchy, we arrive at the level of equations. For instance, an electrical resistor can be described by the equation:

$$p \cdot u = R * p \cdot i \tag{1.1}$$

where the variables $p \cdot u$ and $p \cdot i$ indicate the conjugated variables u and i of the electrical port p. Note that this is an equation and not an assignment statement. It could have been written equally well in the form:

$$p \cdot i = 1/R * p \cdot u \tag{1.2}$$

In a similar way the inductance can be described by the equations:

$$p \cdot u = L * \mathrm{ddt}(p \cdot i) \qquad \text{or} \qquad (p \cdot i) = 1/L * \mathrm{int}(p \cdot u) \tag{1.3}$$

where $\mathrm{ddt}(p \cdot i)$ denotes di/dt and $\mathrm{int}(p \cdot u)$ denotes $\int u\,dt$. In case of an R-element there is no preference for one of the two forms. For the I-element the integral form is preferred in the simulations. 20-sim determines the preferred causal form and derives the equations automatically.

The energy flow or *power P* is the product of *two conjugated signals*, called effort (e) and flow (f):

$$P = ef \tag{1.4}$$

Examples of this expression in the mechanical and electrical domain are

$$P = Fv \qquad \text{or} \qquad P = T\omega \tag{1.5}$$

$$P = ui \tag{1.6}$$

where F is force, v is velocity, T is torque, ω is angular velocity, u is voltage, and i is current.

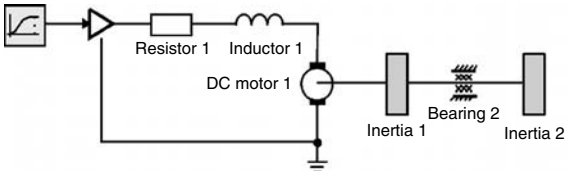


FIGURE 1.8 Complete model in the form of ideal physical elements.

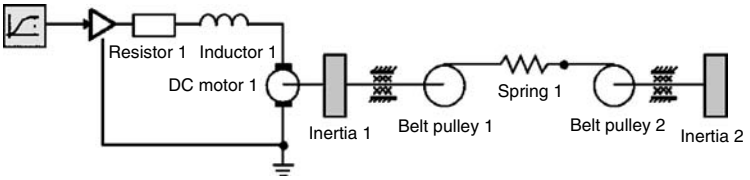


FIGURE 1.9 Model extended with transmission.

When we expand the complete Figure 1.6 we obtain Figure 1.8. When this model is processed a message pops up that indicates that inertia 2 has a dependent state. The two inertias in this model always have the same speed, and therefore, they are dependent. They cannot have independent initial conditions. The message indicates that this element can only be written in derivative form:

$$T = J\, d\omega/dt \tag{1.7}$$

There are several ways to deal with this problem.

1. The two inertias can be combined into one inertia (the program will do this automatically). A message pops up that the dependency of the two inertias has been solved symbolically.
2. Dealing with the derivative causality by means of an implicit integration algorithm.
3. The transmission can be added, including some flexibility in the belt.

If the flexibility is negligible, solution 1 leads to the simplest model. On the other hand, the warning raises the question whether the flexibility of the belt can be disregarded indeed. If not, the model has to be extended with a spring element. It should be noted that this should not be done for numerical reasons only. If the transmission were very stiff, this would result in high-frequency dynamics and lead to unnecessary slow simulations. On the other hand, if the flexibility is important, as it is in this system, the warning draws the designer's attention to the fact that the model may be oversimplified. In Figure 1.9 the transmission, including a spring element, has been added. Processing of this model does not produce any warnings.

This example illustrates how modern software can help to come up with a model that has the complexity that is needed for a particular problem. Physical models, in the form of an iconic diagram, based on connecting elements by means of power ports, may help in this modeling process. The user can select the preferred view, whether this is a bond graph, an iconic diagram with ideal physical element, or a view using higher lever submodels, like in Figure 1.6. In the next section it will be shown how to use this model for the design of controllers.

1.3.2 Control System Design Methodologies

Many processes can be reasonably well controlled by means of PID controllers. This is due to the fact that these processes can be more or less accurately described by means of a second-order model. Tuning rules, like those of Ziegler Nichols, enable less experienced people to tune such controllers. Relatively simple

models can also describe many mechatronic systems. A mechatronic system mostly consists of an actuator, some form of transmission, and a load. A fourth-order model can properly describe such a system. The performance-limiting factor in these systems is the resonance frequency. A combination of position and tacho feedback (basically a PD controller) can be applied here as well. But due to the resonant poles proper selection of the signals to be used in the feedback is essential. Efforts have been made[2,3,13] to derive recipes for tuning such systems, in addition to selecting the proper feedback signals. Computer support tools are essential to enable less experienced designers to use these recipes (Van Amerongen, Coelingh, and De Vries[14]). Coelingh[2] and Coelingh et al.[3] describe a structural design method for mechatronic systems. The method starts with reducing the conceptual design to a fourth-order model that represents the dominant properties of the system in terms of the total mass to be moved and the dominant stiffness. This model still has physical meaningful parameters. In this model appropriate sensors are chosen, as well as a path generator. In the conceptual design phase a simple controller is developed and mechanical properties are changed, if necessary. Then a more detailed design phase follows where also parameter uncertainties are taken into account.

1.3.3 Servo System Design

Here we will consider some simple aspects of the design of a servo system in order to illustrate the advantage of the use of physical models and to illustrate the need for an integrated design approach. We consider the model discussed before, a load driven by an electric motor, through a flexible transmission. The iconic diagram of this model was given in Figure 1.9. In this example a current amplifier has replaced the voltage amplifier allowing the removal of the electrical resistor and the inductance. In the step responses of Figure 1.10 the resonance due to the flexible transmission is clearly visible.

From the equations used for the simulation, 20-sim can automatically derive a model in a form suitable for controller design, such as a state-space description, a transfer function, or poles and zeros. An interface is provided to MATLAB[1] enabling, for instance, to use MATLAB algorithms to compute the gains of advanced controllers like an LQR (optimal state feedback) or LQG controller (with a Kalman filter for state estimation and optimal state feedback). The diagram of the process together with an LQG controller is given in Figure 1.11 and some responses in Figure 1.12.

A properly designed P(I)D controller is able to perform almost similarly, especially when the amount of noise is small. A first attempt could be to use only measurements of the load angle and load speed.

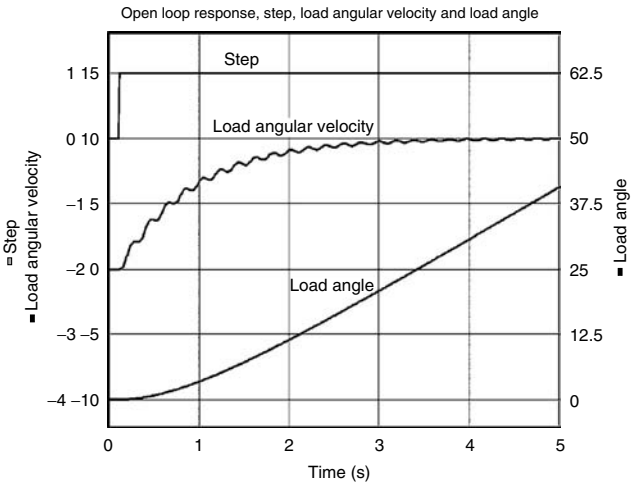


FIGURE 1.10 Open loop responses.

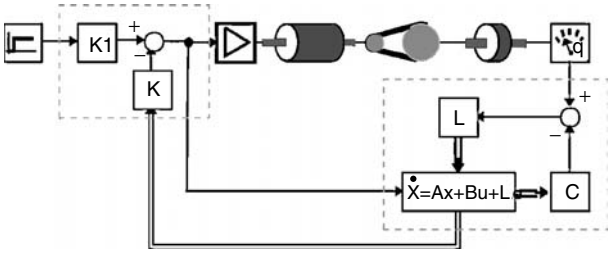


FIGURE 1.11 Process with Kalman filter and state feedback.

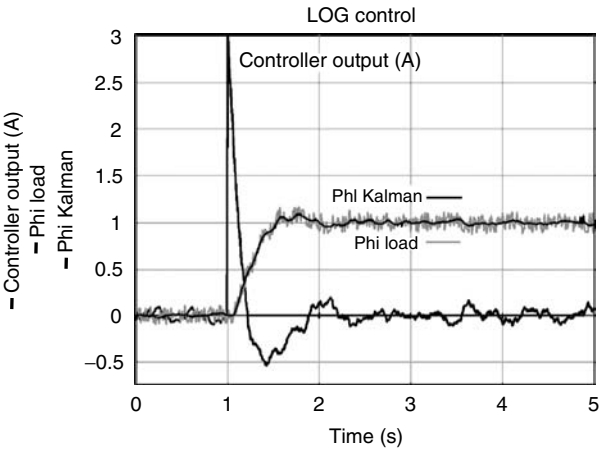


FIGURE 1.12 Response of the LQG-controlled system.

This attempt fails, because feedback of the load speed leads almost immediately to an unstable system as can be seen from the root locus for variations in the gain of the velocity feedback. From the responses of Figure 1.10, 20-sim can easily determine the transfer function between the motor current and the load speed and plot the root locus (Figure 1.13).

Figure 1.14 clearly shows that even a small amount of velocity feedback will lead to an unstable system. It is well known that feedback of the motor speed is a better solution. Using again the model of Figures 1.9 and 1.10 to determine the transfer from input current to motor speed yields the root locus of Figure 1.15.

Complex zeros now accompany the complex poles and because they are close together their influence on the response will be almost negligible. The branch of the root locus on the real axis now shows the desired behavior: moving the dominant pole to the left in the *s*-plane. Combining the feedback of the motor speed with feedback of the load angle yields the PD-controller structure of Figure 1.15 and the responses of Figure 1.16. Except for the noise there is not much difference in the responses of the system with the Kalman filter, although the PD-controlled system is simpler. The observations made here are generally applicable. A system with two resonant (complex) poles and no zeros, such as in Figure 1.13, is difficult to control by means of a simple controller. If complex zeros accompany the resonant poles with an imaginary part smaller than that of the poles, stable control is easily achieved. In the frequency domain this is seen as an anti-resonance, followed by a resonance (type *AR*). On the contrary a type *RA* system, where the resonance frequency is lower than the anti-resonance frequency (the imaginary part of the poles is smaller than that of the zeros), is just as difficult to control as in the case of only resonant poles. The existence and location of resonant zeros is completely determined by the (geometrical) location of

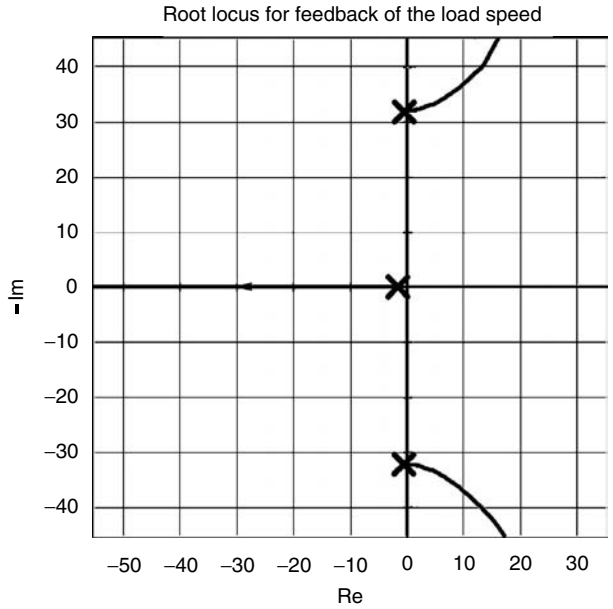


FIGURE 1.13 Root locus for velocity feedback of load axis.

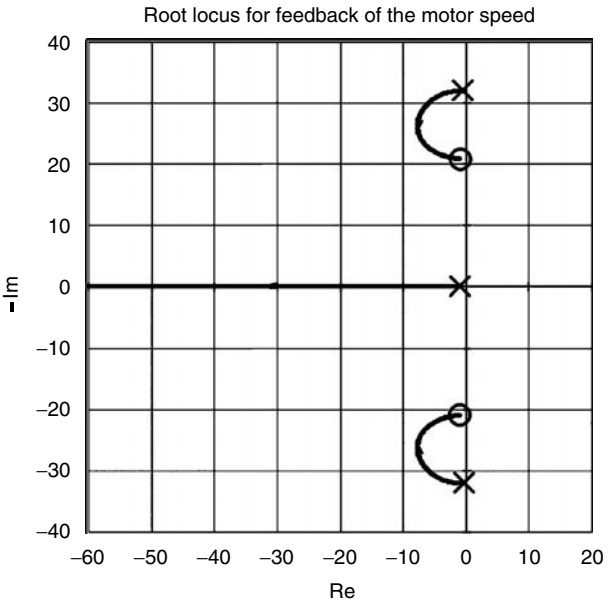


FIGURE 1.14 Root locus for velocity feedback of motor axis.

the sensors in the mechanical system. A careful choice of these sensor locations is therefore crucial for the successful application of a controller. It should be noted that using a properly designed set-point generator could prevent resonance, as seen in Figure 1.10. The set point generator should not excite the resonance frequencies, for instance, by using a low pass filter with bandwidth lower than the resonance frequencies. However, such a set-point generator does not solve the above-mentioned stability problems.

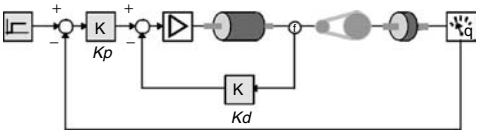


FIGURE 1.15 Servo system with PD-controller.

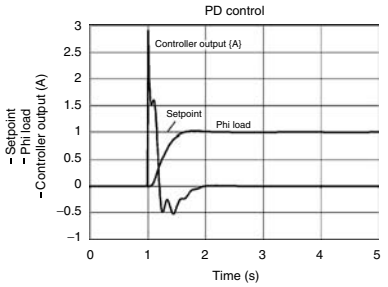


FIGURE 1.16 Responses of the system of Figure 1.16.

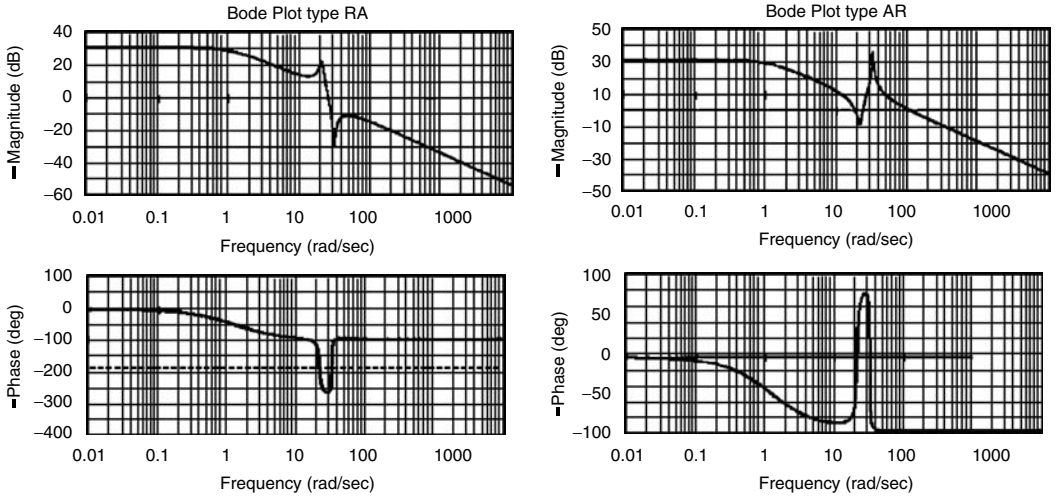


FIGURE 1.17 Bode plots of type *RA* and *AR* systems.

1.3.4 Design of a Mobile Robot

A typical example of the early design procedure is the conceptual design of a mobile assembly robot. Already in a very early stage of the design conflicting demands have to be resolved. Such a robot should be able to collect parts all around a production facility and do the assembly while driving. Because a high accuracy is required between the gripper of the robot and the surface where the parts are located, it is important that floor irregularities and vibration modes of the structure do not prevent proper assembly. On the other hand the path controller, partly based on dead reckoning (i.e., measuring of the wheel speed and orientation), requires that the wheels be very stiff. Damping of disturbances has to be realized by another means of suspension. This has led to the concept of an upper frame and a lower frame, connected by means of springs (Figure 1.18).

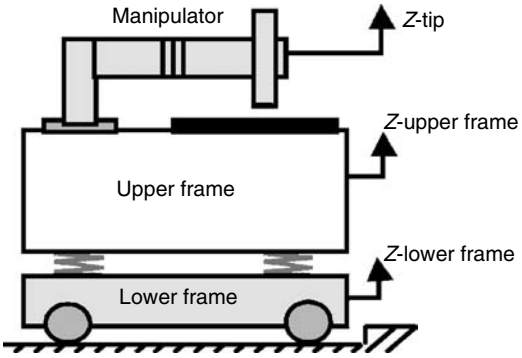


FIGURE 1.18 Conceptual design of the mobile robot.

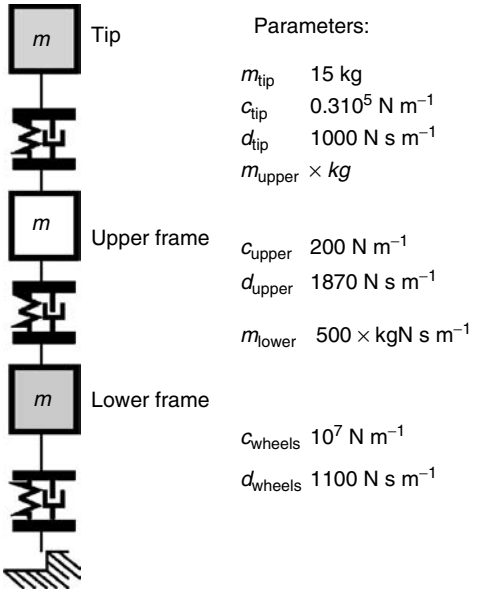


FIGURE 1.19 Simple model with ideal physical elements to compute the error e_{tip}.

The robot can be mounted at the upper frame and should have sufficient bandwidth such that the position error ($e_{tip} = z_{tip} - z_{upper\ frame}$) between the tip of the robot (z_{tip}) and the upper frame ($z_{upper\ frame}$) is small enough.

The next step is to derive a simple model, in order to have some parameters for the weight distribution and the stiffness and damping of the springs. In the model of Figure 1.22 the robot is confronted with a bump in the floor at a speed of 1 m/s.

Based upon the payload—mainly the weight of the batteries—the total mass of the vehicle was estimated to be 500 kg. Stiffness and damping of the wheels follow from the demands for the accuracy of the position estimation. The mass and bandwidth of the controlled manipulator were already known from other studies, yielding the effective stiffness and damping for the robot tip. When also initial estimates of the stiffness and damping of the springs between the upper and lower frame are made, the only parameter to be varied is the weight distribution between the upper frame and lower frame. By using the optimization feature of 20-sim, the optimal weight distribution can easily be found. In order to minimize the error between the tip of the robot and the upper frame (Figure 1.19), the weight has to be placed as much as possible in the

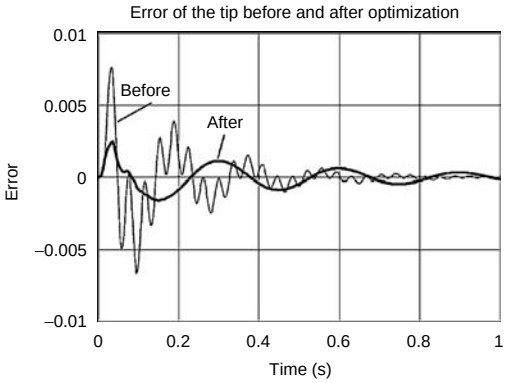


FIGURE 1.20 Error of the tip before and after optimization of the weight distribution between upper and lower frame.

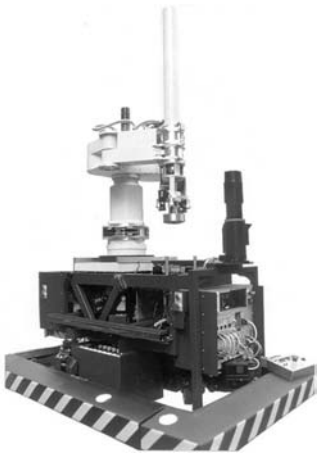


FIGURE 1.21 The mobile robot (MART) after completion.

upper frame (Figure 1.20). This example illustrates how the mechanical configuration of the system is determined by the requirements for good path control and accurate control of the assembly task.

A next step could be to optimize the properties of the suspension between upper and lower frame. This will further improve the error. This decision made in a very early stage of the design directed other design decisions. After completion of the project it appeared that the different parameters of the final construction were close to these early estimates (Figure 1.21).

1.4 Modern Examples of Mechatronic Systems in Action

A few examples have already been treated in the previous sections. In this section two more examples will be given.

1.4.1 Rudder Roll Stabilization of Ships

Nowadays most ships use an autopilot to control the heading of the ship. A rudder is the most commonly used actuator. Some ships, like ferries and naval ships, need also roll stabilization. This can be achieved passively by means of two connected tanks filled with water that generate stabilizing forces that should

be in counter phase with the forces of the waves. In order to make the system effective for varying frequencies of the waves, the water flow between the two tanks should be controlled. For fast ships mostly stabilizing fins are used. These are a kind of actively controlled "wings" that generate the moments needed to counteract the moments of the waves. The fins not only influence the roll motions but also have influence on the heading. On the other hand, the rudder not only influences the heading but also induces roll. In control engineering terms this leads to a multivariable system that requires a multivariable controller design for optimum performance. In practice such a multivariable system is seldom seen and two separate control systems are used.

Another approach is to use only one of the actuators (rudder or fins) to achieve course control and roll reduction. Because the frequencies of the roll motions are outside the bandwidth of the course-control system this is possible. The rudder is most suited as actuator. An additional advantage for naval ships is that removing the fins will reduce the underwater noise of the vessel.

Redesigning the course controller in order to stabilize the roll as well, demonstrates the feasibility of this approach, but also makes clear that the "process"—the ship—should be modified. The most important modification is needed for the steering machine. The maximum speed of the steering machine appears to be the limiting factor for such a system (it should increase from the commonly used values of 3–7°/s to 20–25°/s). By means of dynamic simulations the demands for the steering machine can be found in terms of the maximum speed of the steering machine and the maximum time constant that is allowed for reaching this speed. This requires reengineering of the hydraulic steering machine. A step further would be to consider also changes in the shape of the ship, in order to optimize the parameters that determine the effectiveness of the rudder roll stabilization system.

In order to decide whether this new solution is better, it should be evaluated whether the redesigned steering machine is less expensive than the original rudder and fin actuators. These design issues have to be solved in a very early stage of the design. Rudder roll stabilization has been successfully applied on naval as well as merchant marine ships.[15]

1.4.2 Compensation of Nonlinear Effects in a Linear Motor

Many mechanical systems suffer from nonlinear effects that limit the accuracy that can be achieved. Friction and cogging are two examples. A (linear) feedback controller can diminish the influence of non-linearities, but complete compensation may be difficult. For systems that perform repetitive motions, an Iterative Learning Controller can help to further improve the performance.[16,17] The basic idea is explained in Figure 1.22.

When only the feedback loop is present and under the assumption that there are no disturbances, the error signal and thus the controller signal U_C will be the same for each repetitive motion. It is obvious that the accuracy can be improved when in the next motion the controller signal from the former cycle is used as a feed-forward signal, U_F. The feedback will generate a signal that further compensates for the remaining error by updating the feed-forward signal U_F with the formula

$$U_F^{k+1} = U_F^k + LE^k \tag{1.8}$$

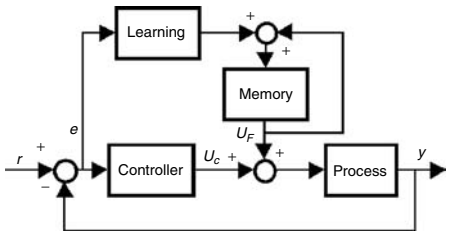


FIGURE 1.22 Principle of iterative learning control.

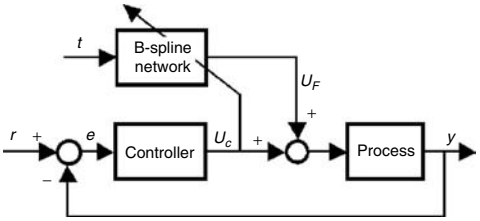


FIGURE 1.23 Learning feed-forward controller for repetitive motions.

where L is the transfer function of the learning filter. The superscript k denotes the kth repetitive motion. The signal U_F should converge to a feed-forward signal that compensates for all repetitive errors. An example of a situation where such errors are present is, for instance, a CD player that has to compensate for the eccentricity of the disk.

A variation on this idea and even more straightforward is the learning feed-forward controller (LFFC) setup of Figure 1.23. When the feed-forward signal would be perfect, the output of the controller would be zero. This implies that this output can be used as a training signal for a neural network. An adaptive B-spline network enables learning of complex nonlinear characteristics. Also support vector machines have been used to implement the learning feed forward.[18] The input of the B-spline network is the time t. It is reset each time a new motion starts. This is called a time-indexed LFFC. Instead of the time, also the reference signal and its derivatives—obtained from a path generator—could be used as index for the network (path-indexed LFFC). The advantage of this structure is that after proper training the LFFC can successfully be used for nonrepetitive motions as well. Velthuis has given a stability analysis for time-indexed as well as path-indexed LFFC.[19] The stability analysis is relatively easy for the time-indexed case. For the path-indexed case it is more complex and some heuristics are required to guarantee a stable system. The main issue is that the number of B-splines should not be too large. On the other hand a sufficiently dense B-spline distribution is desired for an accurate approximation of the nonlinear process. LFFC has successfully been applied to compensate for cogging in an industrial Linear Motor [20] and for compensation of (Coulomb) friction of a linear motor used in a flight simulator.[19] It has also been applied to the tracking control of the mobile robot described in the section Design of a Mobile Robot.[21]

The application to cogging compensation of a linear motor will be described in a little bit more detail. Such a motor is a commonly used element in assembly machines. Even with the best magnets and accurate assembly the error could not be made smaller than 100 μ, with a PID controller in combination with nonlearning feed-forward control. The design goal was to improve the maximally achievable accuracy from 100 μ to less than 10 μ. Figure 1.24 shows a picture of a linear motor.

According to the structure of Figure 1.23 the linear motor is controlled by means of a PID controller, while a B-spline neural network is present to learn the inverse motor model, including the nonlinearity due to cogging. Cogging occurs in DC motors with permanent magnets. It causes more or less sinusoidal shaped forces that depend on the position of the translator with respect to the stator. If these forces really had a sinusoidal shape, they would be easy to compensate for by means of a feed-forward compensator. However, this would require magnets with completely similar magnetic properties and very accurate spacing of the magnets. An alternative is to design a controller that learns the disturbance pattern and compensates it by means of a learning feed-forward compensator. An additional advantage is that such a system can also be used to compensate for other nonlinear effects, such as friction. This has also been demonstrated in a part of a flight simulator (a control stick) where friction forces spoil the feeling of a realistic simulation especially at almost zero speed. Figure 1.25 shows that learning is almost completed after six training cycles.

Learning feed-forward control is an attractive method to compensate for nonlinearities that are present in mechatronic systems, such as cogging and friction. The use of B-spline neural networks results in fast convergence, relatively low computational effort, and a good generalizing ability. Because of recently obtained results with respect to the stability of such systems, robust control systems can be designed.

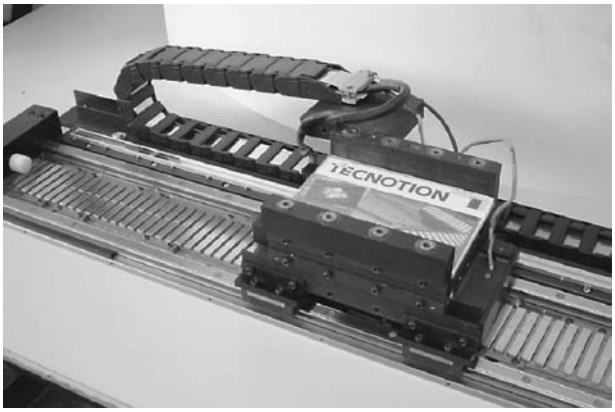


FIGURE 1.24 Linear motor. The magnets that cause the cogging are clearly visible.

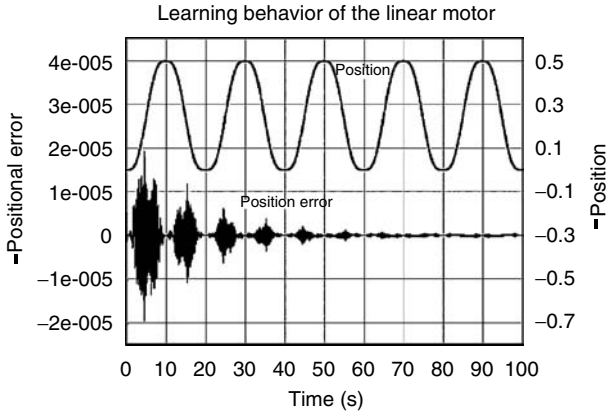


FIGURE 1.25 Position and error signal during learning of the LFFC.

A mechatronic view on this design problem raises the question whether it is possible to use the same techniques to build a less expensive linear motor, when maximum accuracy is not the main goal of the design. It has been demonstrated that a motor constructed with less expensive components and less demanding assembly specifications but with LFFC can compete well with the more expensive construction. The accuracy can typically be improved by a factor 10.

1.5 Special Requirements of Mechatronics that Differentiate from "Classic" Systems and Control Design

The main difference between "ordinary controller design" and mechatronic system design is that the latter deals with the design of the system as a whole. This approach can be considered as optimization of all components of the system simultaneously, although there are no algorithms to do this automatically. In practice the problem is often split into smaller problems that can be optimized. After integration of all the partial solutions a suboptimal system is achieved that can be further optimized by retuning the different parts, taking into account the already available intermediate design of the overall system. In order to achieve optimization of the system as a whole, it is desired that the mechanical part, where power plays a role, and the information processing part (the controller) can be simulated and adjusted simultaneously. This requires that mechanical parameters like masses and compliances be available in simulations of the

controlled system. Examples have been given of modeling and simulation with 20-sim that allows for such an approach.

Mechatronic designers should constantly be aware of the fact that solutions can be found in different domains. Not every mechanical deficiency can easily be solved by control. A good mechanical design may be easier and cheaper to achieve. On the other hand, a good controller may be able to achieve the desired performance much easier and cheaper than a complex mechanical construction. In some cases the combination can even achieve performances that would never have been possible without a mechatronic design.

The same holds for the design of sensors. Each sensor could be fitted with a filter to remove noise from the measurements. But if several sensors are being combined, sensor fusion in a Kalman filter algorithm will benefit from the availability of the raw data.

Communication between all the designers involved and transparency of the design decisions in the various domains are essential for the success of a true mechatronic design.

References

1. Matworks, *The Mathworks: Developers of MATLAB and Simulink,* 2000, www.mathworks.com
2. Coelingh, H.J., *Design Support for Motion Control Systems,* Ph.D. thesis, University of Twente, 2000, also www.rt.el.utwente.nl/clh/
3. Coelingh, H.J., de Vries, T.J.A., van Amerongen, J., Design support for motion control systems—application to the Philips fast component mounter, in *Mechatronics Forum 7th Int. Conf., Mechatronics 2000,* Atlanta, Ga, USA.
4. Controllab Products, *20-sim*, www.20sim.com
5. Broenink, J.F., *Computer-Aided Physical-Systems Modeling and Simulation: a Bond-Graph Approach,* Ph.D. thesis, University of Twente, 1990.
6. Dynasim, *Dymola,* www.dynasim.se/
7. Weustink, P.B.T., de Vries, T.J.A., Breedveld, P.C., *Object Oriented Modeling and Simulation of Mechatronic Systems with 20-sim 3.0,* in *Mechatronics 98,* J. Adolfson and J. Karlsén (Eds.), Elsevier Science, 1998.
8. De Vries, T.J.A., *Conceptual Design of Controlled Electro-Mechanical Systems,* Ph.D. thesis, University of Twente, 1994.
9. Breedveld, P.C., Fundamentals of bond graphs, in *IMACS Annals of Computing and Applied Mathematics, Vol. 3: Modelling and Simulation of Systems,* Basel, 1989, pp. 7–14.
10. Cellier, F.E., Elmqvist, H., Otter, M., Modeling from physical principles, in *The Control Handbook,* W.S. Levine (Ed.), CRC Press, 1996, pp. 99–108.
11. Gawthrop, P., Lorcan Smith, L., *Metamodelling: Bond Graphs and Dynamic Systems,* Prentice-Hall, NJ, 1996.
12. Van Amerongen, J., Modelling, simulation and controller design for mechatronic systems with 20-sim 3.0, in *Proc. 1st IFAC Conf. on Mechatronic Systems,* Darmstadt, Germany, September 2000, pp. 831–836.
13. Groenhuis, H., *A Design Tool for Electromechanical Servo Systems,* Ph.D. thesis, University of Twente, 1991.
14. Van Amerongen, J., Coelingh, H.J., de Vries, T.J.A., "Computer support for mechatronic control system design," *Robotics and Autonomous Systems,* vol. 30, no. 3, pp. 249–260, PII: SO921-8890 (99)00090-1, 2000.
15. Van Amerongen, J., van der Klugt, P.G.M., van Nauta Lemke, H.R., Rudder roll stabilization for ships, *Automatica,* vol. 26, no. 4, pp. 679–690.
16. Arimoto, S., A brief history of iterative learning control, in *Iterative Learning Control: Analysis, Design, Integration and Applications,* Kluwer Academic Publishers, 1988, pp. 3–7.
17. De Vries, T.J.A., Velthuis, W.J.R., van Amerongen, J., Learning feed-forward control: a survey and historical note, in *1st IFAC Conf. on Mechatronic Systems,* Darmstadt, Germany, September 2000, pp. 949–954.

18. De Kruif, Bas J., de Vries, T.J.A., On using a support vector machine in learning feed-forward control, in *Proc. 2001 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics,* Como, Italy, 8–12 July, 2001.
19. Velthuis, W.J.*R., Learning Feed-Forward Control—Theory, Design and Applications,* Ph.D. thesis, University of Twente, 2000, also http://www.rt.el.utwente.nl/vts/
20. Otten, G., de Vries, T.J.A., van Amerongen, J., Rankers, A.M., Gaal, E., Linear motor motion control using a learning feedforward controller, *IEEE/ASME Transactions on Mechatronics,* vol. 2, no. 3, ISSN 1083-4435, 1997, pp. 179–187.
21. Starrenburg, J.G., van Luenen, W.T.C., Oelen, W., van Amerongen, J., Learning feed-forward controller for a mobile robot vehicle, *Control Engineering Practice,* vol. 4, no. 9, 1996, pp. 1221–1230.

2

The Role of Modeling in Mechatronics Design

Jeffrey A. Jalkio
University of St. Thomas

If mechatronics design is more than just the combination of electronic, software, and mechanical design, the additional feature must lie in the ability of the mechatronic designer to optimize a design solution across these disparate fields. This requires a sufficient understanding of each of these fields to determine which portions of an engineering problem are best solved in each of these domains given the current state of technology. In turn, this requires the ability to model the problem and potential solutions using techniques that are domain independent or at least permit easy comparison of solutions and tools from different domains.

For example, the optical inspection system shown in Figure 2.1 depends on optical components in precise alignment, mechanical elements capable of precise motion, transducers for sensing and providing mechanical power, electrical systems to control motion and filter sensor signals, and software for image analysis and motion control. Only by dividing these tasks appropriately among electronics, mechanical components, and software can the system be optimized. This requires an understanding of all the system requirements and limitations as well as the capabilities of each component in the various domains. Modeling of requirements and systems is crucial in determining whether a proposed solution is acceptable as well as in documenting these determinations for future use. In this article we shall examine the varieties of models used at different points in the design process, the diverse roles of these models and their relative strengths and weaknesses in each of these roles, and finally the specific tradeoffs involved in choosing dynamic models for signals and systems analysis.

2.1 Modeling as Part of the Design Process

Models serve different purposes at different points in the design process; so to decide which modeling tools are most effectively employed in different phases we must examine the design process itself. Many descriptions of the design process are available that have been developed by researchers around the world.[1–3] Typically these descriptions serve to systematize the process to improve the productivity of

FIGURE 2.1 An optical inspection system for printed circuit boards. (Used by permission © CyberOptics Corporation 2001, all rights reserved.)

designers or to describe techniques that provide improved product quality, lower cost, or other benefits. However, since our purpose is to examine the modeling needs of the design process, we can consider a simple model that distinguishes phases of the design process in terms of types of design activity rather than a more complex model that may be preferable for other purposes. For this purpose, we can consider a four-phase process consisting of requirements analysis, concept generation, analysis and selection, and detailed design. In the first phase of this process the designer focuses on analysis of the problem without considering possible solutions. In the second phase, conceptual solutions are generated with the hope that an acceptable solution can be found from these initial concepts via combination or modification of concepts or by variation of parameters present in one of the conceptual solutions. In the third phase, these concepts are evaluated and a design is chosen for implementation. The fourth phase consists of identifying design problems that need to be solved to implement the chosen concept and applying the design process to those smaller problems. We shall consider the activities of each of these phases in detail.

2.1.1 Phase 1

The requirements analysis phase consists in obtaining a sufficient understanding of the problem to be solved. The difficulty of this process varies with the scale of the problem, the designers' familiarity with the problem domain, the variability of market needs, and the presence of hidden requirements that are poorly articulated in the initial problem statement. Depending on the nature of the design problem, the requirements identified in this phase may be the needs of a single customer, the common needs of a group of potential customers identified via a market survey, or societal needs identified by government regulations. Most design problems include some combination of these as well as internal requirements such as design guidelines and company policies. The key objective of this phase is to obtain enough detail to know when a design has solved the problem satisfactorily. Models in this phase serve primarily as communication and documentation tools since the primary problem in this phase is the clear communication and documentation of the criteria for design success. Examples of models that aid in this process include specification listings, use case diagrams, sequence diagrams, and context diagrams. Many of these modeling tools have now been standardized as parts of the Unified Modeling Language (UML), which is becoming increasingly important as an analysis tool.[4]

Use case diagrams model the interactions between a system and its users at a very high level of abstraction in terms of purpose of the interaction. Figure 2.2 gives an example of a use case diagram used to document the various operations required of a network printer. The use case diagram helps us avoid overlooking important but rare use cases such as maintenance. It is important to note that use case diagrams do not

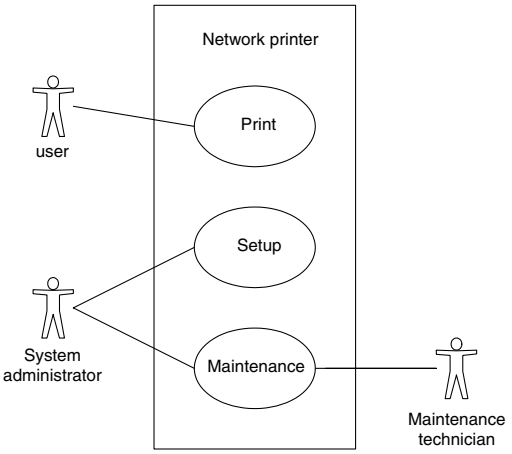


FIGURE 2.2 Use case diagram for a network printer.

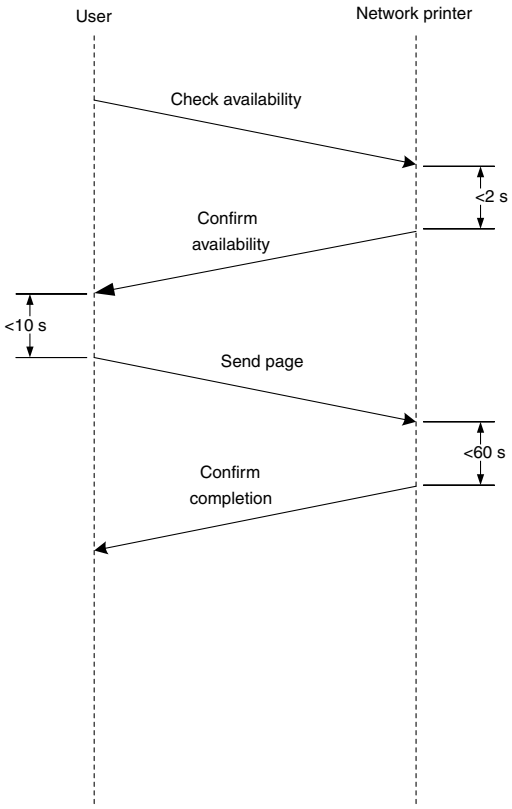


FIGURE 2.3 A sequence diagram for network printer use.

provide information about the nature of the transaction, but rather documents its existence. It serves as a top level model of a system only.

Sequence diagrams can be drawn to describe the details of individual use cases in order to clarify the interactions that must occur, timing constraints, and interactions between the system and multiple elements of the environment. Figure 2.3 shows an example of a sequence diagram for the "print" use case of Figure 2.2.

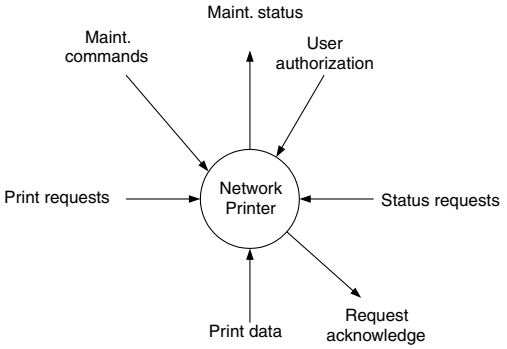


FIGURE 2.4 Context diagram for network printer.

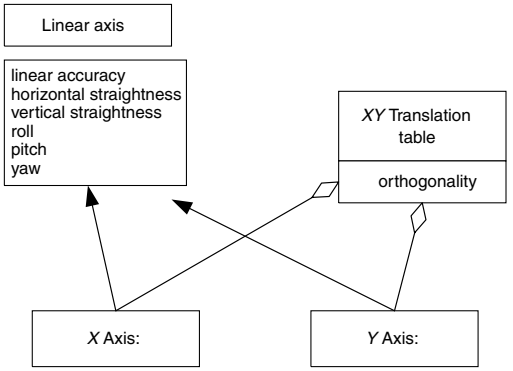


FIGURE 2.5 Class diagram for linear translation stages.

Note that the sequence diagram documents a particular instance of a particular use case. The sequence diagram can show both the direction of information flow and timing requirements. In this regard the sequence diagram and the traditional timing diagram serve similar purposes.

Context diagrams do not capture timing elements but capture the types of information that flow between the system and its environment.[5] While use case diagrams focus attention on the scenarios of system use, context diagrams focus attention on information flows that must exist to enable those scenarios. Both serve useful and complementary purposes. Figure 2.4 shows a context diagram for an inspection system like the example mentioned above. Notice that the context diagram summarizes information flow shown in the various interaction diagrams for all use cases. Regulatory and safety requirements are often in the form of limits on the interactions the system may have with aspects of its environment. These requirements must all be captured for use in later design phases and communicated back to the source of the requirement for verification of accuracy. The context diagram is also used as the top level of a data flow diagram for a system if structured analysis is used.[6]

Just as the context diagram ties to structured analysis techniques, object oriented analysis techniques provides the class relationship diagram as a means of documenting the relationships between a system and its environment and between components of a system.[7] Class relationship diagrams show how a system is composed of subsystems, how components are similar to one another, and how they differ. For example, Figure 2.5 shows a class relationship diagram for a two-axis translation system consisting of two single axis subsystems. This diagram documents the existence of 13 error components that must be specified in the requirements phase. It does this by showing that the *X* and *Y* axis components are instances of a class of single axis translation stages, that each have six error components, and that they are components of a

system that adds a single additional error. By documenting relationships such as composition and inheritance, requirements and the interdependence of requirements can be represented in a compact and comprehensible way.

One key aspect of the requirements definition phase is the importance of defining requirements without specifying a preferred solution embodiment. Hence, modeling methods should be chosen to document these requirements and their intrinsic relationships without implying a particular solution.

2.1.2 Phase 2

In the concept generation phase, our objective is to generate multiple design concepts that might satisfy the requirements identified in phase 1. Here we need modeling techniques that allow us to describe possible solutions with varying levels of detail dependent on the degree of detail needed to document the key elements of the concept. Since individual concepts generated in this phase may only satisfy some portions of the design requirements, it is critical that modeling at this point allow for partial descriptions of embodiments and for the easy combination of design concepts. For this reason, our models must clearly document the portions of the requirements satisfied as well as any unspecified parameters or additional requirements introduced in the concept. For some problems, block diagrams showing interconnections between components solving portions of the problem are a useful modeling tool at this stage. Figure 2.6 shows two possible block diagrams describing a given design concept. The first specifies a particular control algorithm, sensor, and actuator, while the second leaves these particulars unspecified and simply describes a closed loop controller. Depending on the situation either of these may be appropriate descriptions. The first provides details and is closer to a complete design while the second, being more generic, is easier to combine with other concepts to generate hybrid solutions.

Block diagrams are not the only modeling tool appropriate at this stage. For other problems, schematics showing arrangements of components or equations or psuedocode of proposed algorithms may be employed. As this phase continues, design concepts are often combined to form potential solutions to the overall design problem; therefore, it is useful if the model of each concept can contain descriptions of preconditions for its use, results, and other parameters that help a design team determine how to combine concepts. Similarly, once potential solutions are formed by concatenating concepts, it is often useful to clean up the final concept by combining features from different component concepts or eliminating overlapping features that are no longer needed.[8] This process is facilitated by modeling techniques that allow simultaneous modeling of mechanical, thermal, electrical, and software components and concepts. These techniques include the familiar linear graph and bond graph models.[9,10]

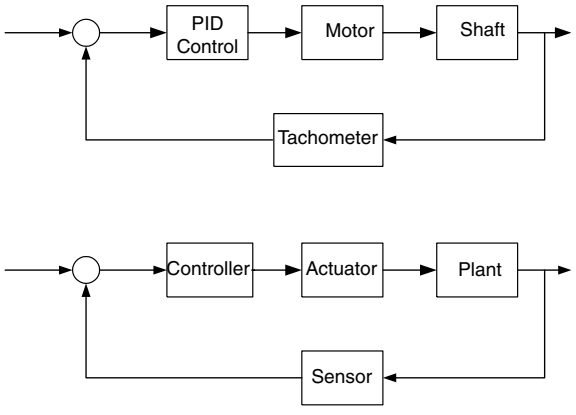


FIGURE 2.6 Block diagrams at two levels of detail.

2.1.3 Phase 3

In the third phase we evaluate potential solutions in terms of the problem requirements. This phase is eased if we have a model that allows us to compare problem requirements with design features. A particular design methodology may include various quality criteria in addition to those set by customer requirements and the regulatory environment, for example, the design axioms of Suh prefer designs with minimum information content and which satisfy each requirement by independent features of the design.[11]

One approach to evaluation is to attempt to find numerical criteria for all requirements and evaluate solutions via minimization of an overall cost function (or equivalently, the maximization of an overall value function). This approach has proven effective in certain types of problems where requirements are amenable to quantification (e.g., the optimal solution being the one that meets requirements with minimal weight or economic cost). Even in these cases, it can be difficult to determine the relative importance of diverse requirements in the formulation of the value function. For example, it may be desirable for a design to have minimal parts cost and minimal weight. An appropriate cost function might be

$$\Psi = \sqrt[n]{\left(\frac{c}{c_0}\right)^p + \left(\frac{w}{w_0}\right)^q}$$

where c and w are the cost and weight, respectively, and c_0 and w_0 are scaling factors indicating when the two factors are of equal importance, while the exponents p, q, and n express the relative importance of minimizing the two factors. Clearly, there are many possible choices for these parameters and many other models that can be used. This highlights the difficulties of this method. An example of this difficulty in a relatively simple case is determining the cost of a component being out of tolerance. Models for this problem range from a step function of cost = 0 within tolerance and cost = C outside tolerance to Taguchi's quadratic cost function.[12] Needless to say, the "optimal" solution found is typically dependent on the cost function chosen. On the other hand, in some cases, the optimum found is not strongly dependent on precise choice of cost function and a satisfactory evaluation can be made without expending excessive effort to obtain an exact cost function. In fact, if there are relatively few design options from which to choose, it is possible to invert this problem. Rather than finding the design that minimizes a particular cost function, one can instead find the range of cost function parameters that result in any particular solution being optimal. It is often easy to determine which range of parameters is most realistic.

2.1.4 Phase 4

The fourth design phase is detailed design, in which the entire process is repeated to resolve open design details for the individual components of the resulting design. This is in keeping with the design heuristic "allocate resources as long as the cost of not knowing exceeds the cost of finding out."[13] The process is inherently recursive, with each high level design decision producing a simpler design problem at a lower level of abstraction with simpler requirements. To accommodate this, we need to be able to model the design at multiple levels of abstraction and to allow the specification of interfaces between components. These models must allow us to define static as well as dynamic (behavioral) components of the interface.

In addition to this recursive aspect of design, there is of course iteration also possible, as dead ends are discovered in an otherwise promising chain of design decisions and as changes in the marketplace demand revisions of a product. To accommodate this iteration, design models should document not only decisions made, but also reasons for those decisions. Otherwise, the reasoning must be rediscovered in each iteration, or worse, may not be rediscovered until it is too late.

2.2 The Goals of Modeling

We have seen that models serve many purposes in mechatronics design. First, models serve to document concepts, assumptions, and requirements and allow the communication of these concepts to others in the design group and the various stakeholders of the design process. Second, models provide a hierarchical framework which allows division of labor in the design process and permits concurrent work on separate components.

Third, models can provide us with insights into the behavior of a system which might be obscured when we attempt to see the system in its full complexity. Fourth, they allow us to grasp similarities with other systems and use prior experience to help solve current problems. Fifth, models provide us a way of identifying and testing our ignorance of the actual behavior of the system by identifying unknown parameters and providing hypotheses worthy of testing. In this section, we shall consider these diverse purposes of modeling and examine the characteristics that cause a particular modeling methodology to be better or more poorly suited for each of these purposes.

2.2.1 Documentation and Communication

The first purpose of a model that we shall consider is documentation. We often forget that aside from the various purposes of modeling in the design process, our models are often our best tools for communicating with our colleagues, our customers, and our successors. Every document generated in the design process is a model of some part of the system. To the extent that we can avoid double documentation of the same concept, we not only reduce workload and decrease time to market, but we also reduce the likelihood of inconsistencies in our documentation. How do our models serve as documentation? Consider a circuit diagram. It is not the circuit, but documentation of the circuit's design. It is also a model of the behavior we are seeking from the circuit. Nonidealities and component variations will cause the actual circuit to behave differently from this model, but the model still serves its purpose. Earlier in the design process, requirements documents also document the product, but in terms of requirements rather than the features that satisfy them.

Design documentation is important for at least four distinct groups. The engineering team itself needs communication among its members to ensure that effort is not wasted on elements that do not interoperate or that do not contribute to the overall product meeting requirements. Communication is required between the design team and the customer to ensure that requirements are correctly understood and implemented in the design. Management needs to understand the status of the project in terms of outstanding risks, ongoing expenses, and tradeoffs that would affect market size. Finally, future design teams can benefit from documentation of earlier design concepts, analyses, and decisions. This role of documentation as a communication with future workers in the field has been recognized as a key part of all scientific disciplines[14] but is often overlooked by engineers focused on meeting short-term deadlines.

What characteristics of a model lead to good documentation and clear communication? Clearly, the model must be understood by others. This requirement encourages the use of standardized modeling techniques whenever possible. It also encourages the use of modeling techniques that are interdisciplinary whenever feasible. When choosing models for documentation purposes, one must remember that each model highlights certain features of the design while hiding others. For example, the sequence diagram clearly documents communication between two objects while hiding details of how the objects generate outgoing messages or interpret incoming ones. If multiple models of the system are used to document the various aspects of the system, there should be some way to cross-reference items between models (e.g., determine that timing requirements documented in a sequence diagram model are met by the dynamics of a system described in a block diagram). Also, if the documentation is to be useful to future designers, it should be possible to cross reference design models based on a variety of criteria including system requirements and included components, so that future designs can benefit from current ones. Clearly, any modeling technique that serves some other purpose in design will also serve as documentation, but the model must be chosen so as to match the features one wishes to document.

2.2.2 Hierarchical Framework

Real mechatronic systems include a large number of components that interact in many ways. Our models simplify these complex interfaces and describe products and processes in terms of simply interacting components with well-defined interfaces and behaviors. This simplification allows us to subdivide complex problems into a set of simpler problems whose solutions can be easily integrated. In complex mechatronic systems where we must consider components that interact in several energy domains, this

simplification is essential. When we draw a block diagram of a system we are dividing the system's behavior into a set of defined elements with modeled behavior and a known set of interactions between those elements, which is itself amenable to analysis. This role of modeling would lead us to select modeling techniques that allow us to divide the model into portions that can be independently tackled by independent engineers. For example, systems of linear differential equations may give us great insight into the fundamental modes of a system, but give us little insight on how to divide a design problem among members of a team. However, a block diagram, a class diagram, or a data flow diagram provides clear interfaces between elements that allow for subdivision.

2.2.3 Insights

Regarding insights into the behavior of the system, different models provide different types of insights. Signal flow graphs of electromechanical systems show the presence of feedback loops that stabilize or destabilize the system. Differential equations provide insights regarding the time scales of various behaviors and the relative importance of various factors as well as providing estimates of the validity of simplifying assumptions. Similarly, sequence and timing diagrams can provide insight regarding the processing power required to meet timing requirements for various use cases. In each case the model provides insights into the problem or the characteristics of a proposed solution by bringing into focus certain aspects of the modeled system while hiding others from view.

2.2.4 Analogies

Related to the insights a model can give us regarding the system under consideration are the insights that a model can provide in allowing us to see similarities to other systems that we or others have seen before. In this regard, modeling techniques that use analogies between various domains can be useful. However, analogies must be chosen with care. Consider the common analogy between electrical and mechanical quantities shown in the left half of Figure 2.7. In this analogy, velocity is analogous to voltage and force is analogous to current. However, as seen in the right half of Figure 2.7, the equations for the electrical system are unchanged in the dual system, in which current and voltage are exchanged and the roles of inductance and resistance are exchanged with capacitance and conductance, respectively. This dual system results in a different mechanical analogy, in which velocity is analogous to current and force is analogous to voltage. Each of these analogies can prove useful in moving design parameters from one energy domain to another, as the duality between the two electrical models can prove useful in circuit design. With any of these analogies, it must be remembered that real components deviate substantially from their idealized models and that the analogies do not strictly hold. This can be both a bane and a blessing, since a design that suffers from component nonidealities might be replaced by an analogous design from a domain with different but less detrimental nonidealities.

Element	Electrical	Mechanical	Electrical	Mechanical
Capacitance	$i = C\frac{dv}{dt}$	$f = M\frac{dv}{dt}$	$v = \frac{1}{C}\int i dt$	$f = k\int v dt$
Inductance	$i = \frac{1}{L}\int v dt$	$f = k\int v dt$	$v = L\frac{di}{dt}$	$f = M\frac{dv}{dt}$
Resistance	$i = \frac{1}{R}v$		$v = Ri$	$f = Bv$
Conductance	$i = Gv$	$f = Bv$	$v = \frac{1}{G}i$	

FIGURE 2.7 Electrical–mechanical analogies.

2.2.5 Identification of Ignorance

The final role of modeling is to help us identify our ignorance. This takes two forms. First, the construction of the model often requires us to name parameters whose numerical value is unknown. This leads us to estimate the possible range of such parameters or to design experiments to determine the parameter's value. Identifying missing details is the first step in determining them. In this way models help us identify our ignorance of details of the problem or our proposed solution. Secondly, the predictions we make with the help of our models are testable. The adequacy of a model to describe system behavior can be determined by comparing actual system behavior with predictions based on that model. Differences between actual system performance and a model's predictions point to errors in the model that may be significant. By testing the validity of our models we identify aspects of the real system that are inadequately reproduced in our model or artifacts of the model that do not exist in the real system. These discrepancies may point to an inappropriate assumption or to a fundamental misunderstanding of the physical system. In either case, the source of these discrepancies should be understood in order to determine if our model is adequate for its intended purpose.

2.3 Modeling of Systems and Signals

In the area of systems and signals we typically deal with the modeling of system dynamics and information flows through the system and its environment. As can be seen from the discussion above, this is but a small part of the modeling needed for system design. However, there are a number of decisions that must be made in selecting a model for the behavior of a dynamic system. These include the choice between analytic and numerical modeling, the use of distributed or lumped parameters, the approach used to model random factors, and the choice between linear and nonlinear models. These issues are considered in the paragraphs below.

2.3.1 Analytical versus Numerical Models

In this area both analytical and numerical tools are available to support our efforts, so we have a number of decisions to make regarding the level of complexity of the models we choose to use for various tasks. Analytical tools provide insights into overall system behaviors and can allow us to make comparisons between dissimilar systems based on a similarity in their models. These analogous systems often prove useful in the concept generation phase of design because they allow us to bring to bear solutions to diverse problems to the solution of the problem at hand. However, as described above, these analogies usually break down upon closer inspection. In the case of analogies in dynamics, they are typically based on systems described by equivalent differential equations (e.g., electronic and mechanical oscillators). However, these linear differential equations are but simplifications that ignore nonlinearities and time-dependent behavior that may greatly affect system behavior. Therefore, when applying analogies one must always be careful to explicitly state assumptions made in the modeling process.

Numerical tools, on the other hand, do not yield the same insights of analytical tools, but provide other and more detailed insights into real system behavior and performance. Typically these tools become most useful later in the design process, both for tradeoff analysis and for detailed design work. The availability of these tools opens many doors to modeling options that have in the past been closed and allows us to consider options that have historically been considered unwieldy. For example, numerical solution of nonlinear differential equations allows modeling of systems that had previously been approached only with linear approximations.

2.3.2 Partial versus Ordinary Differential Equations

One modeling choice that must be made is whether we must model the dynamics of the system using ordinary or partial differential equations. Partial differential equations must be used whenever values of interest vary spatially as well as temporally. For example, if we are considering the design of a robot arm, we are free to use ordinary differential equations if we consider the arm to be rigid, partial differential

equations if we wish to describe the bending of a flexible arm in detail. However, what if we recognize a flexibility in the arm but do not need to know the details of its motion but only the effect of its flexure on the end effector? In this case we can make a lumped parameter model of the arm that summarizes its dynamics in terms of end effector motion and dynamic forces on the driver and end effector. When can this lumped parameter model be used? There are two restrictions. First, the details of the variation of behavior over space must be uninteresting to us. If we need to know these details (e.g., to determine stress concentration, or temperature distribution) we cannot apply the simplified model. Secondly, the partial differential equation may not be amenable to forming a lumped form due to its complexity.

2.3.3 Stochastic versus Deterministic Models

One choice that must be made in the modeling of system behavior is how uncontrolled variability will be represented in the model.[15] The approach taken depends a great deal on the source and characteristics of the variability. For example, the values of some system parameters may not be precisely known, but may be constant over time. Others may vary slowly over time in unknown ways. In the first case, the sensitivity of the system to parameter value can be determined, using a variety of both analytical and numerical techniques. In the latter case, one must examine the speed of parameter variation to determine if the system can be analyzed in quasi-steady state or if the dynamics of the parameter variations are coupled with the dynamics of the system. For this purpose, it is often useful to write equations in dimensionless form where time and scale constants indicate critical speeds for coupling.

When the variation of parameters over time must be considered or when some inputs to a system have uncontrolled variability, we must choose whether to model the input as an arbitrary signal or a signal with known statistical parameters. In the first case, we have just another input and can analyze the system for sensitivity to this input. In the second case, we can characterize the system in terms of autocorrelation or equivalent spectral measures and use optimal filtering techniques to reduce uncertainty.[16] A key point in the statistical modeling of a system is to base the model on the known variability of the process rather than assuming additive, white, Gaussian noise at every turn.[17]

2.3.4 Linear versus Nonlinear

A wealth of techniques are available for the analysis of linear time invariant systems. Unfortunately, the world gives us nonlinear components. One advantage of modeling techniques that allow us to divide a complex system into simpler sub-components is that we can often isolate nonlinearities as individual elements and develop linear models for them. For small variations about a differentiable point on a nonlinear curve, we can always find a linear model, but the value of this model might be extremely limited depending on its accuracy over a reasonable range of input values. In the case of a discontinuous or nondifferentiable nonlinearity, there is a greater problem. A linear model can be found only when we are far from the discontinuity or when the discontinuity is small compared to the linear portion of the model.

However, nonlinearities also add essentially new behaviors to systems that are not possible in purely linear systems. For this reason alone, nonlinear models are needed in certain circumstances. As an example, for a linear system, stable oscillations are only theoretically possible for differential equations with purely imaginary eigenvalues. Positive real parts lead to growing oscillations, while negative real parts lead to damped oscillations. Clearly, the behavior of such a system is highly sensitive to system parameters. Small variations will either kill the oscillation or drive the system into nonlinear regions of operation. Furthermore, even for perfect oscillators, the amplitude of oscillations is unconstrained. However, a nonlinear system such as the Van der Pol equation:

$$\ddot{y} = -\omega^2 y + \alpha\left[1 - \left(\frac{y}{y_0}\right)^2\right] y$$

includes a damping term that increases with the magnitude of the oscillation. For oscillations small compared to y_0 the oscillations grow, while larger oscillations are damped. This and other essentially

nonlinear behavior cannot be modeled adequately with linear models. Although once this behavior is recognized through the use of a nonlinear model, the resulting steady-state system (a stable oscillator) can be modeled for many purposes using its linear (although physically unrealizable) equivalent.

When using differential equations to model a system, we must be particularly careful in our choice of units of measure and in how we group parameters. For example, in the Van der Pol equation above, we can choose an alternative scale for y such that y_0 is 1 and a time scale such that ω is 1. In this case we have

$$z'' = -z + \varepsilon(1 - z^2)z'$$

where $z = y/y_0$, $\tau = \omega t$, and $\varepsilon = \alpha/\omega$. This scaling provides a dimensionless equation as well as providing an indication of the system's natural time scale, the scale of y, and an indication that the size of α relative to ω is important. In general, we find that writing equations in dimensionless form provides three advantages. First, as mentioned above, the scaling factors provide a sense of time scale and magnitude of various features of the system's dynamic behavior. Second, the dimensionless parameters that result from combining physical system parameters provide a way of characterizing the behavior of a wide range of systems in terms of parameters that are easily mapped. Third, by putting the equation into dimensionless form, we are able to categorize it and recognize similar equations in different domains.

References

1. Hubka, V. and Eder, W.E., *Engineering Design—General Procedural Model of Engineering Design,* Heuista, Zurich, 1992.
2. Pahl, G. and Beitz, W., *Engineering Design: A Systematic Approach,* Springer-Verlag, Berlin, 1988.
3. Dym C.L., *Engineering Design—A Synthesis of Views,* Cambridge University Press, New York, 1994.
4. Booch, G., Jacobson, I., Rumbaugh, J., and Rumbaugh, J., *The Unified Modeling Language User Guide,* Addison-Wesley, New York, 1998.
5. Douglass, B., *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns,* Addison-Wesley, New York, 1999.
6. Demarco, T., *Structured Analysis and System Specification,* Yourdon Press, NJ, 1978.
7. Coad, P. and Yourdon, E., *Object-Oriented Analysis,* Yourdon Press, NJ, 1990.
8. Glegg, G.L., *The Design of Design,* Cambridge University Press, London, 1969.
9. Shearer, Murphy, and Richardson, *Introduction to Dynamic Systems,* Addison-Wesley, Reading, MA, 1967.
10. Karnopp, D. and Rosenberg, R., *System Dynamics: A Unified Approach,* Wiley, New York, 1975.
11. Suh, N.P., *Principles of Design,* Oxford University Press, New York, 1990.
12. Taguchi, G. and Yokoyama, Y., *Taguchi Methods: Design of Experiments,* American Supplier Institute, Dearborn, MI, 1994.
13. Koen, B.V., *Definition of the Engineering Method,* American Society for Engineering Education, Washington, 1985.
14. Lonergan, B., *Method in Theology,* University of Toronto, Toronto, 1990.
15. Zhou, K., *Essentials of Robust Control,* Prentice-Hall, NJ, 1998.
16. Papoulis, A., *Probability, Random Variables, and Stochastic Processes,* McGraw-Hill, New York, 1991.
17. Law, A. and Kelton, W., *Simulation, Modeling and Analysis,* McGraw-Hill, New York, 1991.

3

Signals and Systems

Momoh-Jimoh Eyiomika Salami
International Islamic University of Malaysia

Rolf Johansson
Lund Institute of Technology

Kam K. Leang
Qingze Zou
Santosh Devasia
University of Washington

C. Nelson Dorny
University of Pennsylvania

3.1 Continuous- and Discrete-Time Signals

Momoh-Jimoh Eyiomika Salami

Signals are physical variables or quantities measured at various parts of a system, which when processed yield the desired information. A wide variety of signals are often encountered in describing many practical systems. Electrical signal, in form of current and voltage, is the most easily measured quantity, hence the need to use sensors and transducers to transform other non-electrical quantity into electrical signals. These signals must be processed by appropriate techniques if desirable results are to be obtained. Several methods of signal representation, suitable for effective signal processing in both time and frequency domains, are discussed in this section.

3.1.1 Signal Classification [1–4]

Signals are broadly classified as either continuous-time (CT) or discrete-time (DT) signals, and each of these may in turn be categorized as deterministic or random signals. A deterministic signal can always

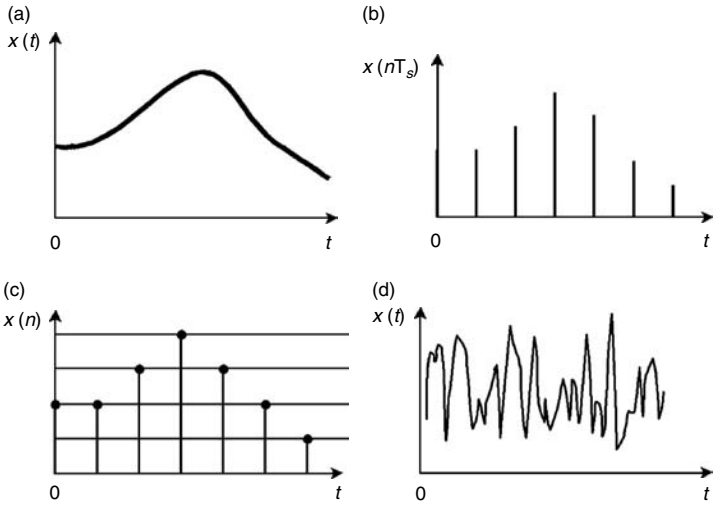


FIGURE 3.1 Description of some classes of signals: (a) continuous-time analog, (b) sampled-data, (c) digital signal, (d) random signal.

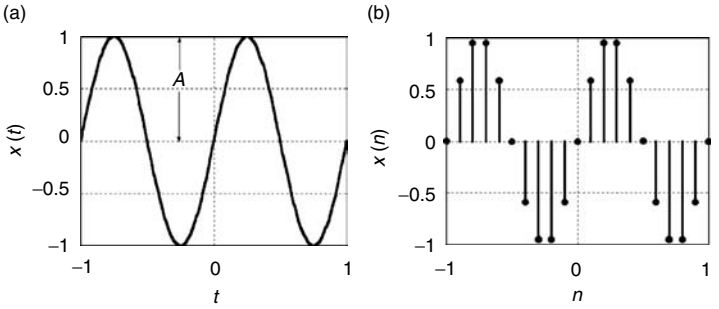


FIGURE 3.2 Description of periodic signals: (a) CT, (b) DT.

be expressed mathematically, whereas the time of occurrence or value of a random signal cannot be predicted with certainty. A CT signal, $x(t)$, has a specified value for every value of time, t, while a DT signal, $x(n)$, has specified a value only at discrete points, that is, for integer values of n. Closely related to CT and DT signals are analog and digital signals, respectively. If the amplitude of a signal can take on any value in a continuous range, then it is an analog signal. On the other hand, the amplitude of a digital signal can have only finite number of values at discrete points. Examples of continuous-time, discrete-time, digital, and random signals are shown in Figure 3.1.

Deterministic signals fall into two main categories, namely periodic and aperiodic signals. A periodic signal has the same values at times separated by one period, T, that is, $x(t)$ satisfies the relation, $x(t) = x(t + T)$, $-\infty < t < \infty$. An example of a CT periodic signal is a sinusoidal waveform of the form $x(t) = A\sin(\Omega t + \theta)$, where θ is the phase in radians, $\Omega = 2\pi F$ is the frequency in radians per second, and F is the frequency in hertz. It should be noted that the frequency range for the analog sinusoids is $-\infty < F < \infty$. A periodic DT signal is described by $x(n) = x(n + N)$, $-\infty, < n < \infty$, where N represents the period. An example of this is the sinusoidal waveform $x(n) = A\sin(2\pi rn + \theta)$, $-\infty < n < \infty$, where r is the signal frequency per sample frequency and has values in the range $-\frac{1}{2} \le r \le \frac{1}{2}$. Samples of sinusoids having frequencies within this range are unique and distinct. However, DT sinusoids whose frequencies are separated by an integer multiple of 2π are identical. Examples of analog and DT sinusoids are depicted in Figure 3.2.

Any deterministic signal that is not periodic is referred to as aperiodic or nonperiodic. Damped sinusoids and exponential decaying signals are common examples of aperiodic signals. For some applications, it may

be quite useful to classify the signals according to their energy or power content. The total energy over the range $t \in (-\infty, \infty)$ for a CT signal is given by

$$E = \lim_{T\to\infty} \int_{-T/2}^{T/2} |x(t)|^2 dt \tag{3.1}$$

and the average power is defined as

$$P = \lim_{T\to\infty} \left[\frac{1}{T} \int_{-T/2}^{T/2} |x(t)|^2 dt \right] \tag{3.2}$$

Consequently, $x(t)$ is an energy signal if and only if $0 < E < \infty$, which implies that $P = 0$. Similarly, $x(t)$ is a power signal if and only if $0 < P < \infty$, indicating $E = \infty$. A signal that fails to satisfy either definition is, therefore, neither energy nor power signal. These definitions are also applicable to DT signals except that the integral in Equations 3.1 and 3.2 is replaced by summation. In general, periodic signals exist for all the time and as such have infinite energy. However, they have finite average power, hence they are power signals. On the other hand, bounded finite-duration signals are energy signals. The classification of a signal to finite energy, finite power, or neither is important so that appropriate and effective procedures can be selected for its analysis.

3.1.2 Singularity Functions

Singularity functions are useful for signal modeling, that is, they serve as basis for representing complex signals to simplify their analysis.

3.1.2.1 The Unit Impulse Function

The impulse or delta function is a mathematical model for representing physical phenomena that occurs within very small time duration; this time duration can be assumed to be equal to zero. The unit delta function is not a mathematical function in the usual sense; rather it is a distribution or a generalized function. Thus, the impulse function can be described by its effect on the *test function* $\phi(t)$, that is,

$$\int_{-\infty}^{+\infty} \phi(t)\delta(t)\,dt = \phi(0) \tag{3.3}$$

provided $\phi(t)$ is continuous at $t = 0$. This equation shows the shifting property of the delta function. A graphical plot of the delta function is shown in Figure 3.3a. Table 3.1 shows the operating properties of the delta function.

3.1.2.2 The Unit Step Function

The unit step function is particularly useful for the mathematical analysis of CT signals. This is depicted in Figure 3.3b, and is defined as

$$u(t) = \begin{cases} 1, & t > 0 \\ 0, & t < 0 \end{cases} \tag{3.4}$$

TABLE 3.1 Properties of the Delta Function

Property	Mathematical Expression
Sampling	$\int_{-\infty}^{+\infty} x(t)\,\delta(t-a)\,dt = x(a)$
Shifting	$x(t)\,\delta(t-a) = x(a)\,\delta(t-a)$
Scaling	$\delta(at \pm b) \equiv \frac{1}{\lvert a\rvert}\,\delta\left(t \pm \frac{b}{a}\right)$
Convolution	$x(t) * \delta(t-a) = \int_{-\infty}^{+\infty} x(\tau)\,\delta(t-\tau-a)\,d\tau = x(t-a)$

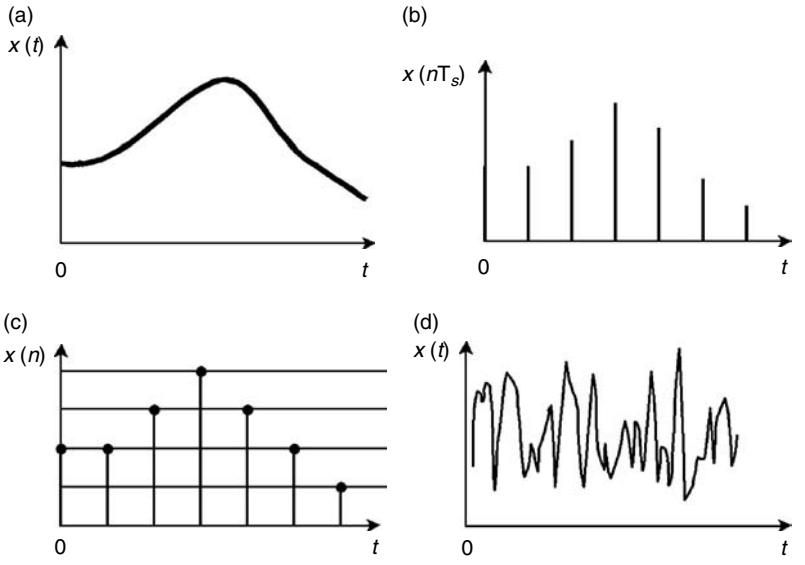


FIGURE 3.3 Description of singularity function: (a) impulse, (b) step, (c) ramp, (d) parabolic.

The signum function is derived from the step function according to

$$\text{sgn}(t) = 2u(t) - 1$$

3.1.2.3 The Ramp Function

Integrating (3.4) yields the ramp function shown in Figure 3.3c. The unit ramp function $r(t)$ is expressed as

$$r(t) = \begin{cases} t, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{3.5}$$

Furthermore, integrating $r(t)$ yields a unit parabolic signal of the form

$$a(t) = \begin{cases} \dfrac{t^2}{2}, & t \geq 0 \\ 0, & t < 0 \end{cases} \tag{3.6}$$

This is depicted in Figure 3.3d.

3.1.3 Basic Continuous-Time Signals

Figure 3.4 shows some of the elementary signals that are often encountered in signal analysis. Some of these signals can be derived directly from the singular functions discussed above. For example, the unit rectangular pulse signal that extends from $-\tau/2$ to $\tau/2$ can be expressed as

$$\prod(t) = u\left(t + \frac{\tau}{2}\right) - u\left(t - \frac{\tau}{2}\right) \tag{3.7}$$

and this is depicted in Figure 3.4a.

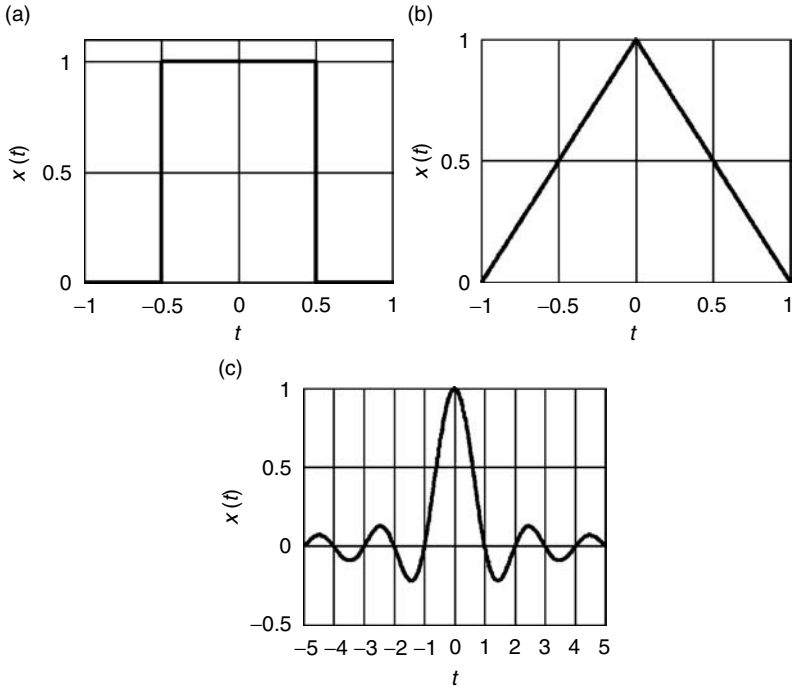


FIGURE 3.4 Description of basic CT signal: (a) rectangular pulse, (b) triangular pulse, and (c) sin *c* function.

The triangular function, denoted as $\Lambda(t)$, is defined as

$$\Lambda(t) = \begin{cases} 1 - |t|, & |t| < 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

Using the convolution theorem, to be discussed in Section 3.1.5, it can be shown that

$$\Lambda(t) = \prod(t) * \prod(t)$$

The sin*c* signal is defined as

$$\text{sin}c(t) = \begin{cases} \dfrac{\sin(\pi t)}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases} \tag{3.9}$$

Both the triangular and sin*c* signals are shown in Figures 3.4b and 3.4c, respectively.

3.1.4 Basic Discrete-Time Signals

The unit sample sequence, denoted as $\delta(n)$, is defined as

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & \text{otherwise} \end{cases}$$

It is also referred to as the unit impulse sequence or Kronecker delta function. The working properties of the unit sample sequence are analogous to that of $\delta(t)$ and these are shown here:

$$\sum_{n=-\infty}^{\infty} x(n)\,\delta(n-m) = x(m)$$

$$x(n)\,\delta(n-m) = x(m)\,\delta(n-m)$$

$$\delta(an \pm b) = \delta\left(n \pm \frac{b}{a}\right)$$

$$x(n) * \delta(n-m) = \sum_{n=-\infty}^{\infty} x(r)\,\delta(n-r-m) = x(n-m)$$

Note that the scaling property is only applicable when both a and b/a are integers. Two other basic signals that are useful for analysis are the unit step and unit ramp signals. The unit step sequence, $u(n)$, is defined as

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} \tag{3.10}$$

whereas the unit ramp signal, denoted as $r(n)$, is given by

$$r(n) = \begin{cases} n, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

The above three sequences are related as follows:

$$\delta(n-k) = u(n-k) - u(n-k-1)$$

$$u(n) = \sum_{m=-\infty}^{n} \delta(n-m)$$

$$r(n) = u(n) * u(n-1)$$

Figure 3.5 illustrates the above DT sequences.

3.1.5 Analysis of Continuous-Time Signals

3.1.5.1 Basic Operations on Signals

There are some important operations that are often performed on signals so as to understand either their characteristics or the physical phenomena generating them. The three most common operations are shifting, time scaling, and reflection. Examples of these operations are illustrated in Figure 3.6, where $x(t)$ is expressed as

$$x(t) = \begin{cases} t+1, & -1 \leq t \leq 3 \\ 3, & 3 < t \leq 6 \\ 0, & \text{otherwise} \end{cases}$$

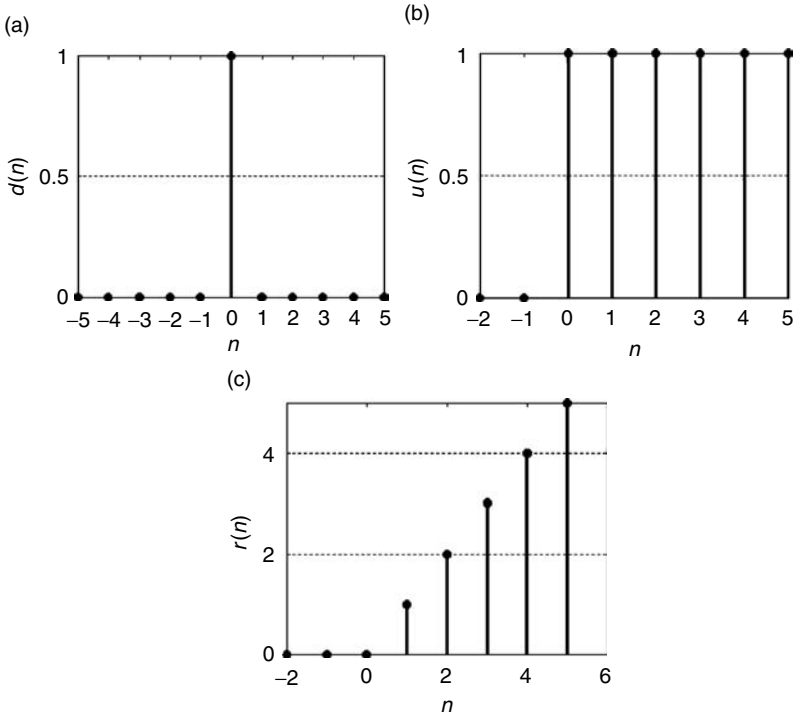


FIGURE 3.5 Description of basic DT signal: (a) unit pulse sequence, (b) unit step sequence, (c) ramp sequence.

3.1.5.2 The Convolution and Correlation Integrals[2]

Though the convolution operation is often associated with systems studies, occasionally this operation may be needed in analyzing signals obtained from a physical system. The convolution of two CT signals $x(t)$ and $y(t)$ yields $z(t)$, where

$$z(t) = x(t) * y(t) = \int_{-\infty}^{\infty} x(\tau)y(t-\tau)\,d\tau \tag{3.11}$$

Convolution is not limited to time-domain since it is also used to determine the frequency-domain spectrum associated with the product of two time-domain signals. The cross-correlation function between $x(t)$ and $y(t)$, denoted as $R_{xy}(t)$, is defined as

$$R_{xy}(t) = x(t) \oplus y(t) = \int_{-\infty}^{\infty} x(\tau)y^*(\tau-t)\,d\tau \tag{3.12}$$

Unlike the convolution there is no reflection operation here. Furthermore, the variable lag, t, is the scanning parameter that measures the degree of similarity between these two signals. If $x(t) = y(t)$, then (3.12) describes the autocorrelation function. Some properties of the correlation functions are given in Table 3.2.

Both convolution and correlation integrals are applicable to the energy as well as power signals. In the case of power signals, the integral is taken over the period T and the result is scaled by $1/T$. Correlation analysis is important as it leads to the computation of the energy spectral density for the transient signals, and power spectral density for both periodic and random signals.

TABLE 3.2 Properties of the Correlation Function

Property	Autocorrelation	Crosscorrelation
Even/Reorder	$R_{xx}(t) = R_{xx}(-t)$	$R_{xy}(t) = R_{yx}(-t)$
Upper Bound	$R_{xx}(0) \geq R_{xx}(t)$, for any t	$\lvert R_{xy}(t)\rvert \leq \sqrt{R_{xx}(0)R_{yy}(0)}$

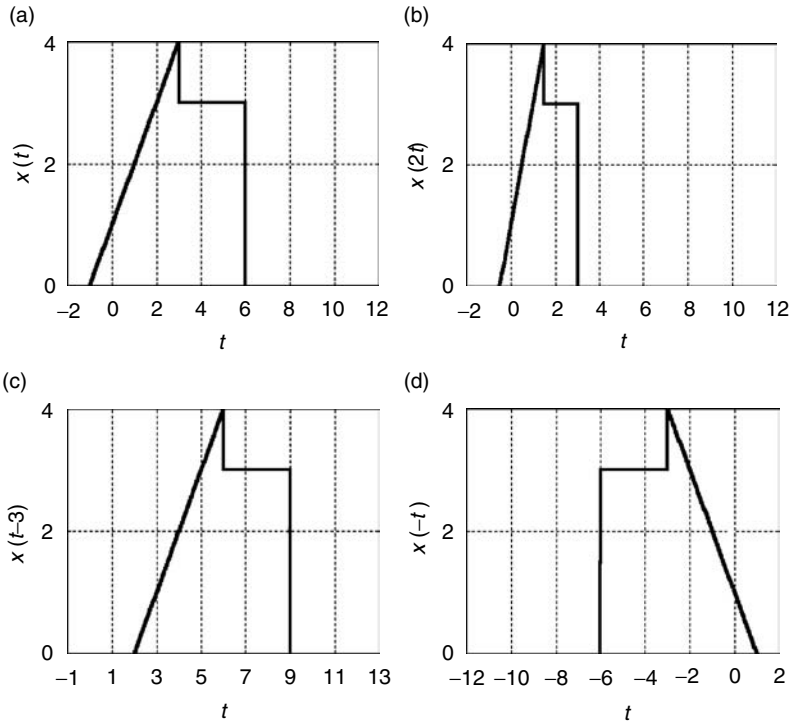


FIGURE 3.6 Basic operations on signal: (a) original signal, (b) scaling, (c) shifting, and (d) reflection.

3.1.6 Fourier Analysis of CT Signals

So far we have discussed only the time-domain methods of analyzing CT signals. The convolution integral is of particular interest since this can be used to study how a signal is modified as it passes through a system. There is need to consider the frequency-domain methods of analysis since the convolution analysis can be laborious. Furthermore, the formulation of convolution integral is based on representing the signals by shifted δ-functions. In many applications, it is more appropriate and desirable to choose a set of orthogonal functions as the basic signals since this approach leads to a reduction in computational complexity as well as providing a graphical representation of the frequency components in a given signal.

3.1.6.1 Orthogonal Basis Functions[2,3]

It is mathematically convenient to represent arbitrary signals as a weighted sum of orthogonal waveforms as this leads to a very much simplified signal analysis as well as showing the fundamental similarity between signals and vectors. Consider a set of basis function $\phi(t)$, $i = 0, \pm1, \pm2,\ldots$. This is said to be orthogonal over an interval (t_1, t_2) if

$$\int_{t_1}^{t_2} \phi_m(t)\,\phi_k^*(t)\,dt = E_k\delta(m-k) \tag{3.13}$$

where $\phi_k^*(t)$ stands for the complex conjugate of the signal. If E_k is equal to unity for all values of k, then the $\phi(t)$ is an orthonormal set. It is relatively easy to approximate a given signal by an appropriate set of orthonormal functions as this leads to minimum error between the actual signal and its approximation. Thus, a given signal $x(t)$ with finite energy over the interval $t_1 < t < t_2$ can be expressed as

$$x(t) = \sum_{k=-\infty}^{\infty} c_k \phi_k(t) \tag{3.14}$$

where

$$c_k = \int_{t_1}^{t_2} x(t)\phi_k^*(t)\,dt, \quad k = 0, \pm 1, \pm 2, \ldots$$

This equation is referred to as the generalized Fourier series of $x(t)$, and the constants c_k, $k = 0, \pm 1, \pm 2, \ldots$, are called the Fourier series coefficients with respect to the orthogonal set $\{\phi(t)\}$.

Denoting the first M terms in Equation 3.14 as $\hat{x}(t)$, the resulting error function is

$$e_M(t) = x(t) - \sum_{k=0}^{M} c_k \phi_k^*(t) \tag{3.15}$$

By computing the average power of this error function and setting its derivatives with respect to c_k to zero, yields an optimal set of $\{c_k\}$ that minimizes the error energy. Also, if $\lim_{M\to\infty} \hat{x}(t) = x(t)$, then the basis functions are said to be complete, that is, the error energy is equal to zero. When dealing with periodic signals, the time interval (t_1, t_2) is equal to the period, T, of the signal. In addition, $\phi_n(t) = \exp\{jn\omega_0\}$ is often selected as the set of basis functions, for $n = 0, \pm 1, \pm 2, \ldots$, and $\omega_0 = 2\pi/T$. The methods of computing the Fourier series coefficients are subsequently discussed.

3.1.6.2 The Complex Exponential Fourier Series

Let the signal $x(t)$ be such that $x(t) = x(t + T)$ and that it satisfies the Dirichlet conditions:[3]

1. $x(t)$ is absolutely integrable over its period, that is,

$$\int_{t_1}^{t_1+T} |x(t)|\,dt < \infty$$

2. the number of maxima and minima of $x(t)$ in each period is finite,
3. the number of discontinuities of $x(t)$ in each period is finite,

then $x(t)$ can be expanded as

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_0 t}, \quad \omega_0 = 2\pi f_0 \tag{3.16}$$

where

$$c_k = \frac{1}{T}\int_{t_1}^{t_1+T} x(t) e^{-jk\omega_0 t}\,dt \tag{3.17}$$

for any arbitrary value of t_1.

The coefficients c_k are called the complex Fourier series coefficients for the signal $x(t)$, which, in general, may be complex numbers.

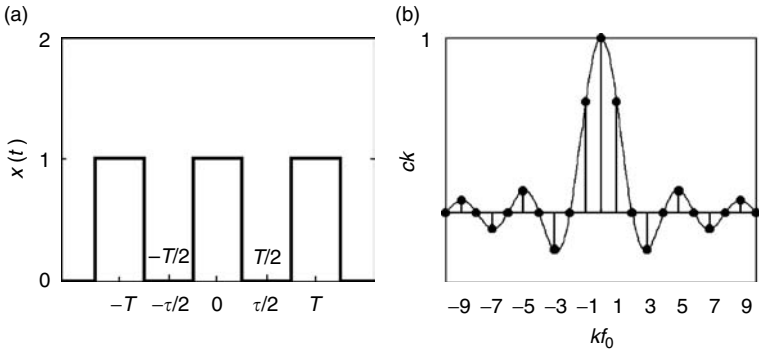


FIGURE 3.7 Rectangular periodic signal and its Fourier series coefficients.

The Dirichlet conditions are only sufficient conditions for the existence of the Fourier series expansion. The Fourier series expansion of signals, which does not satisfy these conditions, can still be obtained.

The complex Fourier series coefficients can be determined by either directly evaluating the integral in (3.17) or using the method of differentiation. The latter method relies on differentiating $x(t)$, for a certain number of times, to produce a train of impulses. These two methods of determining c_k will now be illustrated.

Consider the periodic signal shown in Figure 3.7a, which can be expressed as

$$x(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} \prod\left(\frac{t-nT}{\tau}\right)$$

Substituting this in Equation 3.17 we have

$$\begin{aligned} c_k &= \frac{1}{T}\int_{-T/2}^{T/2} x(t)e^{-j2\pi kf_0 t}dt \\ &= \frac{1}{T}\int_{-\tau/2}^{\tau/2} e^{-j2\pi kf_0 t}dt \\ &= \frac{1}{\pi k}\sin(k\pi f_0\tau) \\ &= \tau f_0 \operatorname{sinc}(kf_0\tau) \end{aligned}$$

Thus,

$$x(t) = \sum_{n=-\infty}^{\infty} \tau f_0 \operatorname{sinc}(kf_0\tau)e^{j2\pi kf_0 t}$$

A graph of the magnitude and phase spectra of the complex Fourier series coefficients are shown in Figure 3.7b for varied τ/T.

Consider the computation of the complex Fourier series coefficients of the signal shown in Figure 3.8a using the method of differentiation. This signal is expanded according to Equation 3.16. Differentiating this equation twice with respect to t yields

$$x''(t) = \sum_{k=-\infty}^{\infty} (j2\pi kf_0)^2 c_k e^{j2\pi kf_0 t}$$

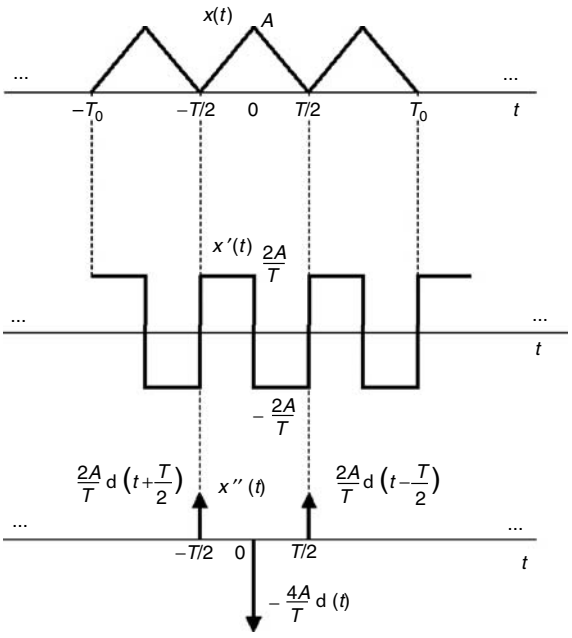


FIGURE 3.8 Illustration of the differentiation technique for Fourier series coefficient computation.

which can be written as

$$x''(t) = \sum_{k=-\infty}^{\infty} \beta_k e^{j2\pi k f_0 t}$$

Figure 3.8c shows the result of differentiating $x(t)$. It is noted that if a signal is periodic, its derivatives will also be periodic. This implies that β_k is the complex Fourier series coefficient of $x''(t)$ and can be computed from

$$\beta_k = \frac{1}{T}\int_{-T/2}^{T/2} x''(t) e^{-j2\pi k f_0 t} dt$$

where

$$x''(t) = 2\frac{A}{T}\left[\delta\left(t+\frac{T}{2}\right) - 2\delta(t) + \delta\left(t-\frac{T}{2}\right)\right]$$

Thus,

$$\beta_k = -8\frac{A}{T^2}\sin^2\left(\frac{\pi k T f_0}{2}\right) = (j2\pi f_0 k)^2 c_k$$

That is,

$$c_k = \begin{cases} \dfrac{2A}{\pi^2 k^2}, & k \text{ odd} \\ 0, & \text{otherwise} \end{cases}$$

and

$$c_0 = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\,dt = \frac{A}{2}.$$

This method of determining c_k, where applicable, is less laborious than the direct method.

3.1.6.3 Fourier Series for Real Signals[3]

A real signal $x(t)$ that satisfies the Dirichlet conditions can be expanded by the following procedure. Recall Equation 3.17 and replace k by $-k$, we obtain

$$c_{-k} = \frac{1}{T}\int_{t_1}^{t_1+T} x(t)e^{j2\pi kf_0 t}\,dt = \left[\frac{1}{T}\int_{t_1}^{t_1+T} x(t)e^{-j2\pi kf_0 t}\,dt\right]^* = c_k^* \tag{3.18}$$

That is, the positive and negative coefficients are complex conjugates of each other for real signals. For such signals, $|c_k|$ has even symmetry and $\angle c_k$ has odd symmetry with respect to $k = 0$. Denote $c_k = (a_k - jb_k)/2$, then $c_{-k} = (a_k + jb_k)/2$, so that

$$c_k e^{j2\pi kf_0 t} + c_{-k}e^{-j2\pi kf_0 t} = \frac{a_k - jb_k}{2}e^{j2\pi kf_0 t} + \frac{a_k + jb_k}{2}e^{-j2\pi kf_0 t}$$

Since c_0 is real and defined as $c_0 = a_0/2$, then we can write

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}[a_k\cos(2\pi kf_0 t) + b_k\sin(2\pi kf_0 t)] \tag{3.19}$$

This relationship, which only holds for a real periodic signal $x(t)$, is called the trigonometric Fourier series expansion. Since

$$\begin{aligned} c_k = \frac{a_k - jb_k}{2} &= \frac{1}{T}\int_{t_1}^{t_1+T} x(t)e^{-j2\pi kf_0 t}\,dt \\ &= \frac{1}{T}\int_{t_1}^{t_1+T} x(t)[\cos(2\pi kf_0 t) - j\sin(2\pi kf_0 t)]\,dt \end{aligned} \tag{3.20}$$

Hence

$$a_k = \frac{2}{T}\int_{t_1}^{t_1+T} x(t)\cos(2\pi kf_0 t)\,dt \tag{3.21a}$$

$$b_k = \frac{2}{T}\int_{t_1}^{t_1+T} x(t)\sin(2\pi kf_0 t)\,dt \tag{3.21b}$$

and

$$a_0 = \frac{2}{T}\int_{t_1}^{t_1+T} x(t)\,dt$$

The third method of Fourier series expansion of real signals is given by

$$x(t) = c_0 + \sum_{k=1} c_k\cos(2\pi kf_0 t + \theta_k) \tag{3.22}$$

TABLE 3.3 Fourier Series Symmetry Conditions

Type of Symmetry	Real Fourier Series Coefficient	Complex Fourier Series Coefficients	Comments
Even periodic $x(t) = x(-t)$	$a_k = \frac{4}{T}\int_0^{T/2} x(t)\cos(2\pi k f_0 t)\,dt$ $b_k = 0$	$c_k = \frac{1}{2}a_k$ c_k has real value	Phase of c_k is either zero or π
Odd periodic $x(t) = -x(-t)$	$a_k = 0$ $b_k = \frac{4}{T}\int_0^{T/2} x(t)\sin(2\pi k f_0 t)\,dt$	$c_k = -j\frac{1}{2}b_k$ c_k has imaginary values	Phase of c_k is either $\pi/2$ or $-\pi/2$
Half-wave even symmetry $x(t) = x\left(t+\frac{T}{2}\right)$	a_{2k} and b_{2k} may have nonzero values but $a_{2k+1} = 0, \quad b_{2k+1} = 0$	$c_{2k} \neq 0$ $c_{2k+1} = 0$	
Half-wave odd symmetry $x(t) = -x\left(t+\frac{T}{2}\right)$	a_{2k+1} and b_{2k+1} may have nonzero values but $a_{2k} = 0, \quad b_{2k} = 0$	$c_{2k} = 0$ $c_{2k+1} \neq 0$	

TABLE 3.4 Properties of the Fourier Series

Property	Signal Description	Fourier Series Coefficients, c_k
Linearity	$ax(t) + by(t)$; a, b constants	$a\alpha_k + b\beta_k$
Multiplication	$x(t)y(t)$	$\alpha_k * \beta_k$
Convolution	$x(t) * y(t)$	$\alpha_k\beta_k$
Parseval's theorem	$\frac{1}{T}\int_{t_1}^{t_1+T} \lvert x(t)^2\rvert\,dt$	$\sum_{k=-\infty}^{\infty} \lvert\alpha_k\rvert^2$
Time shift	$x(t \pm \tau)$	$\alpha_k e^{\pm j2\pi f_0 \tau}$
Differentiation	$\frac{d^n}{dt^n}x(t)$	$(j2\pi k f_0)^n \alpha_k$
Integration	$\int_T x(\tau)\,d\tau$	$(j2\pi k f_0)^{-1}\alpha_k, \quad \alpha_0 = 0$

where c_0 is the dc component, c_k and θ_k represent the amplitude and phase angle of the kth harmonic, respectively. Equation 3.22 is called the harmonic form of Fourier series expansion of $x(t)$. The parameters c_k and θ_k are related to a_k and b_k according to

$$c_0 = \frac{a_0}{2}, \qquad c_k = \sqrt{a_k^2 + b_k^2}, \qquad \theta_k = \tan^{-1}\frac{b_k}{a_k}$$

3.1.6.4 Properties of the Fourier Series[1,4]

Knowledge of signal symmetry can simplify its complex Fourier series coefficients computation. While many forms of symmetry can be established, the following important types of symmetry are more often encountered in signal analysis:

- Even symmetry, $x(t) = x(-t)$
- Odd symmetry, $x(t) = -x(t)$
- Half-wave odd symmetry, $x(t) = -x(t + T/2)$

The effects of symmetry on the Fourier series computations are shown in Table 3.3. The other properties on Fourier series are summarized in Table 3.4, where α_k and β_k are the complex Fourier series coefficients of $x(t)$ and $y(t)$, respectively.

3.1.7 Fourier Transform

Frequency domain method of analyzing periodic CT signals has been presented in the previous section. A different technique, termed the Fourier transform, is used for the analysis of aperiodic signals. The development of the Fourier transform is based on Equations 3.16 and 3.17. Substituting Equations 3.17 into 3.16 gives

$$x(t) = \sum_{k=-\infty}^{\infty} \left(\frac{1}{T} \int_{-T/2}^{T/2} x(\lambda) e^{-j2\pi k f_0 \lambda} d\lambda \right) e^{j2\pi k f_0 t} \tag{3.23}$$

Allowing $T \to \infty$, then $1/T \to df$, $kf_0 \to f$, and Equation 3.23 becomes

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi k f t} dt \tag{3.24a}$$

and

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df \tag{3.24b}$$

3.1.8 Properties of the Fourier Transform[5,6]

Some of the basic properties of the Fourier transform that are often used in analysis are given in Table 3.5.

The Fourier transform pairs of the following basic signals are also useful for analysis:

$$K \leftrightarrow K\delta(f)$$

$$\operatorname{sgn}(t) \leftrightarrow \frac{1}{j\pi f}$$

$$u(t) \leftrightarrow \frac{1}{2}\delta(f) + \frac{1}{j2\pi f}$$

$$\cos(2\pi f_0 t) \leftrightarrow \frac{1}{2}\delta(f - f_0) + \frac{1}{2}\delta(f + f_0)$$

A two-step procedure is required in order to determine the Fourier transform of a periodic signal. If $x(t)$ is a periodic signal with period T, then $x(t)$ can be Fourier series expanded as

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k f_0 t}, \quad f_0 = \frac{1}{T} \tag{3.25}$$

Applying the linearity property to this equation gives

$$X(f) = \sum_{k=-\infty}^{\infty} c_k \delta(f - k f_0) \tag{3.26}$$

as the Fourier transform of any periodic signal $x(t)$. Equation 3.26 explains the difference between the frequency spectrum produced by the Fourier series analysis and the amplitude spectral density produced by the Fourier transformation. Thus, the frequency spectrum is a (discrete) display of c_k versus kf_0, whereas

TABLE 3.5 Properties of the Fourier Transform

Property	Signal Description	Fourier Transform
Linearity	$ax(t) + by(t)$; a, b constants	$aX(f) + bY(f)$
Evenness and oddness	$x(-t) = x(t)$	$X(f) = 2\int_0^\infty x(t)\cos(2\pi ft)\,dt$
	$x(-t) = -x(t)$	$X(f) = -2\int_0^\infty x(t)\sin(2\pi ft)\,dt$
Time shift	$x(t-\tau)$	$e^{-j2\pi f\tau}X(f)$
Time scale	$x(at)$	$\frac{1}{\lvert a\rvert}X\left(\frac{f}{a}\right)$
Time reversal	$x(-t)$	$X^*(f)$
Duality	$X(t)$	$x(-f)$
Time convolution	$x(t) * y(t)$	$X(f)\,Y(f)$
Frequency convolution	$x(t)\,y(t)$	$X(f) * Y(f)$
Modulation	$x(t)e^{j2\pi f_0 t}$	$X(f-f_0)$
Time differentiation	$\frac{d^n}{dt^n}x(t)$	$(j2\pi f)^n X(f)$
Frequency differentiation	$t^n x(t)$	$\left(\frac{j}{2\pi}\right)^n \frac{d^n}{df^n}X(f)$
Integration	$\int_{-\infty}^{\infty} x(\tau)\,d\tau$	$\frac{1}{j2\pi f}X(f) + \frac{1}{2}X(0)\,\delta(f)$
Correlation	$R_{xy}(\tau) = \int_{-\infty}^{\infty} y(t)x(t-\tau)\,dt$	$Y(-f)\,X(f)$
Parseval's theorem	$\int_{-\infty}^{\infty} \lvert x(t)\rvert^2\,dt$	$\int_{-\infty}^{\infty} \lvert X(f)\rvert^2\,df$

the amplitude spectral density gives a continuous display of the amplitude density spectrum, which in this case is in the form of impulses, rather than just a number.

Similarly the Fourier transform of a train of impulses of the form

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{3.27}$$

is given by

$$P(f) = \frac{1}{T}\sum_{k=-\infty}^{\infty} \delta(f - kF_s), \quad F_s = \frac{1}{T} \tag{3.28}$$

3.1.8.1 Energy and Power Spectral Density[6]

Suppose $x(t)$ is an aperiodic signal with a Fourier transform $X(f)$, then its energy is given by

$$E = R_{xx}(0) = \int_{-\infty}^{\infty} \lvert x(t)\rvert^2\,dt = \int_{-\infty}^{\infty} \lvert X(f)\rvert^2\,df \tag{3.29}$$

This is Parseval's theorem and it shows that the principle of conservation of energy in the time and frequency domains holds. The amplitude spectrum $X(f)$ can be expressed as

$$X(f) = \lvert X(f)\rvert \angle X(f)$$

and denoting

$$S_{xx}(f) = |X(f)|^2$$

then the total energy of the signal is given by

$$E = \int_{-\infty}^{\infty} S_{xx}(f)\,df \tag{3.30}$$

where $S_{xx}(f)$ represents the distribution of the signal energy as a function of frequency. $S_{xx}(f)$ is termed the energy spectral density for the finite energy signal $x(t)$.

Consider a periodic signal $x(t)$ with an autocorrelation function

$$R_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} x(t)x^*(t-\tau)\,dt$$

Then,

$$R_{xx}(0) = \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} |x(t)|^2\,dt \tag{3.31}$$

which equals the signal power. Following the same procedure for energy signals, we define $S_{xx}(f)$ as the Fourier transform of $R_{xx}(\tau)$ so that

$$P = R_{xx}(0) = \int_{-\infty}^{\infty} S_{xx}(f)\,df \tag{3.32}$$

and $S_{xx}(f)$ is termed the power spectral density of the periodic signal $x(t)$.

The need to analyze stationary random signals also arises in many practical situations. The properties of such signals can be inferred from their correlation functions. For example the autocorrelation function, $\phi_{xx}(\tau)$ of a stationary random signal decreases and goes to zero as τ increases since the events become uncorrelated for a large separation of time. Hence, $\phi_{xx}(\tau) = \phi_{xx}(-\tau)$ and its Fourier transform exists. Consequently we can write

$$\phi_{xx}(0) = \int_{-\infty}^{\infty} \Gamma_{xx}(f)\,df \tag{3.33}$$

where $\Gamma_{xx}(f)$ and $\phi_{xx}(0)$ represent, respectively, the power spectral density and the average power of a random process.

3.1.9 Sampled Continuous-Time Signals

Discrete-time (DT) signals arise either naturally or by sampling continuous-time (CT) signals; however, the latter form is more often encountered in practice. In this case, a digital signal is formed from a CT signal through the process of analog-to-digital conversion. The first part of this process is the sampling of the analog signal, that is, the conversion of $x(t)$ into $x(nT_s)$, where T_s is the sampling period and its reciprocal, $F_s = 1/T_s$, is the sampling frequency in samples per second. The sampling frequency must be appropriately selected to avoid spectral distortion (aliasing), thereby ensuring that $x(t)$ can be reconstructed from its samples. To gain a good understanding of this procedure the sampling process is examined in the frequency domain.

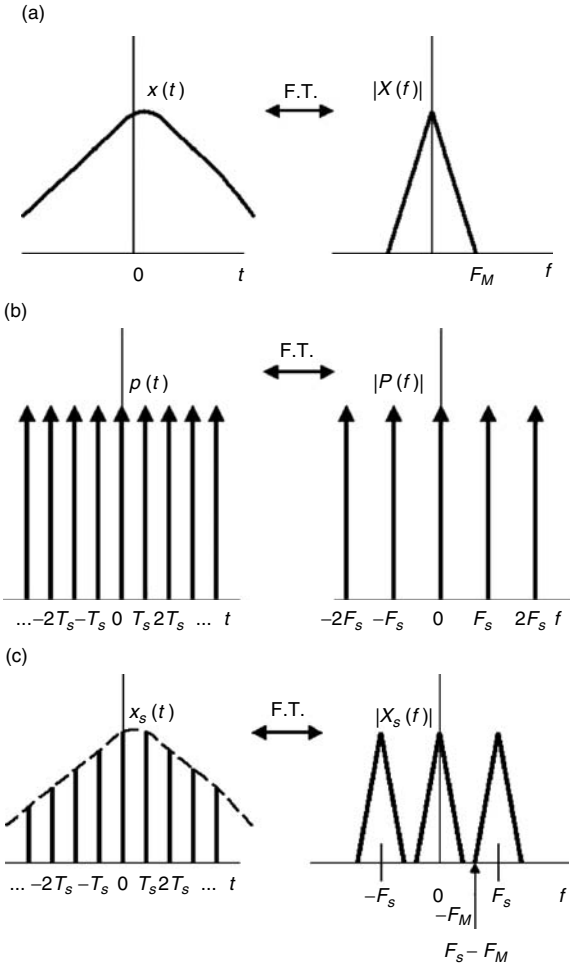


FIGURE 3.9 Ideal impulse sampling process: (a) bandlimited continuous-time signal and its Fourier transform, (b) train of impulses and its spectra, (c) sampled signal and its spectra.

3.1.9.1 Impulse Sampling[6–9]

Consider an idealized impulse-sampling process shown in Figure 3.9, where $x(t)$ is to be sampled using the train of impulses $p(t)$, so that

$$x_s(t) = x(t)p(t) = x(t)\sum_{n=-\infty}^{\infty}\delta(t-nT_s) = \sum_{n=-\infty}^{\infty}x(nT_s)\delta(t-nT_s) \tag{3.34}$$

The main difference between $x_s(t)$ and $x(nT_s)$ is that the former is essentially a CT signal with zero values except at the integer values of T_s, while the latter is a perfectly selected sample of $x(t)$ as a result of impulse sampling.

The Fourier transform of Equation 3.34 yields

$$X_s(f) = X(f) * P(f) = F_s\sum_{k=-\infty}^{\infty}X(f-F_s) \tag{3.35}$$

where $X(f)$ is the spectrum of $x(t)$.

It is observed that $X_s(f)$ consists of a scaled and periodically repeated version of $X(f)$ with period F_s. As shown in Figure 3.9c, it is obvious that when $F_s - F_M \geq F_M$, there is no overlapping of the spectra and the signal $x(t)$ can be recovered completely from $x_s(t)$. However, if $F_s - F_M < F_M$ the replicas of $X(f)$ will overlap, resulting in a distorted spectrum and as such, $x(t)$ can no longer be recovered from its sampled version. Consequently, in order to recover $x(t)$ from its samples, the sampling frequency should be such that

$$F_s - F_M \geq F_M$$

that is,

$$F_s \geq 2F_M$$

This is called the Nyquist sampling theorem. The minimum sampling frequency $F_s = 2F_M$ is called the Nyquist frequency. Sampling a signal at less than the Nyquist frequency results in a spectral distortion termed aliasing. Furthermore, sampling a signal at a frequency of at least Nyquist frequency implies that an ideal low-pass filter (LPF) with a gain of $1/F_s$ and cutoff frequency F_c can be used to recover its original spectrum, where $F_M \leq F_c \leq F_s - F_M$.

Suppose we want to reconstruct $x(t)$ from its samples. Assume that $X(f)$ is the spectrum of $x(nT_s)$, with no aliasing, as shown in Figure 3.9c. Thus,

$$X_a(f) = \begin{cases} \frac{1}{F_s}X(f), & |f| \leq \frac{F_s}{2} \\ 0, & |f| > \frac{F_s}{2} \end{cases} \tag{3.36}$$

Note that

$$X_a(f) = \sum_{n=-\infty}^{\infty} x(nT_s)e^{-j2\pi nfT_s} \tag{3.37}$$

so that its inverse Fourier transform is given by

$$\begin{aligned} x_a(t) &= \frac{1}{F_s}\sum_{n=-\infty}^{\infty} x(nT_s)\int_{-F_s/2}^{F_s/2} e^{j2\pi f(t-nT_s)}df \\ &= \sum_{n=-\infty}^{\infty} x(nT_s)\frac{\sin[\pi(t-nT_s)/T_s]}{\pi(t-nT_s)/T_s} \end{aligned} \tag{3.38}$$

This is the formula for the reconstruction of $x(t)$ from its samples. That is, $x(t)$ is generated by multiplying the appropriately shifted function $g(t) = \sin c(tF_s)$ by the corresponding samples of $x(nT_s)$.

3.1.9.2 Practical Sampling[8–10]

The above discussion on sampling is based on the idealized models of periodic impulse sampling and bandlimited interpolation. In practice, CT signals are not precisely bandlimited just as impulse signals and ideal low-pass filters do not exist physically. Figure 3.10 represents the block diagram for the conversion of continuous signals into their discrete forms. The continuous-time signal is prefiltered, sampled, quantized, and finally encoded into finite-length words of, say, b bits. The prefilter, which is also called anti-aliasing filter (AAF), is a low-pass filter that is needed to limit the input signal bandwidth to $F_s/2$ prior to sampling to avoid aliasing. In practice this filter will possess non-ideal characteristics, hence

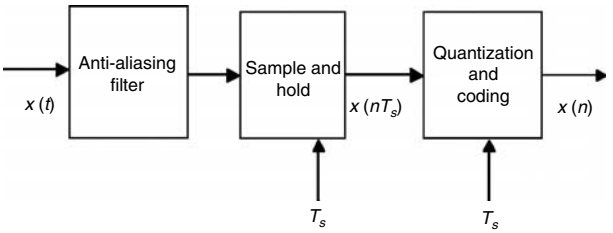


FIGURE 3.10 Practical sampling of continuous-time signals.

it should be designed to provide sufficient attenuation, usually to a level undetectable by the analog-to-digital converter (ADC) at frequencies above the Nyquist frequency.

The prefiltered signal is fed into the ADC where it will be converted to a DT signal. The ADC has an in-built sample-and-hold circuit and it operates at the sampling rate of F_s; however, the sampling function has a finite width as opposed to the impulse sampling discussed above. The sampling operation can be modeled by the finite-width pulse sampler shown in Figure 3.11b, in which the sampling gate is open for τ-out of T_s seconds and shorted to ground the remainder of the sampling interval. Here $p(t)$ is expressed as

$$p(t) = \sum_{n=-\infty}^{\infty} \prod \left(\frac{t - nT_s}{\tau} \right)$$

which can be Fourier series expanded as

$$p(t) = \sum_{n=-\infty}^{\infty} c_k e^{j2\pi k f_s t}$$

where

$$c_k = \frac{\tau}{T_s} \sin c \left(k \frac{\tau}{T_s} \right)$$

The Fourier transform of the sampled signal can be written as

$$X_s(f) = \sum_{n=-\infty}^{\infty} c_k X(f - kF_s)$$

Note that c_k is not constant in this expression (as opposed to the impulse sampling) since its value depends on the harmonic number (k) as well as the duty cycle τ/T_s. The discrete-time signal, $x(nT_s)$, is fed into the quantizer where each sample is transformed into one of the nearest finite sets of prescribed values, that is, $\hat{x}(n) = Q(x(nT_s))$, where $\hat{x}(n)$ is the quantized sample. The quantization process is shown in Figure 3.11d for zero-order sample hold ADC, where L_i denotes the quantization level and Δ is the quantization step. The quantization error (or noise) incurred in this process is

$$L_i - \frac{\Delta}{2} < x(nT_s) < L_i + \frac{\Delta}{2}$$

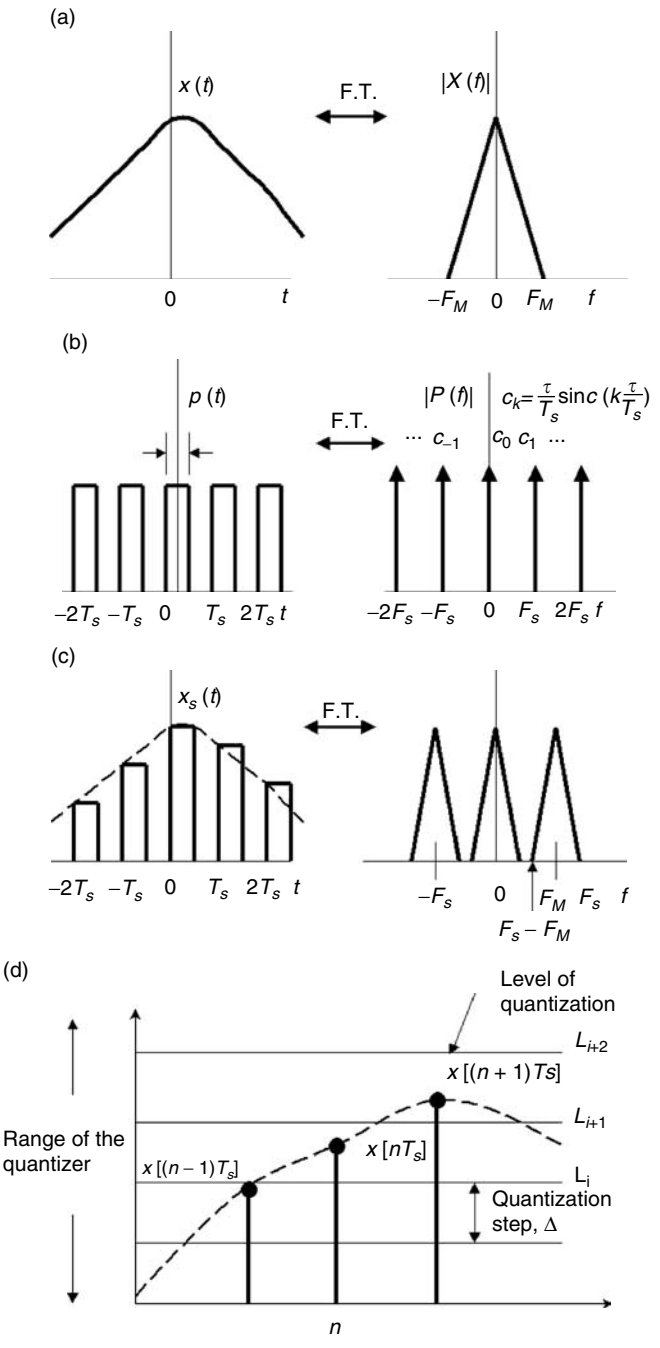


FIGURE 3.11 Finite-width pulse sampling signals and spectra: (a) bandlimited signal and spectrum, (b) finite-width train of pulses and its transform, (c) sampled signal and its spectra, (d) quantization process.

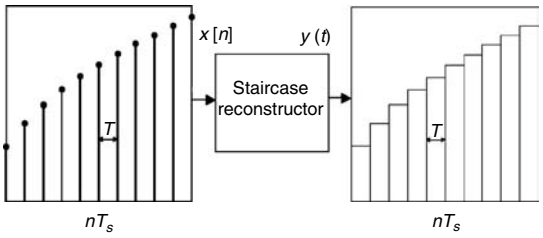


FIGURE 3.12 Digital-to-analog conversion process.

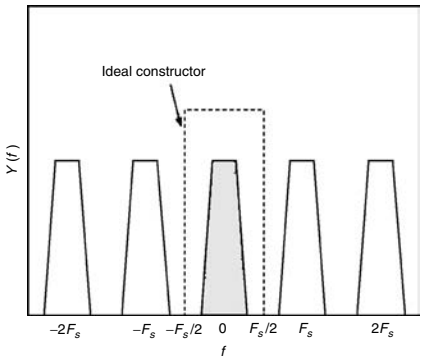


FIGURE 3.13 Frequency response of the ideal reconstruction filter.

From statistical considerations, the noise power is found to be $\Delta^2/12$ watts. A common measure of the performance of the ADC is the ratio of the signal power to noise power, and this is called the signal-to-quantization noise power, which expressed in decibels (dB) is

$$\text{SQNR(dB)} = 1.76 + 6.02b$$

3.1.9.3 Digital-to-Analog Conversion[8–12]

Reconstruction of the analog signal from its sampled form is closely akin to lowpass filtering of the sampled signal. Figure 3.12 shows how an analog signal can be reconstructed by filling the gaps between samples and holding the current value constant till the next sample is received. Consider an ideal reconstruction filter with an impulse response function $h(t)$, then its response is given by

$$y(t) = \sum_{n=-\infty}^{\infty} x(nT_s)\,h(t - nT_s)$$

Taking the Fourier transform of this equation gives

$$Y(f) = H(f)X(f)$$

where $X(f)$ is the periodic spectrum of $x(nT_s)$ as shown in Figure 3.13 and $h(t) = \sin c(F_s t)$. Note that $h(t)$ is noncausal; hence it cannot be used for real-time applications. Furthermore, since $h(t)$ is not time-limited, an infinite number of impulse responses must be used for interpolating between values in order to obtain exact results. Consequently, alternative reconstruction filters such as zero-order hold (staircase), first-order, or fractional-order holds are used in practice. However, the staircase reconstruction filter is, by and large, the simplest and most widely used in practice. The impulse response of this filter is given

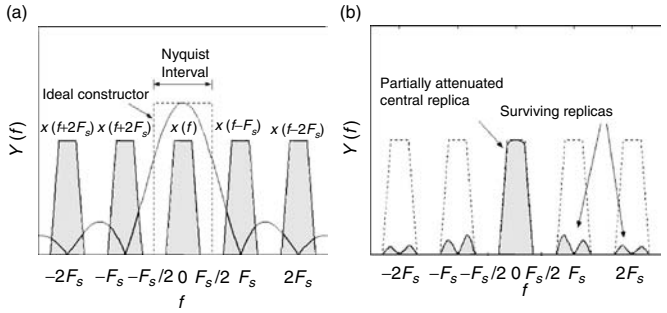


FIGURE 3.14 (a) Staircase reconstruction in frequency domain, (b) Result of staircase reconstruction in frequency domain.

as $h_o(t) = u(t) - u(t - T_s)$ since it must have a duration of T_s seconds to fill the entire gap between the samples. Thus, in effect this filter, as its name implies, generates a staircase approximation to the original analog signal. The frequency response of the filter is

$$H_o(f) = T\frac{\sin(\pi f T_s)}{\pi f T_s}e^{-j\pi f T_s}$$

Though the output of the staircase reconstruction filter is smoother than its sampled form, see Figure 3.14a, it contains spurious high-frequency components due to the sudden jumps in the staircase levels as different sampled values are considered. In addition, holding each of $x(nT_s)$ by T_s seconds introduces a time shift of $T_s/2$ to the output signal. However, this time delay has virtually no effect on the quality of the output signal. Figure 3.14b compares the signal spectra before and after the staircase reconstruction filter. It is noted that the output spectrum is slightly distorted due to non-ideal characteristics of $H_o(f)$ and that distorted and attenuated versions of $X(f)$ remain centered at nonzero multiples of F_s. These remaining spectral replicas may be removed by using an anti-imaging filter.[8,11,12] In essence, the anti-imaging filter smoothens out the discontinuities produced by the staircase reconstruction filter as illustrated in Figure 3.15.

3.1.10 Frequency Analysis of Discrete-Time Signals

The analysis of discrete-time (DT) signals in the frequency domain is very much similar to that of continuous-time (CT) signals. As in the CT analysis, the techniques for the analysis depend on the type of signal. Analysis of aperiodic DT signals will be considered first.

3.1.10.1 Discrete-Time Fourier Transform[6–8]

The decomposition of an aperiodic DT signal into its frequency components is carried out using discrete-time Fourier transformation (DTFT). Thus, the DTFT of $x(n)$ is given by

$$X(f) = \sum_{n=-\infty}^{\infty} x(n)e^{-j2\pi fn} \tag{3.39}$$

Unlike the Fourier transform of analog signal, $X(f)$ is periodic with period F_s; hence, the frequency range for a DT signal is unique over the frequency interval $(-F_s/2, F_s/2)$ or, equivalently $(0, F_s)$. Note that Equation 3.39 must be absolutely summable in order for $X(f)$ to exist, that is,

$$\sum_{n=-\infty}^{\infty} |x(n)| < \infty$$

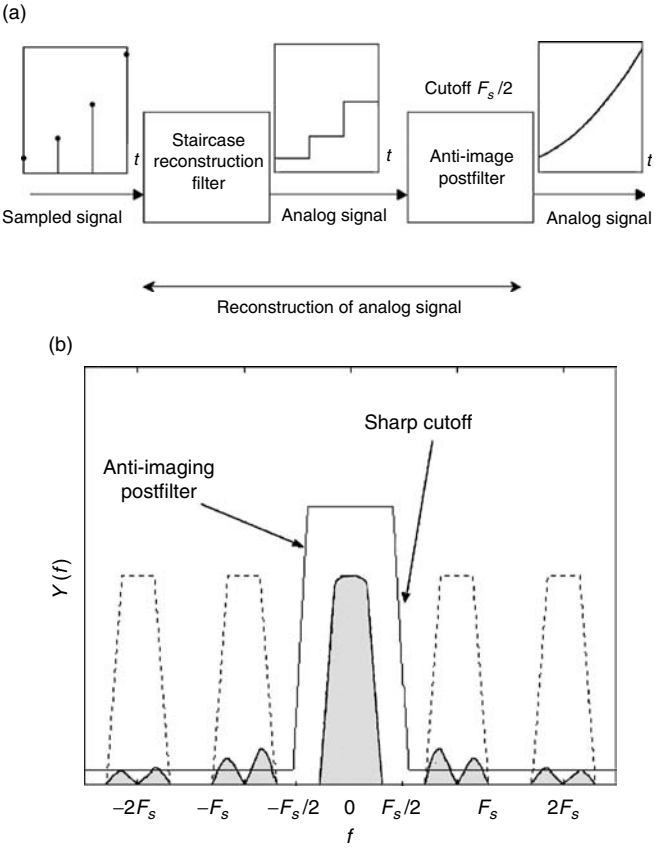


FIGURE 3.15 (a) Reconstruction of analog signal. (b) effect of anti-image postfiltering.

If the spectrum of a signal exists, then we can find the signal from its spectrum through the inverse DTFT. The inverse DTFT of $X(f)$ is given as

$$x(n) = \frac{1}{F_s}\int_{-F_s/2}^{F_s/2} X(f)e^{j2\pi fn}df$$

The properties of the DTFT of a DT signal are shown in Table 3.6.

For DT signals, the convolution of two sequences $x(n)$ and $y(n)$ is expressed as

$$x(n) * y(n) = \sum_{m=-\infty}^{\infty} x(m)y(n-m) \tag{3.40}$$

while the cross-correlation function of $x(n)$ and $y(n)$ is given by

$$R_{xy}(n) = x(n) \oplus y(n) = \sum_{m=-\infty}^{\infty} x(m)y^*(m-n)$$

When $x(n) = y(n)$, then the autocorrelation of $x(n)$ is

$$R_{xx}(n) = \sum_{m=-\infty}^{\infty} x(m)x^*(m-n)$$

TABLE 3.6 Properties of DTFT

Property	Signal Description	Frequency Domain
Even symmetry (real signal)	$x_e(n) = \frac{1}{2}\{x(n) + x(-n)\}$	$X_e(f) = \sum_{n=-\infty}^{\infty} x_e(n)\cos(2\pi nf)$
Odd symmetry (real signal)	$x_o(n) = \frac{1}{2}\{x(n) - x(-n)\}$	$X_o(f) = -\sum_{n=-\infty}^{\infty} x_o(n)\sin(2\pi nf)$
Linearity	$ax(n) + by(n)$	$aX(f) + bY(f)$
Time shifting	$x(n-m)$	$e^{-j2\pi fm}X(f)$
Time reversal	$x(-n)$	$X(-f)$
Convolution	$x(n) * y(n)$	$X(f)Y(f)$
Correlation	$R_{xy}(n) = x(n) \oplus y(n)$	$S_{xy}(f) = X(f)Y(-f)$
Wiener–Khintchine theorem	$R_{xx}(n)$	$S_{xx}(f)$
Frequency shifting	$e^{j2\pi f_0 n}x(n)$	$X(f-f_0)$
Modulation	$x(n)\cos(2\pi nf_0)$	$\frac{1}{2}\{X(f+f_0) + X(f-f_0)\}$
Multiplication	$x(n)y(n)$	$\frac{1}{F_s}\int_{F_s} X(\lambda)Y(f-\lambda)d\lambda$
Differentiation in the frequency domain	$nx(n)$	$\frac{j}{2\pi}\frac{dX(f)}{df}$
Time differencing	$x(n) - x(n-1)$	$(1 - e^{-j2\pi f})X(f)$
Summation	$\sum_{m=-\infty}^{n} x(m)$	$\frac{X(f)}{(1-e^{-j2\pi f})} + \frac{F_s X(0)}{2}\sum_{m=-\infty}^{\infty}\delta(f - mF_s)$
Conjugation	$x^*(n)$	$X^*(-f)$
Parseval's theorem	$\sum_{n=-\infty}^{\infty} x(n)y^*(n)$	$\frac{1}{F_s}\int_{F_s} X(f)Y^*(f)df$

With the exception of the reflection operation (for the convolution), the procedures for computing the convolution and correlation are the same. Hence, it is more computationally efficient to use the same algorithm for evaluating both functions. To achieve this, one of the sequences is reflected (only for the correlation analysis), followed by convolution operation, that is,

$$R_{xy}(n) = x(n) * y^*(-n)$$

and

$$R_{xx}(n) = x(n) * x^*(-n)$$

The energy of an aperiodic signal is computed from

$$E = \sum_{n=-\infty}^{\infty} \left|x(n)^2\right| = \sum_{n=-\infty}^{\infty} x(n)x^*(n)$$

Substituting the conjugated form of 3.40 into this equation gives

$$E = \frac{1}{F_s}\int_{F_s} |X(f)|^2 df = \frac{1}{F_s}\int_{F_s} S_{xx}(f)\,df \tag{3.41}$$

This expression relates the distribution of the energy of an aperiodic signal to frequency. The quantity $|X(f)|^2$ is called the energy spectral density of $x(n)$. The DTFT of some commonly encountered signals are shown in Table 3.7.

TABLE 3.7 DTFT of Common DT Signals

$x(n)$	Frequency-Domain Representation, $X(f)$
$\delta(n)$	1
$A, -\infty < n < \infty$	$AF_s \sum_{k=-\infty}^{\infty} \delta(f - mF_s), \; F_s = \frac{1}{T}$
$u(n)$	$\frac{1}{1 - e^{-j2\pi f}} + \frac{F_s}{2} \sum_{k=-\infty}^{\infty} \delta(f - kF_s)$
$\Pi\left(\frac{n}{2q+1}\right)$	$\frac{\sin[(2q+1)\pi f]}{\sin(\pi f)}$
$\Lambda\left(\frac{n}{q}\right)$	$\frac{\sin^2(\pi f q)}{q \sin^2(\pi f)}$
$\mathrm{sgn}(n)$	$\frac{2}{1 - e^{-j2\pi f}}$
$\alpha^n u(n)$	$\frac{1}{1 - \alpha e^{-j2\pi f}}, \lvert\alpha\rvert < 1$
$\alpha^{\lvert n\rvert}$	$\frac{1 - \alpha^2}{1 - 2\alpha\cos(2\pi f) + \alpha^2}, \lvert\alpha\rvert < 1$
$n\alpha^n u(n)$	$\frac{\alpha e^{-j2\pi f}}{(1 - \alpha e^{-j2\pi f})^2}, \lvert\alpha\rvert < 1$
$e^{-\alpha n} u(n)$	$\frac{1}{1 - e^{-(\alpha + j2\pi f)}}, \lvert\alpha\rvert > 0$
$e^{-\alpha \lvert n\rvert}$	$\frac{1 - e^{-2\alpha}}{1 - 2e^{-\alpha}\cos(2\pi f) + e^{-2\alpha}}, \lvert\alpha\rvert > 0$
$e^{j2\pi f_0 n}$	$F_s \sum_{k=-\infty}^{\infty} \delta(f - f_0 + kF_s)$
$\cos(2\pi f_0 n + \theta)$	$\frac{F_s}{2} \sum_{k=-\infty}^{\infty} \{e^{j\theta}\delta(f - f_0 + kF_s) + e^{-j\theta}\delta(f + f_0 + kF_s)\}$
$\frac{\sin(2\pi f_c n)}{\pi n}$	$\Pi\left(\frac{f}{2f_c}\right)$

3.1.10.2 Discrete Fourier Series[6–8]

Suppose $x(n)$ is a periodic DT signal with period N, then it is possible to obtain its discrete Fourier series (DFS) expansion in a manner analogous to the computation of the complex Fourier series for the CT signals. The orthogonal basis function for the DFS is $W_N^{-n} = e^{j2\pi n/N}$ so that the decomposition of $x(n)$ into a sum of N harmonically related complex exponentials is expressed as

$$x(n) = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N} = \sum_{k=0}^{N-1} c_k W_N^{-kn} \tag{3.42}$$

where c_k represents the discrete Fourier series coefficients. Multiplying both sides of 3.42 by W_N^{mn}, summing over one period, and using the fact that

$$\sum_{k=0}^{N-1} W_N^{n(k-m)} = \begin{cases} N, & k = m \\ 0, & k \neq m \end{cases}$$

TABLE 3.8 Properties of the Discrete Fourier Series

Property	Signal Description	Discrete Fourier Series Coefficients
Linearity	$\sum_{m=0}^{Q} \beta_m x_m(n)$	$\sum_{m=0}^{Q} \beta_m c_{m,k}$
Time shift	$x(n-m)$	$c_k W_N^{km}$
Time-reversal	$x(-n)$	c_{-k}
Modulation	$x(n)\, W_N^{-mn}$	c_{k-m}
Real $x(n)$	$x(n)$	$c_k = c_k^*$
$x_e(n)$ is an even DT signal	$x_e(n)$	$\mathrm{Re}(c_k)$
$x_0(n)$ is an odd DT signal	$x_o(n)$	$j\{\mathrm{Im}(c_k)\}$

results in

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, W_N^{kn}, \quad k = 0, 1, 2, \ldots, N-1 \tag{3.43}$$

The DFS coefficients $\{c_k\}$ provide the description of $x(n)$ in the frequency domain since c_k represents the amplitude and phase spectra associated with the kth harmonic component. Note that $\{c_k\}$ is a periodic sequence with fundamental period N, that is, $c_k = c_{k+N}$. Hence, any N consecutive samples of the signal or its DFS coefficients provide a complete description of the signal in the time or frequency domains. As shown in Table 3.8, the properties of the DFS follow closely those given for the DTFT. One of the main advantages of DFS over the DTFT is the replacement of the infinite summation in the DTFT by a finite sum in the DFS, thus allowing the computations of DFS and inverse DFS by digital computers.

Consider a periodic DT signal with period N, then its average power is

$$P = \frac{1}{N}\sum_{n=0}^{N-1} |x(n)|^2 \tag{3.44}$$

Using Equation 3.42 in 3.44 gives

$$P = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x^*(n) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left(\sum_{k=0}^{N-1} c_k^* W_N^{kn}\right) = \sum_{k=0}^{N-1} |c_k|^2 \tag{3.45}$$

This is a Parseval's relation for the DT periodic signals, which indicates that the average power in $x(n)$ is the sum of the harmonics power. Consequently, the sequence $|c_k|^2$ is the power spectral density as this represents the distribution of power as a function of frequency.

3.1.10.3 The Discrete Fourier Transform[6,8,13]

The discrete Fourier transform (DFT) is an important and extremely powerful technique for analyzing DT signals. Contrary to the DTFT, the DFT is applicable to finite-length sequences and it produces finite-length discrete spectra. Consequently, this transformation is amenable to digital computations and it is suitable for use in digital hardware implementations. As a result of its practicability, DFT has become a very useful tool for analyzing various waveforms or data that arise in many disciplines.

The DFT is a mapping of an N sample sequence, $x(n)$, into another N sequence $X(k)$ in the frequency domain, that is,

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn}, \quad k = 0, 1, \ldots, N-1 \tag{3.46}$$

TABLE 3.9 Properties of DFT

Property	Signal Description	Discrete Fourier Transform
Linearity	$\sum_{m=0}^{Q} a_m x_m(n)$	$\sum_{m=0}^{Q} a_m X_m(k)$
Circular shift	$x[(n-m) \bmod N]$	$W_N^{km} X(k)$
Modulation	$W_N^{-qn} x(n)$	$X[(k-q) \bmod N]$
Time reversal	$x^*[-n \bmod N]$	$x^*(k)$
Complex conjugation	$x^*(n)$	$X^*[-k \bmod N]$
Circular convolution	$\sum_{m=0}^{N-1} x(m) y[(n-m) \bmod N]$	$X(k)Y(k)$
Multiplication	$x(n)y(n)$	$\frac{1}{N}\sum_{m=0}^{N-1} X(m)\, Y[(k-m) \bmod N]$
Parseval's theorem	$\sum_{k=0}^{N-1} x(n) y^*(n)$	$\frac{1}{N}\sum_{k=0}^{N-1} X(k)\, Y^*(k)$
Real part of signal	$\mathrm{Re}\{x(n)\}$	$\frac{1}{2}\{X(k) + X^*(N-k)\}$
Imaginary part of signal	$j\,\mathrm{Im}\{x(n)\}$	$\frac{1}{2}\{X(k) - X^*(N-k)\}$
Complex even	$x_{ce}(n) = \frac{1}{2}\{x(n) + x^*(N-n)\}$	$X_R(k)$
Complex odd	$x_{co}(n) = \frac{1}{2}\{x(n) - x^*(N-n)\}$	$jX_I(k)$
Any real signal	$x(n)$	$X(k) = X^*(N-k)$ $X_R(k) = X_R(N-k)$ $X_I(k) = -X_I(N-k)$ $\lvert X(k)\rvert = \lvert X(N-k)\rvert$ $\angle X(k) = -\angle X(N-k)$

$X(k)$ is called the kth harmonic and this exists provided all the samples of $x(n)$ are bounded. In order to recover $x(n)$ from $X(k)$ we need to perform inverse transformation. Thus, the inverse discrete Fourier transform (IDFT) of $X(k)$ is defined as

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, W_N^{-kn}, \quad n = 0, 1, \ldots, N-1 \tag{3.47}$$

The DFT and DFS are conceptually different in the sense that the DFS is only applicable to periodic signals whereas DFT assumes that the signal is periodic, that is, there is an inherent windowing of nonperiodic signal if analyzed by the DFT technique. Also the scaling factor is applicable to the synthesis equation for the DFT operation, whereas it is used in the analysis equation in the DFS analysis.

The DFT properties bear strong resemblance to those of the DFS and DTFT as shown in Table 3.9.

The following should be noted in the DFT computation: Circular shift operation can be considered as wrapping the part of the sequence that falls outside of 0 to $N-1$ to the front of the sequence, that is, $x[(-n) \bmod N]$ is equivalent to $x(N-n)$.

As a result of the circular shift, linear convolution as given by Equation 3.40 is different from circular convolution given in Table 3.9. Thus,

$$x(n) \otimes_N y(n) = \sum_{m=0}^{N-1} x(m) y[(n-m) \bmod N] = \sum_{m=0}^{N-1} x[(n-m) \bmod N] y(m)$$

where $x[(n-m) \bmod N]$ is the reflected and circularly translated version of $x(n)$. However, by appropriate selecting the value of N, both the circular and linear convolution can be the same. Thus, if signals $x(n)$ and $y(n)$ are of length N_1 and N_2, respectively, then circular and linear convolution are the same provided $N \geq N_1 + N_2 - 1$.

Circular correlation can be implemented by circular convolution since the cross correlation of the two sequences $x(n)$ and $y(n)$ can be expressed as

$$R_{xy}(n) = x(n) \otimes_N y(n) = x(n) \otimes_N y^*[(-n) \bmod N]$$

From Equations 3.46 and 3.47, it is noted that to compute the DFT coefficients would require N^2 complex multiplications and $N(N-1)$ complex additions. This can be a computational load if N is very large. Fast Fourier transforms (FFTs) are different types of algorithms that have been developed to speed up the computation of the DFT coefficients. Readers can refer to many textbooks where the development of the various forms of the FFT algorithms has been discussed.[7,8] It can be shown that the number of computations required for the FFT algorithms can be expressed as a constant times $N * \log_2 N$. Consequently, there is a reduction in computation when an FFT algorithm is used for the DFT computation.

3.1.10.4 Remarks on the DFT Processing of Signals

Zero Padding[6,8]

The use of FFT for DFT coefficients computation imposes some constraints on the value of N, for example N has to be a power of 2 for radix-2 FFT algorithm. Also, circular convolution is an undesirable solution as it manifests from the IDFT of the product of the transform of two sequences. Zero-padding is a technique for remedying the above situations, that is, the zero-padding is used either to augment the sequence length so that either a radix-2 FFT algorithm can be used or to ensure that both the linear and circular convolution are the same. In addition, this procedure is also used to provide a better-looking display of the signal spectrum since the frequency spacing of the FFT samples decreases as N increases.

Error Sources[8,10]

The resultant spectrum of a sampled signal comprises the analog spectrum repeated at the integer multiples of the sampling frequency. The overlapping of the analog signal spectrum with its shifted versions causes *aliasing*. In practice, excessive errors due to aliasing are minimized by either increasing the sampling rate or prefiltering the signal to remove the high-frequency spectral components. Another source of error in the DFT processing of signals is the *spectral leakage*. This is caused by using a window to truncate an infinitely long signal to obtain a finite-length data for DFT processing. It is known that windowing the samples of a signal in the time domain transforms to convolution of sampled signal spectrum and the window spectrum in the frequency domain. Suppose the window width does not correspond to an integer multiple of periods of all the frequency components of a discrete-time signal, then a single frequency component will spread (leak) into other frequency locations in the DFT of the truncated data. This phenomenon causes spectral distortion and makes it difficult to determine whether or not two closely adjacent frequencies are present in a signal. Spectral resolution becomes better if the window width is increased, or by choosing a window function with low sidelobes. The *picket-fence effect* arises because only a finite number of frequency points of a continuous-frequency spectrum are produced by the DFT. It is therefore possible to miss the peak of a particular frequency component in a signal because it is located between two adjacent frequency points in the spectrum. Since the frequency spacing $\Delta f = F_s/N$, this problem can be alleviated by increasing N the number of DFT points while maintaining the same sampling rate, implying having more samples in the DFT or a employing zero-padding technique.

DFT Parameter Selection[6]

The sampling period, T_s, frequency resolution (spacing), Δf, and the DFT length, N, are the three parameters that must be specified in performing DFT signal processing. Since $F_s = N\Delta f = 1/T_s$, these parameters are related according to $\Delta f = 1/NT_s$. If T_x denotes the length of the sampled data, then $T_x = MT_s$. Thus, there is

no spectral distortion if $N\check{S}M = T_x/T_s$. Equivalently, there will be no spectral distortion if

$$\Delta f = \frac{1}{T} = \frac{1}{NT_s} \leq \frac{1}{MT_s} = \frac{1}{T_x}$$

Parameters for the DFT processing of a sampled continuous signal must be carefully selected to avoid spectral distortion due to aliasing or data truncation. Assuming a window width of T_x seconds and that the signal has a maximum bandwidth of B hertz, then based on the sampling theorem we would have negligible aliasing, provided $F_s = 1/T_s \geq 2B$. Spectral leakage due to sharp data truncation is avoided provided the frequency resolution is selected to satisfy $1/\Delta f = T \geq T_x$. Consequently, spectral distortion due to aliasing and spectral leakage can be avoided if the length of the DFT is selected to satisfy $N = F_s/\Delta f \geq 2BT_x$.

References

1. Soliman, S.S., and Srinath, M.D., *Continuous and Discrete Signals and Systems,* Prentice-Hall, 1998.
2. O'Flynn, M., and Moriarty, E., *Linear Systems: Time Domain and Transform Analysis,* Wiley, 1987.
3. Lathi, B.P., *Modern Digital and Analog Communication Systems,* Oxford University Press, Third Edition, 1998.
4. Proakis, J.G., and Salehi, M., *Communication Systems Engineering,* Prentice-Hall, 1994.
5. Taylor, F.J., *Principles of Signals and Systems,* McGraw-Hill, 1994.
6. Carlson, G.E., *Signal and Linear System Analysis,* Wiley, Second Edition, 1998.
7. Proakis, J.G., and Manolakis, D.G., *Digital Signal Processing: Principles, Algorithms, and Applications,* Prentice-Hall, 1996.
8. Oppenheim, A.V., and Schafer, R.W., with Buck, J.R., *Discrete-Time Signal Processing,* Prentice-Hall, Second Edition, 1999.
9. Houts, R.C., *Signal Analysis in Linear Systems,* Saunders College Publishing, 1991.
10. Ziemer, R.E., Tranter, W.H., and Fannin, D.R., *Signals and Systems: Continuous and Discrete,* MacMillan Publishing Company, Third Edition, 1993.
11. Orfanidis, S.J., *Introduction to Signal Processing,* Prentice-Hall, 1996.
12. Haykin, S., and Veen, B.V., *Signals and Systems,* Wiley, 1999.
13. Taylor, F., and Mellot, J., *Hands-On Digital Signal Processing,* McGraw-Hill, 1998.

3.2 *z* Transforms and Digital Systems

Rolf Johansson

A digital system (or discrete-time system or sampled-data system) is a device such as a digital controller or a digital filter or, more generally, a system intended for digital computer implementation and usually with some periodic interaction with the environment and with a supporting methodology for analysis and design. Of particular importance for modeling and analysis are recurrent algorithms—for example, difference equations in input–output data—and the z transform is important for the solution of such problems.

The z **transform** is being used in the analysis of linear time-invariant systems and discrete time signals—for example, for digital control or filtering—and may be compared to the Laplace transform as used in the analysis of continuous-time signals and systems, a useful property being that the convolution of two time-domain signals is equivalent to multiplication of their corresponding z transforms. The z transform is important as a means to characterize a linear time-invariant system in terms of its pole–zero locations, its transfer function and Bode diagram, and its response to a large variety of signals. In addition, it provides important relationships between temporal and spectral properties of signals. The z transform generally appears in the analysis of difference equations as used in many branches of engineering and applied mathematics.

3.2.1 The z Transform

The z transform of the sequence $\{x_k\}_{-\infty}^{+\infty}$ is defined as the generating function

$$X(z) =]\{x\} = \sum_{k=-\infty}^{\infty} x_k z^{-k} \tag{3.48}$$

where the variable z has the essential interpretation of a forward shift operator so that

$$]\{x_{k+1}\} = z]\{x_k\} = zX(z) \tag{3.49}$$

The z transform is an infinite power series in the complex variable z^{-1} where $\{x_k\}$ constitutes a sequence of coefficients. As the z transform is an infinite power series, it exists only for those values of z for which this series converges and the *region of convergence* of $X(z)$ is the set of z for which $X(z)$ takes on a finite value. A sufficient condition for existence of the z transform is convergence of the power series

$$\sum_{k=-\infty}^{\infty} |x_k| \cdot |z^{-k}| < \infty \tag{3.50}$$

The region of convergence for a finite-duration signal is the entire z plane except $z = 0$ and $z = \infty$. For a one-sided infinite-duration sequence $\{x_k\}_{k=0}^{\infty}$, a number r can usually be found so that the power series converges for $|z| > r$. Then, the *inverse z transform* can be derived as

$$x_k = \frac{1}{2\pi i} \oint X(z) z^{k-1} dz \tag{3.51}$$

where the contour of integration encloses all singularities of $X(z)$. In practice, it is standard procedure to use tabulated results; some standard z transform pairs are to be found in Table 3.10.

3.2.2 Digital Systems and Discretized Data

Periodic sampling of signals and subsequent computation or storing of the results requires the computer to schedule sampling and to handle the resulting sequences of numbers. A measured variable $x(t)$ may be available only as periodic observations of $x(t)$ as sampled with a time interval T (the sampling period). The sample sequence can be represented as

$$\{x_k\}_{-\infty}^{\infty}, \quad x_k = x(kT) \quad \text{for } k = \ldots, -1, 0, 1, 2, \ldots \tag{3.52}$$

and it is important to ascertain that the sample sequence adequately represents the original variable $x(t)$; see Figure 3.16. For ideal sampling it is required that the duration of each sampling be very short and the sampled function may be represented by a sequence of infinitely short impulses $\delta(t)$ (the Dirac impulse). Let the sampled function of time be expressed thus:

$$x_\Delta(t) = x(t) \cdot T \sum_{k=-\infty}^{\infty} \delta(t - kT) = x(t) \cdot \text{Ш}_T(t) \tag{3.53}$$

where

$$\text{Ш}_T(t) \overset{\Delta}{=} T \sum_{k=-\infty}^{\infty} \delta(t - kT) \tag{3.54}$$

TABLE 3.10 Properties of the z Transform

z transform	$(\{f_k\}) = F(z)$
Convolution	$(\{f_k * g_k\}) = (\{f_k\}) \cdot (\{g_k\})$
	$(\{f_k \cdot g_k\}) = (\{f_k\}) * (\{g_k\})$
Forward shift	$(\{f_{k+1}\}) = z\ (\{f_k\}) = zF(z)$
Backward shift	$(\{f_{k-1}\}) = (\{f_k\}) = z^{-1}F(z)$
Linearity	$(\{af_k + bg_k\}) = a\ (\{f_k\}) + b\ (\{g_k\})$
Multiplication	$(\{a^k f_k\}) = F(a^{-1}z)$
Final value	$\lim_{k\to\infty} f_k = \lim_{k\to 1}(1 - z^{-1})F(z)$
Initial value	$f_0 = \lim_{z\to\infty} F(z)$

	Time Domain		z Transform
Impulse	$\delta_k = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$	$\leftrightharpoons$	$\{\delta_k\} = 1, \quad z \in C$
Step function	$\sigma_k = \begin{cases} 0, & k < 0 \\ 1, & k \geq 0 \end{cases}$	$\leftrightharpoons$	$](\sigma_k) = \dfrac{z}{z-1}, \quad \lvert z\rvert > 1$
Ramp function	$x_k = k \cdot \sigma_k$	$\leftrightharpoons$	$X(z) = \dfrac{z}{(z-1)^2}, \quad \lvert z\rvert > 1$
Exponential	$x_k = a^k \cdot \sigma_k$	$\leftrightharpoons$	$X(z) = \dfrac{z}{z-a}, \quad \lvert z\rvert > \lvert a\rvert$
Sinusoid	$x_k = \sin \omega k \cdot \sigma_k$	$\leftrightharpoons$	$X(z) = \dfrac{z \sin \omega}{z^2 - 2z\cos\omega + 1}, \quad \lvert z\rvert > 1$

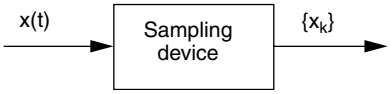


FIGURE 3.16 A continuous-time signal $x(t)$ and a sampling device that produces a sample sequence $\{x_k\}$.

and where the sampling period T is multiplied to ensure that the averages over a sampling period of the original variable x and the sampled signal x_Δ, respectively, are of the same magnitude. A direct application of the discretized variable $x_\Delta(t)$ in Equation 3.53 verifies that the spectrum of x_Δ is related to the z transform $X(z)$ as

$$X_\Delta(i\omega) = \wedge\{x(t) \cdot \sqcup\!\sqcup_T(t)\} = T\sum_{k=-\infty}^{\infty} x_k \exp(-i\omega kT) = TX(e^{i\omega T}) \tag{3.55}$$

Obviously, the original variable $x(t)$ and the sampled data are not identical, and thus it is necessary to consider the distortive effects of discretization. Consider the spectrum of the sampled signal $x_\Delta(t)$ obtained as the Fourier transform

$$X_\Delta(i\omega) = \wedge\{x_\Delta(t)\} = \wedge\{x(t)\} * \wedge\{\sqcup\!\sqcup_T(t)\} \tag{3.56}$$

where

$$\wedge\{\sqcup\!\sqcup_T(t)\} = \sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi}{T}k\right) = \frac{T}{2\pi}\sqcup\!\sqcup_{2\pi/T}(\omega) \tag{3.57}$$

so that

$$X_\Delta(i\omega) = \mathcal{F}\{x(t)\} * \mathcal{F}\{\sqcup\!\!\downarrow_T(t)\} = \sum_{k=-\infty}^{\infty} X\left[i\left(\omega - \frac{2\pi}{T}k\right)\right] \tag{3.58}$$

Thus, the Fourier transform X_Δ of the sampled variable has a periodic extension of the original spectrum $X(i\omega)$ along the frequency axis with a period equal to the sampling frequency $\omega_s = 2\pi/T$. There is an important result based on this observation known as the *Shannon sampling theorem*, which states that the continuous-time variable $x(t)$ may be reconstructed from the samples $\{x_k\}_{-\infty}^{+\infty}$ if and only if the sampling frequency is at least twice that of the highest frequency for which $X(i\omega)$ is nonzero. The original variable $x(t)$ may thus be recovered as

$$x(t) = \sum_{k=-\infty}^{\infty} x_k \frac{\sin\frac{\pi}{T}(t - kT)}{\frac{\pi}{T}(t - kT)} \tag{3.59}$$

The formula given in Equation 3.59 is called *Shannon interpolation*, which is often quoted though it is valid only for infinitely long data sequences and would require a noncausal filter to reconstruct the continuous-time signal $x(t)$ in real-time operation. The frequency $\omega_n = \omega_s/2 = \pi/T$ is called the *Nyquist frequency* and indicates the upper limit of distortion-free sampling. A nonzero spectrum beyond this limit leads to interference between the sampling frequency and the sampled signal (*aliasing*); see Figure 3.17.

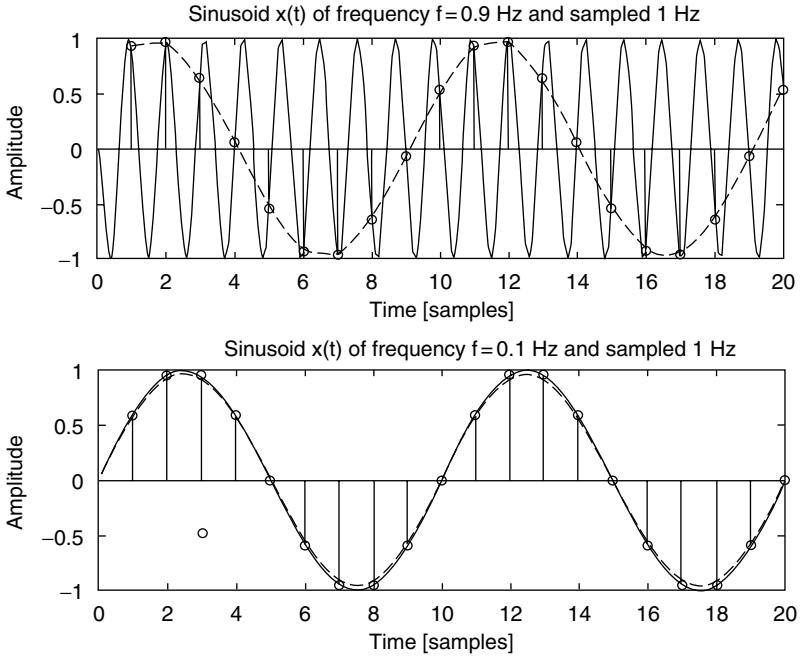


FIGURE 3.17 Illustration of aliasing appearing during sampling of a sinusoid $x(t) = \sin 2\pi \cdot 0.9t$ at the insufficient sampling frequency 1 Hz (sampling period $T = 1$) (*upper graph*). The sampled signal exhibits aliasing with its major component similar to a signal $x(t) = \sin 2\pi \cdot 0.1t$ sampled with the same rate (*lower graph*).

3.2.3 The Discrete Fourier Transform

Consider a finite length sequence $\{x_k\}_{k=0}^{N-1}$ that is zero outside the interval $0 \eth k \eth N-1$. Evaluation of the z transform $X(z)$ at N equally spaced points on the unit circle $z = \exp(i\omega_k T) = \exp[i(2\pi/NT)kT]$ for $k = 0, 1, \ldots, N-1$ defines the *discrete Fourier transform* (DFT) of a signal x with a sampling period h and N measurements:

$$X_k = \mathrm{DFT}\{x(kT)\} = \sum_{l=0}^{N-1} x_l \exp(-i\omega_l lT) = X(e^{i\omega_k T}) \tag{3.60}$$

Notice that the discrete Fourier transform $\{X_k\}_{k=0}^{N-1}$ is only defined at the discrete frequency points

$$\omega_k = \frac{2\pi}{NT}k, \quad \text{for } k = 0, 1, \ldots, N-1 \tag{3.61}$$

In fact, the discrete Fourier transform adapts the Fourier transform and the z transform to the practical requirements of finite measurements. Similar properties hold for the discrete Laplace transform with $z = \exp(sT)$, where s is the Laplace transform variable.

3.2.4 The Transfer Function

Consider the following discrete-time linear system with input sequence $\{u_k\}$ (stimulus) and output sequence $\{y_k\}$ (response). The dependency of the output of a linear system is characterized by the convolution-type equation and its z transform,

$$\begin{aligned} y_k &= \sum_{m=0}^{\infty} h_m u_{k-m} + v_k = \sum_{m=-\infty}^{\kappa} h_{k-m} u_m + v_k, \quad k = \ldots, -1, 0, 1, 2, \ldots \\ Y(z) &= H(z)U(z) + V(z) \end{aligned} \tag{3.62}$$

where the sequence $\{v_k\}$ represents some external input of errors and disturbances and with $Y(z) = \{y\}$, $U(z) = \{u\}$, $V(z) = \{v\}$ as output and inputs. The *weighting function* $h(kT) = \{h_k\}_{k=0}^{\infty}$, which is zero for negative k and for reasons of causality is sometimes called *pulse response* of the digital system (compare *impulse response* of continuous-time systems). The pulse response and its z transform, the *pulse transfer function*,

$$H(z) = \{h(kT)\} = \sum_{k=0}^{\infty} h_k z^{-k} \tag{3.63}$$

determine the system's response to an input $U(z)$; see Figure 3.18. The pulse transfer function $H(z)$ is obtained as the ratio

$$H(z) = \frac{X(z)}{U(z)} \tag{3.64}$$

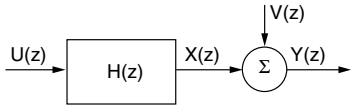


FIGURE 3.18 Block diagram with an assumed transfer function relationship $H(z)$ between input $U(z)$, disturbance $V(z)$, intermediate $X(z)$, and output $Y(z)$.

and provides the frequency domain input–output relation of the system. In particular, the Bode diagram is evaluated as $|H(z)|$ and arg $H(z)$ for $z = \exp(i\omega_k T)$ and for $|\omega_k| < \omega_n = \pi/T$, that is, when $H(z)$ is evaluated for frequency points up to the Nyquist frequency ω_n along the unit circle.

3.2.5 State-Space Systems

Alternatives to the input–output representations by means of transfer functions are the state-space representations. Consider the following finite dimensional discrete state-space equation with a state vector $x_k \in \mathrm{R}^n$, input $u_k \in \mathrm{R}^p$, and observations $y_n \in \mathrm{R}^m$.

$$\begin{cases} x_{k+1} = \Phi x_k + \Gamma u_k \\ \qquad\qquad\qquad k = 0, 1, \ldots \\ y_k = Cx_k + Du_k \end{cases} \tag{3.65}$$

with the pulse transfer function

$$H(z) = C(zI - \Phi)^{-1}\Gamma + D \tag{3.66}$$

and the output variable

$$Y(z) = C\sum_{k=0}^{\infty} \Phi^k z^{-k} x_0 + H(z)\,U(z) \tag{3.67}$$

where possible effects of initial conditions x_0 appear as the first term. Notice that the initial conditions x_0 can be viewed as the net effects of the input in the time interval $(-\infty, 0)$.

3.2.6 Digital Systems Described by Difference Equations (ARMAX Models)

An important class of nonstationary stochastic processes is one in which some deterministic response to an external input and a stationary stochastic process are superimposed. This is relevant, for instance, when the external input cannot be effectively described by some probabilistic distribution. A discrete-time model can be formulated in the form of a difference equation with an external input $\{u_k\}$ that is usually considered to be known:

$$y_k = -a_1 y_{k-1} - \cdots - a_n y_{k-n} + b_1 u_{k-1} + \cdots + b_n u_{k-n} + w_k + c_1 w_{k-1} + \cdots + c_n w_{k-n} \tag{3.68}$$

Application of the z transform permits formulation of Equation 3.68 as

$$A(z^{-1})\,Y(z) = B(z^{-1})\,U(z) + C(z^{-1})\,W(z) \tag{3.69}$$

where

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \cdots + a_n z^{-n} \\ B(z^{-1}) &= 1 + b_1 z^{-1} + \cdots + b_n z^{-n} \\ C(z^{-1}) &= 1 + c_1 z^{-1} + \cdots + c_n z^{-n} \end{aligned} \tag{3.70}$$

Stochastic models including the A polynomial, according to Equations 3.69 and 3.70, are known as **autoregressive (AR) models** and models including the C polynomial are known as **moving-average (MA) models**, whereas the B polynomial determines the effects of the external input (X). Notice that the term *moving average* is here somewhat misleading, as there is no restriction that the coefficients should add to 1 or that the coefficients are nonnegative. An alternative description is *finite impulse response* or *all-zero filter*.

Thus, the full model of Equation 3.69 is an **autoregressive moving average model** with external input (ARMAX) and its pulse transfer function $H(z) = B(z^{-1})/A(z^{-1})$ is stable if and only if the *poles*—that is, the complex numbers $z_1,\ldots, z_n$ solving the equation $A(z^{-1}) = 0$—are strictly inside the unit circle, that is, $|z_i| < 1$. The *zeros* of the system—that is, the complex numbers $z_1,\ldots, z_n$ solving the equation $B(z^{-1}) = 0$—may take on any value without any instability arising, although it is preferable to obtain zeros located strictly inside the unit circle, that is, $|z_i| < 1$ (*minimum-phase zeros*). By linearity $\{y_k\}$ can be separated into one purely deterministic process $\{x_k\}$ and one purely stochastic process $\{v_k\}$:

$$\begin{cases} A(z^{-1})X(z) = B(z^{-1})U(z) \\ \\ A(z^{-1})V(z) = C(z^{-1})W(z) \end{cases} \quad \text{and} \quad \begin{cases} y_k = x_k + v_k \\ \\ Y(z) = X(z) + V(z) \end{cases} \tag{3.71}$$

The type of decomposition (Equation 3.71) that separates the deterministic and stochastic processes is known as the *Wold decomposition*.

3.2.7 Prediction and Reconstruction

Consider the problem of predicting the output d steps ahead when the output $\{y_k\}$ is generated by the ARMA model,

$$A(z^{-1})Y(z) = C(z^{-1})W(z) \tag{3.72}$$

which is driven by a zero-mean white noise $\{w_k\}$ with covariance $\mathbb{E}\{w_i w_j\} = \sigma_w^2 \delta_{ij}$. In other words, assuming that observations $\{y_k\}$ are available up to the present time, how should the output d steps ahead be predicted optimally? Assume that the polynomials $A(z^{-1})$ and $C(z^{-1})$ are mutually prime with no zeros for $|z| \geq 1$. Let the C polynomial be expanded according to the *Diophantine equation*,

$$C(z^{-1}) = A(z^{-1})F(z^{-1}) + z^{-d}G(z^{-1}) \tag{3.73}$$

which is solved by the two polynomials

$$\begin{aligned} F(z^{-1}) &= 1 + f_1 z^{-1} + \cdots + f_{nF} z^{-nF}, \qquad n_F = d - 1 \\ G(z^{-1}) &= g_0 + g_1 z^{-1} + \cdots + g_{nG} z^{-nG}, \quad n_G = \max(n_A - 1,\ n_C - d) \end{aligned} \tag{3.74}$$

Interpretation of z^{-1} as a *backward shift operator* and application of Equations 3.72 and 3.73 permit the formulation

$$y_{k+d} = F(z^{-1})w_{k+d} + \frac{G(z^{-1})}{C(z^{-1})} y_k \tag{3.75}$$

Let us, by $\hat{y}_{k+d|k}$, denote linear d-step predictors of y_{k+d} based upon the measured information available at time k. As the zero-mean term $F(z^{-1})w_{k+d}$ of Equation 3.75 is unpredictable at time k, it is natural to suggest the following d-step predictor:

$$\hat{y}_{k+d|k} = \frac{G(z^{-1})}{C(z^{-1})} y_k \tag{3.76}$$

The prediction error satisfies

$$\begin{aligned} \varepsilon_{k+d} &= (\hat{y}_{k+d|k} - y_{k+d}) \\ &= \frac{G(z^{-1})}{C(z^{-1})} y_k - \frac{A(z^{-1})F(z^{-1}) + z^{-d}G(z^{-1})}{C(z^{-1})} y_{k+d} \\ &= -F(z^{-1})w_{k+d} \end{aligned} \tag{3.77}$$

Let $\{\cdot | \wedge_k\}$ denote the *conditional mathematical expectation* relative to the measured information available at time k. The conditional mathematical expectation and the covariance of the d-step prediction relative to available information at time k is

$$\begin{aligned} \%\{\hat{y}_{k+d|k} - y_{k+d} \,|\, \wedge_k\} &= \%\{-F(z^{-1})w_{k+d} \,|\, \wedge_k\} = 0 \\ \%\{(\hat{y}_{k+d|k} - y_{k+d})^2 \,|\, \wedge_k\} &= \%\{[F(z^{-1})w_{k+d}]^2 \,|\, \wedge_k\} \\ &= \%\{(w_{k+d} + f_1 w_{k+d-1} + \cdots + f_{d-1}w_{k+1})^2 \,|\, \wedge_k\} \\ &= (1 + f_1^2 + \cdots + f_{nF}^2)\sigma_w^2 = 0 \end{aligned} \tag{3.78}$$

It follows that the predictor of Equation 3.76 is unbiased and that the prediction error only depends on future, unpredictable noise components. It is straightforward to show that the predictor of Equation 3.76 achieves the lower bound of Equation 3.78 and that the predictor of Equation 3.76 is optimal in the sense that the prediction error variance is minimized.

Example 3.1—An Optimal Predictor for a First-Order Model

Consider for the first-order ARMA model

$$y_{k+1} = -a_1 y_k + w_{k+1} + c_1 w_k \tag{3.79}$$

The variance of a one-step-ahead predictor $\hat{y}_{k+1|k}$ is

$$\begin{aligned} \%\{(\hat{y}_{k+1|k} - y_{k+1})^2 \,|\, \wedge_k\} &= \%\{(\hat{y}_{k+1|k} + a_1 y_k - c_1 w_k)^2 \,|\, \wedge_k\} + \%\{w_{k+1}^2 \,|\, \wedge_k\} \\ &= \%\{(\hat{y}_{k+1|k} + a_1 y_k - c_1 w_k)^2 \,|\, \wedge_k\} + \sigma_w^2 \geq \sigma_w^2 \end{aligned} \tag{3.80}$$

The optimal predictor satisfying the lower bound in Equation 3.80 is obtained from Equation 3.80 as

$$\hat{y}^o_{k+1|k} = -a_1 y_k + c_1 w_k \tag{3.81}$$

which, unfortunately, is not realizable as it stands because w_k is not available to measurement. Therefore, the noise sequence $\{w_k\}$ has to be substituted by some function of the observed variable $\{y_k\}$. A linear predictor chosen according to Equation 3.76 is

$$\hat{y}_{k+1|k} = \frac{G(z^{-1})}{C(z^{-1})} y_k = \frac{c_1 - a_1}{1 + c_1 z^{-1}} y_k \tag{3.82}$$

3.2.8 The Kalman Filter

Consider the linear state-space model

$$\begin{aligned} x_{k+1} &= \Phi x_k + v_k, \quad x_k \in \mathrm{R}^n \\ y_k &= C x_k + w_k, \quad y_k \in \mathrm{R}^m \end{aligned} \tag{3.83}$$

where $\{v_k\}$ and $\{w_k\}$ are assumed to be independent zero-mean white-noise processes with covariances Σ_v and Σ_w, respectively. It is assumed that $\{y_k\}$, but not $\{x_k\}$, is available to measurement and that it is desirable to predict $\{x_k\}$ from measurements of $\{y_k\}$.

Introduce the state predictor,

$$\begin{aligned} \hat{x}_{k+1|k} &= \Phi \hat{x}_{k|k-1} - K_k(\hat{y}_k - y_k), \quad \hat{x}_{k|k-1} \in \mathrm{R}^n \\ \hat{y}_k &= C\hat{x}_{k|k-1}, \qquad y_k \in \mathrm{R}^m \end{aligned} \tag{3.84}$$

The predictor of Equation 3.84 has the same dynamics matrix Φ as the state-space model of Equation 101.83 and, in addition, there is a correction term $K_k(\hat{y}_k - y_k)$ with a factor K_k to be chosen. The prediction error is

$$\tilde{x}_{k+1|k} = \hat{x}_{k+1|k} - x_{k+1} \tag{3.85}$$

The prediction-error dynamics is

$$\tilde{x}_{k+1} = (\Phi - K_k C)\tilde{x}_k + v_k - K_k w_k \tag{3.86}$$

The mean prediction error is governed by the recursive equation

$$\%\{\tilde{x}_{k+1}\} = (\Phi - K_k C)\%\{\tilde{x}_k\} \tag{3.87}$$

and the mean square error of the prediction error is governed by

$$\begin{aligned} \%\{\tilde{x}_{k+1}\tilde{x}_{k+1}^T\} &= \%\{[(\Phi - K_k C)\tilde{x}_k + v_k - K_k w_k][(\Phi - K_k C)\tilde{x}_k + v_k - K_k w_k]^T\} \\ &= (\Phi - K_k C)\%\{\tilde{x}_k \tilde{x}_k^T\}(\Phi - K_k C)^T + \Sigma_v + K_k \Sigma_w K_k \end{aligned} \tag{3.88}$$

If we denote

$$P_k = \%\{\tilde{x}_k \tilde{x}_k^T\}, \qquad Q_k = \Sigma_w + C P_k C^T \tag{3.89}$$

then Equation 3.88 is simplified to

$$P_{k+1} = \Phi P_k \Phi^T - K_k C P_k \Phi - \Phi^T P_k C^T K_k^T + \Sigma_v + K_k Q_k K_k^T \tag{3.90}$$

By completing squares of terms containing K_k we find

$$P_{k+1} = \Phi P_k \Phi^T + \Sigma_v - \Phi P_k C^T Q_k^{-1} C P_k \Phi^T + (K_k - \Phi P_k C^T Q_k^{-1}) Q_k (K_k - \Phi P_k C^T Q_k^{-1})^T \tag{3.91}$$

where only the last term depends on K_k. Minimization of P_{k+1} can be done by choosing K_k such that the positive semidefinite K_k-dependent term in Equation 3.91 disappears. Thus P_{k+1} achieves its lower bound for

$$K_k = \Phi P_k C^T (\Sigma_w + C P_k C^T)^{-1} \tag{3.92}$$

and the *Kalman filter* (or *Kalman–Bucy filter*) takes the form

$$\begin{aligned} \hat{x}_{k+1|k} &= \Phi \hat{x}_{k|k-1} - K_k(\hat{y}_k - y_k) \\ \hat{y}_k &= C\hat{x}_{k|k-1}, \qquad K_k = \Phi P_k C^T (\Sigma_w + C P_k C^T)^{-1} \\ P_{k+1} &= \Phi P_k \Phi^T + \Sigma_v - \Phi P_k C^T (\Sigma_w + C P_k C^T)^{-1} C P_k \Phi^T \end{aligned} \tag{3.93}$$

which is the optimal predictor in the sense that the mean square error (Equation 3.88) is minimized in each step.

Example 3.2—Kalman Filter for a First-Order System

Consider the state-space model

$$x_{k+1} = 0.95 x_k + v_k, \qquad y_k = x_k + w_k \tag{3.94}$$

where $\{v_k\}$ and $\{w_k\}$ are zero-mean white-noise processes with covariances $\mathcal{E}\{v_k^2\} = 1$ and $\mathcal{E}\{w_k^2\} = 1$, respectively.

The Kalman filter takes on the form

$$\begin{aligned} \hat{x}_{k+1|k} &= 0.95\hat{x}_{k|k-1} - K_k(\hat{x}_{k|k-1} - y_k) \\ K_k &= \frac{0.95 P_k}{1 + P_k} \\ P_{k+1} &= 0.95^2 P_k + 1 - \frac{0.95^2 P_k^2}{1 + P_k} \end{aligned} \tag{3.95}$$

The result of one such realization is shown in Figure 3.19.

Defining Terms

Autoregressive (AR) model: An autoregressive time series of order n is defined via $y_k = -\Sigma_{m=1}^{n} a_m y_{k-m} + w_k$. The sequence $\{w_k\}$ is usually assumed to consist of zero-mean identically distributed stochastic variables w_k.

Autoregressive moving average (ARMA) model: An autoregressive moving average time series of order n is defined via $y_k = -\Sigma_{m=1}^{n} a_m y_{k-m} + \Sigma_{m=0}^{n} c_m w_{k-m}$. The sequence $\{w_k\}$ is usually assumed to consist of zero-mean identically distributed stochastic variables w_k.

Discrete Laplace transform: The discrete Laplace transform is a counterpart to the Laplace transform with application to discrete signals and systems. The discrete Laplace transform is obtained from the z transform by means of the substitution $z = \exp(sT)$, where T is the sampling period.

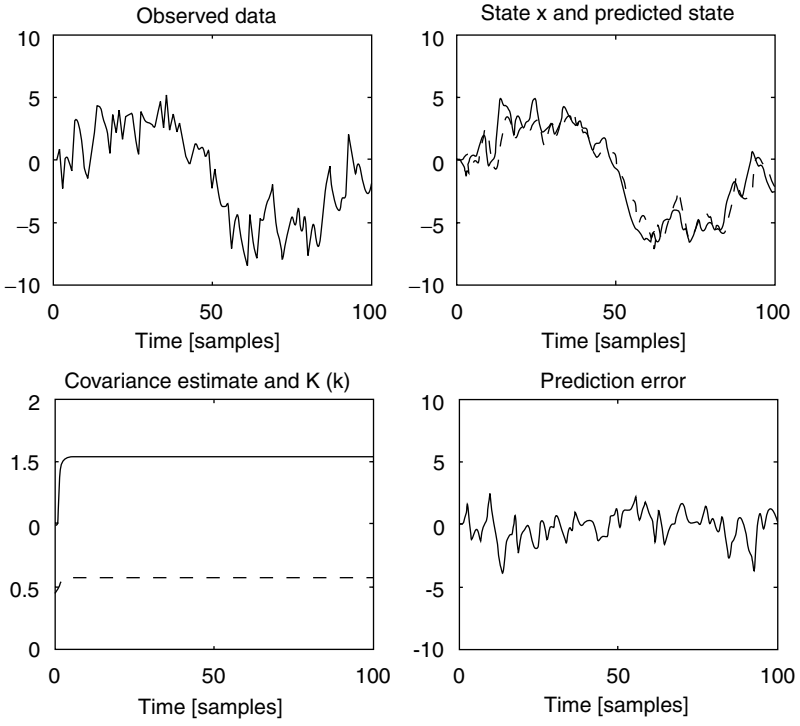


FIGURE 3.19 Kalman filter applied to one-step-ahead prediction of x_{k+1} in Equation 3.94. The observed variable $\{y_k\}$, the state $\{x_k\}$, and the predicted state $\{\hat{x}_k\}$, the estimated variance $\{P_k\}$ and $\{K_k\}$, and the prediction error $\{\tilde{x}_k\}$ are shown in a 100-step realization of the stochastic process. (From Johansson, R. 1993. *System Modeling and Identification.* Prentice-Hall, Englewood Cliffs, NJ.)

Moving average (MA) process: A moving average time series of order n is defined via $y_k = \Sigma_{m=0}^{n} c_m w_{k-m}$. The sequence $\{w_k\}$ is usually assumed to consist of zero-mean identically distributed stochastic variables w_k.

Rational model: AR, MA, ARMA, and ARMAX are commonly referred to as rational models.

Time Series: A sequence of random variable $\{y_k\}$, where k belongs to the set of positive and negative integers.

***z* transform:** A generating function applied to sequences of data and evaluated as a function of the complex variable z with interpretation of frequency.

References

Box, G. E. P. and Jenkins, G. M. 1970. *Time Series Analysis: Forecasting and Control.* Holden-Day, San Francisco, CA.

Hurewicz, W. 1947. Filters and servo systems with pulsed data. In *Theory of Servomechanisms,* H. M. James, N. B. Nichols, and R. S. Philips, eds., McGraw-Hill, New York.

Jenkins, G. M. and Watts, D. G. 1968. *Spectral Analysis and Its Applications.* Holden-Day, San Francisco, CA.

Johansson, R. 1993. *System Modeling and Identification.* Prentice-Hall, Englewood Cliffs, NJ.

Jury, E. I. 1956. Synthesis and critical study of sampled-data control systems. *AIEE Trans.* 75: 141–151.

Kalman, R. E. and Bertram, J. E. 1958. General synthesis procedure for computer control of single and multi-loop linear systems. *Trans. AIEE.* 77: 602–609.

Kolmogorov, A. N. 1939. Sur l'interpolation et extrapolation des suites stationnaires. *C. R. Acad. Sci.* 208: 2043–2045.
Ragazzini, J. R. and Zadeh, L. A. 1952. The analysis of sampled-data systems. *AIEE Trans.* 71:225–234.
Tsypkin, Y. Z. 1950. Theory of discontinuous control. *Avtomatika i Telemekhanika.* Vol. 5.
Wiener, N. 1949. *Extrapolation, Interpolation and Smoothing of Stationary Time Series with Engineering Applications.* John Wiley & Sons, New York.

Further Information

Early theoretical efforts developed in connection with servomechanisms and radar applications (Hurewicz, 1947). Tsypkin (1950) introduced the discrete Laplace transform and the formal *z* transform definition was introduced by Ragazzini and Zadeh (1952) with further developments by Jury (1956). Much of prediction theory was originally developed by Kolmogorov (1939) and Wiener (1949) whereas state-space methods were forwarded by Kalman and Bertram (1958). Pioneering textbooks on time-series analysis and spectrum analysis are provided by Box and Jenkins (1970) and Jenkins and Watts (1968).

Detailed accounts of time-series analysis and the *z* transform and their application to signal processing are to be found in

- Oppenheim, A. V. and Schafer, R. W. 1989. *Discrete-Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ.
- Proakis, J. G. and Manolakis, D. G. 1989. *Introduction to Digital Signal Processing.* Maxwell MacMillan Int. Ed., New York.

Theory of time-series analysis and its application to discrete-time control is to be found in

- Åström, K. J. and Wittenmark, B. 1990. *Computer-Controlled Systems,* 2nd ed., Prentice-Hall, Englewood Cliffs, NJ.

Theory of time-series analysis and methodology for determination and validation of discrete-time models and other aspects of system identification are to be found in

- Johansson, R. 1993. *System Modeling and Identification.* Prentice-Hall, Englewood Cliffs, NJ.

Good sources to monitor current research are

- *IEEE Transactions on Automatic Control*
- *IEEE Transactions on Signal Processing*

Examples of easy-to-read survey articles for signal processing applications are

- Cadzow, J. A. 1990. Signal processing via least-squares error modeling. *IEEE ASSP Magazine.* 7:12–31, October.
- Schroeder, M. R. 1984. Linear prediction, entropy, and signal analysis. *IEEE ASSP Magazine.* 1:3–11, July.

3.3 Continuous- and Discrete-Time State-Space Models

Kam K. Leang, Qingze Zou, and Santosh Devasia

3.3.1 Introduction

In this section we introduce the modeling of continuous- and discrete-time systems using the state-space approach. The state-space approach is a technique that uses a set of first order differential equations to represent the behavior of a system in the time-domain. The state-space approach has an advantage over frequency-domain approaches such as the transfer-function approach: it can be used to model linear,

nonlinear, time-varying, and multivariable systems, whereas the transfer-function approach is suited to linear time-invariant (LTI) systems [1, Chapter 3]. In addition, models expressed in first order state-space form in the time-domain can be readily solved by a digital computer or microprocessor, which makes this approach quite useful for the design and control of modern mechatronic systems. Furthermore, there is a wide variety of available computer software, such as MATLAB [2], that take advantage of the state-space form for analyzing and solving design problems. Therefore, the state-space approach can be used to investigate the behavior and facilitate in the design of both continuous- and discrete-time systems, the fundamentals of which will be the focus of this section.

In the following, we begin with an example: the modeling of a piezoceramic actuator and use the example throughout the section. The concept of a system state is introduced and we explain the state-space equation for linear systems and present its solution. The topic of linearization of nonlinear systems is briefly mentioned. The relationships between time- and frequency-domain models are discussed and a procedure for obtaining a state-space model using experimental frequency-domain (frequency-response) data is presented. This section closes with a discussion of discrete-time state-space modeling and concluding remarks. Useful MATLAB commands are also included as footnotes.

3.3.2 States and the State-Space

3.3.2.1 An Example Piezoceramic Actuator

We begin by modeling a piezoceramic actuator, which is an example mechatronic (electromechanical) system. When a voltage is applied to a piezoceramic material, its dimension changes. This change in dimension can be used to precisely position an object or tool (such as a sensor), therefore making piezoceramics suitable actuators for a wide variety of applications. For example, due to their ability to achieve positioning with sub-nanometer level precision, piezoceramic actuators have become ideal for emerging nanotechnologies. In particular, a piezo-tube actuator is used in scanning probe microscopes (SPMs, see Figure 3.20) to precisely position a probe tip for high-precision nanofabrication, surface modification, and the acquisition of images of atoms [3]. The probe tip can be positioned in the three coordinate axes (x, y, and z), with each motion controlled by an independent voltage source ($V_x(t)$, $V_y(t)$, and $V_z(t)$). Scanning of the probe is performed parallel to the sample surface along the x- and y-axis; the z-axis movement allows motion of the probe perpendicular to the sample surface. An accurate mathematical model of the dynamics of a piezo-tube actuator is required for the analysis and design of SPM systems. A designer can exploit the known information of the system from its model to improve or optimize a design for building faster and more reliable SPMs. For example, an approach that has been

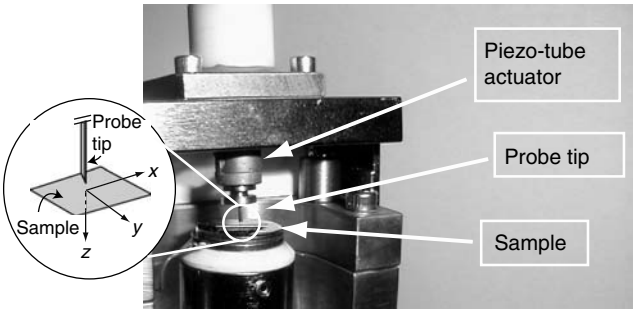


FIGURE 3.20 The main components of a scanning probe microscope (SPM) used for surface analysis, which includes the piezo-tube actuator, the probe tip, and the sample. The configuration of the probe tip and sample with respect to the coordinate axes (x, y, and z) are shown in the magnified view.

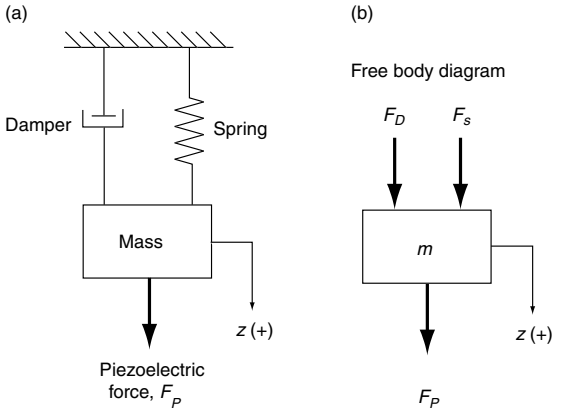


FIGURE 3.21 (a) A simple *lumped model* of the piezo-tube actuator modeled along the *z*-axis consisting of a mass, a spring, and a damper [4]. The positive *z*-direction is indicated by the arrow and the "+" sign. (b) The forces acting on the mass (free body diagram).

successfully implemented is the inversion-based control method, which finds the inputs required to achieve exact tracking by inverting the system model [3]. This technique works best when the dynamics of the system are well characterized and understood. In general, the analysis and design of control systems also requires a system model. Thus for analysis and design, it is crucial to obtain an accurate mathematical model that describes the behavior of a system. Modeling of the example piezo-tube system is considered in the following.

Simple Model of a Piezo-Tube Actuator

We will model the dynamics of the piezo-tube actuator along the *z*-axis where the input is the applied voltage $V_z(t)$ and the output of the system is the displacement of the probe tip $z(t)$. We begin the modeling by simplifying the system as an isolated mass, an ideal spring, and a damper as shown in Figure 3.21a. The entire mass of the piezo-tube is lumped into one mass element m, the internal elastic behavior of the piezo-tube is modeled as a spring, and the structural damping in the piezo-tube is modeled as a damper or a viscous friction element (such models are referred to as *lumped models* [4]). A mathematical relationship between the applied voltage $V_z(t)$ and the displacement of the probe tip $z(t)$ can be obtained using physical laws. Applying Newton's second law (the sum of all external forces F_i acting on a body is equal the product of its mass m and acceleration $\ddot{z}(t)$) we can write the equation of motion as

$$\sum_i F_i(t) = m\ddot{z}(t) \tag{3.96}$$

As shown in Figure 3.21b (the free body diagram), there are three external forces acting on the piezo-tube. First, the force exerted by the spring is assumed to be proportional to the displacement of the probe tip, that is,

$$F_s(t) = -kz(t) \tag{3.97}$$

where k is the spring constant with SI units [N/m]. Second, the damping force is considered to be proportional to the velocity of the probe tip $\dot{z}(t)$, that is,

$$F_D(t) = -c\dot{z}(t) \tag{3.98}$$

where c is the viscous friction or damping coefficient with SI units [N · s/m]. Third, induced strain ε in the piezoceramic material is proportional to the applied voltage $V_z(t)$ [5], and by Hooke's Law, the induced stress σ is proportional to the induced strain ε. Hence, the induced force $F_p(t)$ (stress σ times the cross-sectional area) is proportional to the applied voltage $V_z(t)$, that is,

$$F_p(t) = bV_z(t) \tag{3.99}$$

where b is a constant with SI units [N/V]. Rewriting Equation 3.96 in terms of the three external forces, the equation of motion becomes

$$\sum_{i=1}^{3} F_i(t) = F_S(t) + F_D(t) + F_p(t) = -kz(t) - c\dot{z}(t) + bV_z(t) = m\ddot{z}(t) \tag{3.100}$$

which is called the *mass-spring-damper* model. Note that the relationship between the input voltage $V_z(t)$ and the displacement $z(t)$ of the probe tip (i.e., the model of the dynamics) is a second order differential equation. The response of the probe tip (displacement of mass m) to an applied voltage $V_z(t)$ can be obtained in the frequency-domain by using the Laplace transform technique [6, Chapter 2, Section 5]; however, the state-space approach can be used to obtain the solution directly in the time-domain. In the remaining sections, the state-space approach to modeling is presented and the *mass-spring-damper* model of the piezo-tube actuator will be used as an example.

3.3.3.2 States of a System

We begin by introducing the concept of a state, which is the basis for the state-space approach. In general, a state can be defined as the following:

> The state $x(t_0)$ of a dynamic system at time t_0 is a set of variables that, together with the input $u(t)$, for $t \geq t_0$, determines the behavior of the system for all $t \geq t_0$ [7, Chapter 2, Section 1.1].

Fundamental to this definition is the notion that the state summarizes the current configuration of a system. Therefore, the memory of a dynamical system is preserved in the state variables at the current time t_0 (called initial condition), and the future behavior of the system is determined by the initial condition $x(t_0)$ and the applied input $u(t)$, for $t \geq t_0$. The state of a system can be written as the set

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \tag{3.101}$$

where n is the number of states.* Any set of variables that satisfy the above definition can be a valid state, hence the state is not unique [8, Chapter 2, Section 2].

Example

The state variables required to describe the mass-spring-damper system can be chosen as the position $z(t)$ and velocity $\dot{z}(t)$ of the mass. We can write the state vector as

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix} \tag{3.102}$$

*For a discussion on the minimal set of states required to describe a system (minimal realization), see [7, Chapter 7].

where the number of states is two ($n = 2$). If the position $z(t)$ and velocity $\dot{z}(t)$ of the mass are known at time t_0, along with the applied voltage $V_z(t)$ defined for $t \geq t_0$, then the future behavior of the system (i.e., the state $x(t)$) can be determined by solving the differential Equation 3.100.

3.3.3.3 The Linear State-Space Equation and Its Solution

For a linear system, the evolution of the states of a system over time can be described by a set of linear first order differential equations of the form:

$$\begin{aligned}
x_1(t) &= \frac{dx_1(t)}{dt} = a_{11}(t)x_1(t) + \cdots + a_{1n}(t)x_n(t) + b_{11}(t)u_1(t) + \cdots b_{1p}(t)u_p(t) \\
x_2(t) &= \frac{dx_2(t)}{dt} = a_{21}(t)x_1(t) + \cdots + a_{2n}(t)x_n(t) + b_{21}(t)u_1(t) + \cdots b_{2p}(t)u_p(t) \\
&\vdots \\
x_n(t) &= \frac{dx_n(t)}{dt} = a_{n1}(t)x_1(t) + \cdots + a_{nn}(t)x_n(t) + b_{n1}(t)u_1(t) + \cdots b_{np}(t)u_p(t)
\end{aligned} \tag{3.103}$$

where n is the number of states (or the order of the system) and p is the number of inputs.* Defining the input vector as

$$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_p(t) \end{bmatrix} \tag{3.104}$$

and the state vector $x(t)$ as defined in Equation 3.101, the set of first order differential equations given by Equation 3.103 can be rewritten in compact matrix form as [8, Chapter 2, Section 2]

$$\begin{aligned}
\dot{x}(t) &= \begin{bmatrix} a_{11}(t) & a_{12}(t) & \cdots & a_{1n}(t) \\ a_{21}(t) & a_{22}(t) & \cdots & a_{2n}(t) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}(t) & a_{n2}(t) & \cdots & a_{nn}(t) \end{bmatrix} x(t) + \begin{bmatrix} b_{11}(t) & b_{12}(t) & \cdots & b_{1p}(t) \\ b_{21}(t) & b_{22}(t) & \cdots & b_{2p}(t) \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1}(t) & b_{n2}(t) & \cdots & b_{np}(t) \end{bmatrix} u(t) \\
&= A(t)x(t) + B(t)u(t)
\end{aligned} \tag{3.105}$$

where $A(t)$ is an $n \times n$ matrix and $B(t)$ is an $n \times p$ matrix. For a system defined with q outputs $y(t)$, which are assumed to be a linear combination of the state $x(t)$ and input $u(t)$, we can write the output equation as

$$\begin{aligned}
y(t) &= \begin{bmatrix} c_{11}(t) & c_{12}(t) & \cdots & c_{1n}(t) \\ c_{21}(t) & c_{22}(t) & \cdots & c_{2n}(t) \\ \vdots & \vdots & \ddots & \vdots \\ c_{q1}(t) & c_{q2}(t) & \cdots & c_{qn}(t) \end{bmatrix} x(t) + \begin{bmatrix} d_{11}(t) & d_{12}(t) & \cdots & d_{1p}(t) \\ d_{21}(t) & d_{22}(t) & \cdots & d_{2p}(t) \\ \vdots & \vdots & \ddots & \vdots \\ d_{q1}(t) & d_{q2}(t) & \cdots & d_{qp}(t) \end{bmatrix} u(t) \\
&= C(t)x(t) + D(t)u(t)
\end{aligned} \tag{3.106}$$

*Given a higher order differential equation, a set of first order differential equations can be obtained by a procedure known as reduction to first order as presented in [9].

where $C(t)$ is a $q \times n$ matrix and $D(t)$ is a $q \times p$ matrix. In general, the matrices $A(t)$, $B(t)$, $C(t)$, and $D(t)$ are time varying; however, in this chapter we will only consider the time-invariant case where A, B, C, and D are constant matrices, then Equations 3.107 and 3.108 are called the linear time-invariant (LTI) state and output equations, respectively.*

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.107}$$

$$y(t) = Cx(t) + Du(t) \tag{3.108}$$

The response of the system to an applied input can be quantified by the evolution of the system state $x(t)$ and the output $y(t)$. The state-space Equation 3.107 is a set of first order differential equations in matrix form, which can be solved in time for a given initial condition $x(t_0)$ as [8, Chapter 3]

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)\,d\tau \tag{3.109}$$

Note that solution (3.109) is the sum of two terms: the first term is the effect of initial condition $x(t_0)$ and the second is the effect of the applied input $u(t)$ between $t_0 \le \tau \le t$.† Using the output Equation 3.108 and the state solution given by (3.109), the output $y(t)$ becomes

$$y(t) = Ce^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} Ce^{A(t-\tau)}Bu(\tau)\,d\tau + Du(t) \tag{3.110}$$

The system response $y(t)$ to an applied input $u(t)$ is characterized by the system matrices (A, B, C, D). For example, the output $y(t)$ will be bounded for any bounded input if the system is stable and the system is stable if the real parts of all the eigenvalues of A are less than zero (strictly negative) [8, Chapter 4, Section 4].‡

Example

For the mass-spring-damper example system, the state-space equation can be found by differentiating the states $x(t)$ defined in Equation 3.102 and using the equation of motion 3.100 to obtain

$$\begin{aligned} x_1(t) = z(t) &= x_2(t) \\ x_1(t) = \dot{z}(t) &= -\left(\frac{k}{m}\right)z(t) - \left(\frac{c}{m}\right)\dot{z}(t) + \left(\frac{b}{m}\right)V_z(t) = -\left(\frac{k}{m}\right)x_1(t) - \left(\frac{c}{m}\right)x_2(t) + \left(\frac{b}{m}\right)u(t) \end{aligned} \tag{3.111}$$

We choose the position of the mass $z(t)$ to be the output of the system, and write the state-space and output equation in the form given by Equations 3.107 and 3.108 as

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -(k/m) & -(c/m) \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ b/m \end{bmatrix} u(t) \tag{3.112}$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) \tag{3.113}$$

*For a detailed discussion of the solution of linear time-varying equations, see [7, Chapter 4, Section 5].

†The MATLAB command `lsim` simulates the time response of LTI models to arbitrary inputs.

‡The MATLAB command `eig (A)` returns the eigenvalues of the system matrix `A`.

3.3.3.4 Linearization of Nonlinear Systems

A general form of the state-space equation (for nonlinear systems) is

$$\dot{x}(t) = g(x, u) \tag{3.114}$$

$$y(t) = h(x, u) \tag{3.115}$$

where g and h can be nonlinear functions.* The behavior of nonlinear systems is beyond the scope of this section; however, a detailed discussion can be found in [10]. The behavior of a nonlinear system can be approximated by a linear model in a neighborhood of an equilibrium point. Such linearizations can simplify the analysis and design of nonlinear systems because the tools developed for linear systems can be applied under certain conditions [10]. Let x_0 and u_0 be the equilibrium point and equilibrium input, respectively, such that [10, Chapter 1]

$$g(x_0, u_0) = 0 \tag{3.116}$$

$$h(x_0, u_0) = y_0 \tag{3.117}$$

Consider small perturbations in the equilibrium point $x(t) = x_0 + \bar{x}(t)$, the input $u(t) = u_0(t) + \bar{u}(t)$, and the output $y(t) = y_0 + \bar{y}(t)$. If the perturbation $\bar{x}(t)$ is small for all t, we obtain the following by expanding Equation 3.114 in Taylor series (neglecting higher order terms of $\bar{x}(t)$ and $\bar{u}(t)$):

$$\begin{aligned} x_0 + \dot{\bar{x}}(t) &= g(x_0 + \bar{x}(t), u_0 + \bar{u}(t)) \\ \dot{\bar{x}}(t) &= g(x_0, u_0) + \left.\frac{\partial g}{\partial x}\right|_{\substack{x=x_0 \\ u=u_0}} \bar{x}(t) + \left.\frac{\partial g}{\partial u}\right|_{\substack{x=x_0 \\ u=u_0}} \bar{u}(t) \end{aligned} \tag{3.118}$$

Recognizing that $g(x_0, x_0) = 0$ we obtain

$$\dot{\bar{x}}(t) = \bar{A}\bar{x}(t) + \bar{B}\bar{u}(t) \tag{3.119}$$

where

$$\bar{A} = \left.\frac{\partial g}{\partial x}\right|_{\substack{x=x_0 \\ u=u_0}} \quad \text{and} \quad \bar{B} = \left.\frac{\partial g}{\partial u}\right|_{\substack{x=x_0 \\ u=u_0}} \tag{3.120}$$

The matrices $\bar{A}$ and $\bar{B}$ are the Jacobians evaluated at x_0 and u_0. Equation 3.119 is a linear state equation and is valid for small perturbations about x_0 and u_0. A similar result can be obtained for the change $\bar{y}(t)$ in the output from the equilibrium value y_0 as

$$\bar{y}(t) = \bar{C}\bar{x}(t) + \bar{D}\bar{u}(t) \tag{3.121}$$

where

$$\bar{C} = \left.\frac{\partial h}{\partial x}\right|_{\substack{x=x_0 \\ u=u_0}} \quad \text{and} \quad \bar{D} = \left.\frac{\partial h}{\partial u}\right|_{\substack{x=x_0 \\ u=u_0}} \tag{3.122}$$

*The MATLAB command `ode45` can be used to obtain the numeric solution to the general nonlinear state space equation.

3.3.4 Relationship between State Equations and Transfer-Functions

3.3.4.1 State-Space to Transfer-Function

The input-to-output relationship of a dynamic system in the frequency-domain is represented by a transfer-function, which can be obtained by taking the Laplace transform of (3.107) and (3.108) with zero initial conditions as follows [8, Chapter 3, Section 5]:

$$sX(s) = AX(s) + BU(s), \tag{3.123}$$

$$Y(s) = CX(s) + DU(s), \tag{3.124}$$

where s is the Laplace variable. Solving (3.123) for $X(s)$ and substituting into (3.124), the ratio of the output $Y(s)$ to input $U(s)$ for a single-input single-output system (SISO) can be found as

$$\begin{aligned} G(s) &= \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D \\ &= \frac{N(s)}{D(s)} \end{aligned} \tag{3.125}$$

where I is an $n \times n$ identity matrix. In Equation 3.125, $N(s)$ and $D(s)$ are referred to as the numerator and denominator polynomial of $G(s)$, respectively.*

Analogous to the state-space equation, the boundedness of the output response $y(t)$ to a bounded input $u(t)$ is characterized by the roots of the denominator polynomial $D(s)$, that is, the values of s for which $D(s) = 0$. In particular, the output $y(t)$ will be bounded for any bounded input, that is, system is stable if the real parts of all the roots of $D(s)$ are less than zero (strictly negative).† Alternatively, a convenient method to determine stability without having to find the roots of $D(s)$ explicitly is the Routh–Hurwitz stability criterion [6, Chapter 6].

Example

With the state-space description of the mass-spring-damper system defined in Equations 3.112 and 3.113, the transfer-function realization using Equation 3.125 becomes

$$\begin{aligned} G(s) &= \frac{Y(s)}{U(s)} = \begin{bmatrix} 1 & 0 \end{bmatrix} \left[s \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -(k/m) & -(c/m) \end{bmatrix} \right]^{-1} \begin{bmatrix} 0 \\ b/m \end{bmatrix} + [0] \\ &= \frac{b/m}{s^2 + (c/m)s + k/m} \end{aligned} \tag{3.126}$$

The input to the system is the applied voltage $V_z(t)$ and the output is the displacement of the mass $z(t)$.

3.3.4.2 Frequency-Response Using Transfer-Functions

Consider a linear single-input single-output (SISO) stable system with transfer-function description $G(s)$. When the system $G(s)$ is excited by a sinusoidal input of the form

$$u(t) = P\sin(\omega t) \tag{3.127}$$

with amplitude P and frequency ω, the output response (after the transients decay) will also be a sinusoid of the form

$$y(t) = MP\sin(\omega t + \phi) \tag{3.128}$$

*The MATLAB command `ss2tf` can be used to convert a state-space realization to a transfer-function.

†The MATLAB command `roots (den)` can be used to find the roots of `den`, where `den` is the coefficients of $D(s)$.

with the same frequency ω and a phase shift ϕ [6, Chapter 8]. The output amplitude is the input amplitude scaled by M, the magnitude gain. The magnitude gain M is found by taking the magnitude of $G(s)$ evaluated at $s = j\omega$, that is,

$$M = |G(s)|_{s=j\omega} \tag{3.129}$$

Usually, the magnitude gain M is expressed in units of decibels (dB), i.e., M [dB] = 20log M. The phase shift ϕ is the angle of $G(s)$ evaluated at $s = j\omega$, that is,

$$\phi = \angle G(s)|_{s=j\omega} \tag{3.130}$$

with units of degrees. The plot of the magnitude gain M and the phase shift ϕ versus the frequency ω gives a graphical representation of the frequency-response (Bode plots) of a system.* These plots can be obtained experimentally by measuring the magnitude gain and phase shift between the input and output response of a system over a range of frequencies. Additionally, a system's transfer-function can be obtained from an experimental frequency-response data by using curve-fitting software. In Section 3.3.5, we present this approach to determine a model for a system using experimental frequency-response data.

3.3.4.3 Transfer-Function to State-Space

In Section 3.3.4, a transfer-function model was obtained for a system in state-space form. In the following, an approach for realizing a state-space model from a transfer-function $G(s)$ is presented. For a realizable transfer function $G(s)$ of a SISO system of the form

$$G(s) = \frac{b_0 s^n + b_1 s^{n-1} + \cdots + b_n}{s^n + a_1 s^{n-1} + \cdots + a_n} \tag{3.131}$$

the controllable canonical state-space form written in terms of the coefficients of $G(s)$ is

$$x(t) = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u(t) \tag{3.132}$$

$$y(t) = [(b_1 - a_1 b_0) \quad (b_2 - a_2 b_0) \quad \cdots \quad (b_n - a_n b_0)]x(t) + [b_0]u(t) \tag{3.133}$$

where the number of states n is equal to the highest power of the denominator of $G(s)$.† The smallest possible dimension for realizing a system, referred to as the minimum realization, is an important factor to consider in analysis and design.‡ Models of minimum order require less computational power in simulation and implementation compared to higher order models. For information about other equivalent canonical state-space forms, refer to [7, Chapter 4, Sections 3 and 4].

*The MATLAB command `bode` plots the magnitude gain and phase shift for a linear system.

†The MATLAB command `tf2ss` generates the controllable canonical form realization of a transfer function $G(s)$. Other useful commands include `ss2tf`, `zp2tf`, and `tf2zp`.

‡For a detailed discussion of minimal realizations for multi-input multi-output systems, see [7, Chapter 7].

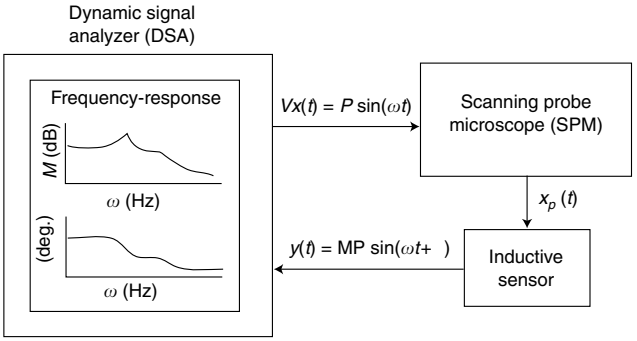


FIGURE 3.22 A schematic of the experimental setup used to determine the frequency-response of the piezo-tube actuator. An inductive sensor measured the displacement of the actuator along the *x*-axis, and the frequency-response data from the DSA were used to estimate the system model.

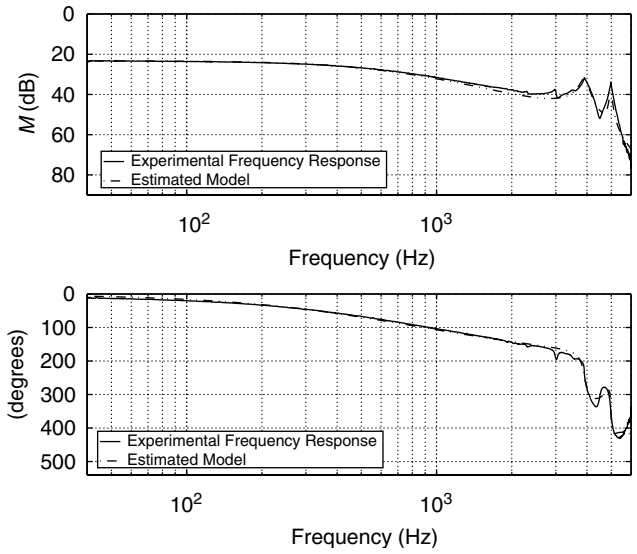


FIGURE 3.23 The experimental magnitude gain and phase versus frequency plots for the piezo-tube actuator measured along the *x*-axis with superimposed model frequency-response. Solid line represents experimental data; dashed line represents results from estimated model.

3.3.5 Experimental Modeling Using Frequency-Response

An approach to modeling using experimental frequency-response data is presented in this section. Using a dynamic signal analyzer (DSA), the frequency-response of the dynamics along the *x*-axis for the piezo-tube actuator was measured.* A sinusoidal input voltage $V_x(t)$ with frequency varying between 10 Hz and 6 kHz was generated by a DSA and applied to the scanning probe microscope (SPM) system as shown in Figure 3.22. Using an inductive sensor, the displacement $x_p(t)$ of the piezo-tube along the *x*-axis was measured and fed back to the DSA to compute the frequency-response (M and ϕ versus frequency ω plots). Figure 3.23 shows the Bode plots obtained by the DSA between the applied voltage $V_x(t)$ and the output of the inductive sensor $y(t)$. An estimate of the system model from the frequency-response

*Stanford Research Systems, model SRS785.

data was then found with the MATLAB software.* The transfer-function between the applied input voltage $V_x(t)$ and the output of the inductive sensor $y(t)$ was found to be

$$G_1(s) = \frac{Y(s)}{V_x(s)} = \frac{5.544 \times 10^5 s^4 - 7.528 \times 10^9 s^3 + 1.476 \times 10^{15} s^2 - 4.571 \times 10^{18} s + 9.415 \times 10^{23}}{s^6 + 1.255 \times 10^4 s^5 + 1.632 \times 10^9 s^4 + 1.855 \times 10^{13} s^3 + 6.5 \times 10^{17} s^2 + 6.25 \times 10^{21} s + 1.378 \times 10^{25}} \tag{3.134}$$

with units of V/V. Equation 3.135 was scaled by the inductive sensor gain (30 Å/V) and the transfer-function between the applied voltage $V_x(t)$ and the actual displacement of the piezo-tube $x_p(t)$ is given by

$$G_2(s) = \frac{X_p(s)}{V_x(s)} = \frac{1.663 \times 10^7 s^4 - 2.258 \times 10^{11} s^3 + 4.427 \times 10^{16} s^2 - 1.371 \times 10^{20} s + 2.825 \times 10^{25}}{s^6 + 1.255 \times 10^4 s^5 + 1.632 \times 10^9 s^4 + 1.855 \times 10^{13} s^3 + 6.5 \times 10^{17} s^2 + 6.25 \times 10^{21} s + 1.378 \times 10^{25}} \tag{3.135}$$

with units of Å/V.

3.3.5.1 Time Scaling of a Transfer-Function Model

We present below an approach for rescaling time for $G_2(s)$ from seconds [s] to milliseconds [ms]. We briefly recall the time scaling property of the Laplace transform presented in [1, Chapter 3, Section 1.4]. Let $F(s)$ be the Laplace transform of $f(t)$, that is,

$$f(t) \xrightarrow{L} F(s) \tag{3.136}$$

where L denotes the Laplace transform operator. Now, consider a new time scale defined as $\hat{t} = at$, where a is a constant. The Laplace transform of $f(\hat{t}) = f(at)$ is given by

$$f(\hat{t}) = f(at) \xrightarrow{L} \frac{1}{|a|} F\left(\frac{s}{a}\right) = \hat{F}(s) \tag{3.137}$$

Using relation (3.137), we can reduce the coefficients of $G_2(s)$ by changing the time units of both the input signal $u(t)$ and output signal $y(t)$ as follows:

$$\hat{G}(s) = \frac{\hat{Y}(s)}{\hat{U}(s)} = \frac{Y(s/a)/|a|}{U(s/a)/|a|} = \frac{Y(s/a)}{U(s/a)} = G\left(\frac{s}{a}\right) \tag{3.138}$$

Therefore, to rescale time for $G_2(s)$ from seconds [s] to millisecond [ms], we choose $\hat{t} = at = 0.001t$ and the new rescaled transfer $G_2(s)$ becomes

$$\hat{G}_2(s) = G_2\left(\frac{s}{a}\right)\Big|_{a=0.001} = G_2(1000s)$$

$$\hat{G}_2(s) = \frac{16.63 s^4 - 225.8 s^3 + 4.427 \times 10^4 s^2 - 1.371 \times 10^5 s + 2.825 \times 10^7}{s^6 + 12.55 s^5 + 1.632 \times 10^3 s^4 + 1.855 \times 10^4 s^3 + 6.5 \times 10^5 s^2 + 6.25 \times 10^6 s + 1.378 \times 10^7} \tag{3.139}$$

*The MATLAB command `invfreqs` gives real numerator and denominator coefficients of experimentally determined frequency response data.

Note that the time unit of the input and output signal of $\hat{G}_2(s)$ are now in milliseconds, not seconds! The coefficients of the numerator and denominator polynomials are smaller and this form ($\hat{G}_2(s)$) is less prone to computational errors due to round off than the form $G_2(s)$, Equation 3.135.

3.3.5.2 The State-Space Model

The state-space realization for $\hat{G}_2(s)$ expressed in controllable canonical form (Equations 3.132 and 3.133) is given by the following:

$$\dot{x}(t) = \begin{bmatrix} -12.55 & -1.632\times10^{3} & -1.855\times10^{4} & -6.50\times10^{5} & -6.25\times10^{6} & -1.378\times10^{7} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(t) \tag{3.140}$$

$$y(t) = [0 \;\; 16.63 \;\; -225.8 \;\; 4.427\times10^{4} \;\; -1.371\times10^{5} \;\; 2.825\times10^{7}]\,x(t) \tag{3.141}$$

The time unit for Equations 3.140 and 3.141 are milliseconds [ms]. If the initial state at t_0 is known, along with the applied voltage $V_x(t)$ defined for $t \geq t_0$, the future behavior of the system, that is, the state $x(t)$ and output $y(t)$, can be determined from Equations 3.140 and 3.141, respectively.

3.3.6 Discrete-Time State-Space Modeling

3.3.6.1 Introduction

The study of discrete-time systems is important to the analysis and the design of modern mechatronics systems where digital computers or small microprocessors are predominantly used to control systems. Digital computers and microprocessors output or acquire information at discrete time instants. For example, the input applied by a digital computer to actuate the piezo-tube changes at discrete time instants. Similarly, the displacment of the piezo-tube can only be measured at specified time instants using digital computers; therefore, in comparison to a continuous-time control system where the input signals change continuously over time, the input of a discrete-time system changes once in a while. Such discrete-time systems are studied next.

Consider a continuous-time system with continuous input $u(t)$ and output $y(t)$ as described by Equations 3.107 and 3.108. Let a digital computer or microprocessor be used to provide the input $u[k]$ and measure the output $y[k]$ as depicted in Figure 3.24 (such systems with continuous and discrete signals are

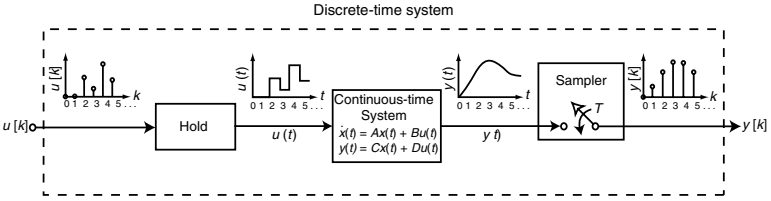


FIGURE 3.24 A block diagram of a discrete-time system showing signals in graphic form. Note that $u[k] = u(k \cdot T)$ and $y[k] = y(k \cdot T)$, for $k = 0,1,2,\ldots$, and the sampling period T is assumed to be constant.

called sampled-data systems). The input $u[k]$ and output $y[k]$ of this system are discrete with $u[k] = u(k \cdot T)$ and $y[k] = y(k \cdot T)$ for $k = 0,1,2,\ldots$, where T is the constant sampling period. The discrete input $u[k]$ is applied to the continuous system from a digital computer or microprocessor and is held constant during the time interval T (zero-order hold). A sampler acquires the output of the continuous system at each time instant T yielding the discrete output $y[k]$. The discrete system is between the input $u[k]$ and the output $y[k]$ [11, Chapter 1].* The equivalent discrete-time state-space representation of the continuous-time state-space model given by Equations 3.107 and 3.108 is given by (the details of the formulation can be found in [11, Chapter 5, Section 5])

$$x[k+1] = A_D x[k] + B_D u[k] \tag{3.142}$$

$$y[k] = C_D x[k] + D_D u[k] \tag{3.143}$$

where

$$A_D = e^{AT}, \quad B_D = \left(\int_0^T e^{A\lambda} d\lambda\right) B, \quad C_D = C, \text{ and } D_D = D \tag{3.144}$$

and matrices C_D and D_D are not changed by the sampling.† This discrete model (Equations 3.142 and 3.143) is the representation of the sampled-data system shown in Figure 3.24.

3.3.6.2 Solutions to the Discrete-Time State-Space Equations

The solution to the discrete model (Equations 3.142 and 3.143) is given by

$$x[k] = A_D^k x[0] + \sum_{j=0}^{k-1} A_D^{k-j-1} B_D u[j] \tag{3.145}$$

$$y[k] = CA_D^k x[0] + C\sum_{j=0}^{k-1} A_D^{k-j-1} B_D u[j] + Du[k] \tag{3.146}$$

for each sampling step k. Details of the formulation can be found in [11, Chapter 5, Section 3]. The state response $x[k]$ to an applied input $u[k]$ is characterized by the system matrices (A_D, B_D, C_D, D_D). In particular, the output $y[k]$ will be bounded for any bounded input $u[k]$ if the system is stable. A system in the form given by Equation 3.142 is stable if the magnitude of all the eigenvalues of A_D are less than unity, that is, lie within the unit circle center at the origin of the z-plane [11, Chapter 5, Section 6].

3.3.6.3 The z Transform and Relationship with the State-Space

The input-to-output relationship in the frequency-domain for a discrete-time system is represented by a discrete transfer-function called the z transform, written in terms of the variable z [12, Chapter 4]. Analogous to the continuous-time case, the model of a dynamic system in discrete transfer-function form can be useful in the design and control of systems [12, Chapter 7]. If the system model is available in discrete transfer-function form, then a state-space realization can be found as follows. Given a discrete system described by the following z-transform $G(z)$:

$$G(z) = \frac{d_0 + d_1 z^{-1} + \cdots + d_n z^{-n}}{1 + c_1 z^{-1} + \cdots + c_n z^{-n}} \tag{3.147}$$

*We do not discuss quantizing and quantization error. See [11, Chapter 1, Section 3] for details.

†Given a continuous-time state-space model (A, B, C, D), the MATLAB command `c2d`, gives the discrete time equivalent for a specified sampling period T.

the controllable canonical discrete state-space realization for $G(z)$ is

$$x[k+1] = \begin{bmatrix} -c_1 & -c_2 & \cdots & -c_{n-1} & -c_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} x[k] + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u[t] \tag{3.148}$$

$$y[k] = [(d_1 - c_1 d_0) \quad (d_2 - c_2 d_0) \quad \ldots \quad (d_n - c_n d_0)]x[k] + [d_0]u[k] \tag{3.149}$$

The number of states n is equivalent to the highest power of the denominator of $G(z)$. For information about other equivalent canonical state-space forms, refer to [11, Chapter 5, Section 2].

Example

Consider the continuous-time state-space model of the piezo-tube system described by Equations 3.140 and 3.141. A digital computer with the sampling rate of 10 kHz ($T = 1.0 \times 10^{-4}$) is used to provide the control input $u[k]$ and measure its displacement along the x-axis (output $y[k]$). The discrete-time state-space model with (A_D, B_D, C_D, and D_D) given by Equation 3.144 is

$$x[k+1] = \begin{bmatrix} 0.999 & -0.163 & -1.85 & -65.0 & -624.5 & -1377.1 \\ 9.99 \times 10^{-5} & 0.999 & -9.26 \times 10^{-5} & -3.25 \times 10^{-3} & -3.12 \times 10^{-2} & -6.69 \times 10^{-2} \\ 5.00 \times 10^{-9} & 1.00 \times 10^{-4} & 1 & -1.08 \times 10^{-7} & -1.04 \times 10^{-6} & -2.30 \times 10^{-6} \\ 1.67 \times 10^{-13} & 5.00 \times 10^{-9} & 1.00 \times 10^{-4} & 1 & -2.60 \times 10^{-11} & -5.74 \times 10^{-11} \\ 4.17 \times 10^{-18} & 1.67 \times 10^{-13} & 5.00 \times 10^{-9} & 1.00 \times 10^{-4} & 1 & -1.15 \times 10^{-15} \\ 8.33 \times 10^{-23} & 4.17 \times 10^{-18} & 1.67 \times 10^{-13} & 5.00 \times 10^{-9} & 1.00 \times 10^{-4} & 1 \end{bmatrix} x[k]$$

$$+ \begin{bmatrix} 9.99 \times 10^{-5} \\ 4.99 \times 10^{-9} \\ 1.67 \times 10^{-13} \\ 4.17 \times 10^{-18} \\ 8.33 \times 10^{-23} \\ 1.39 \times 10^{-27} \end{bmatrix} u[k] \tag{3.150}$$

$$y[k] = \begin{bmatrix} 0 & 16.63 & -225.8 & 4.427 \times 10^4 & -1.371 \times 10^5 & 2.825 \times 10^7 \end{bmatrix} x[k] \tag{3.151}$$

The realization given by Equations 3.150 and 3.151 was found using the MATLAB command '`c2d`'.

3.3.7 Summary

We presented tools for modeling continuous- and discrete-time systems using the state-space approach in this section. The state-space approach to modeling is a powerful technique for the analysis and design of mechatronic and dynamic systems, and can take advantage of tools available in modern digital computers and microprocessors. The discussion of the system states and the state-space was motivated by an example piezo-tube actuator system. We considered the modeling of linear systems and a technique for linearizing nonlinear systems was briefly introduced. The frequency-response of a system and an approach to modeling using experimental frequency-response data was presented. Relationships between models

expressed in the frequency- and time-domain for both continuous- and discrete-time systems was discussed. For additional details about the concepts mentioned in this section and those not covered, it is recommended that the reader consider the attached references for further reading.

References

1. Franklin, G. F., et al., *Feedback Control of Dynamic Systems,* 3rd ed., Addison-Wesley, New York, 1994.
2. Hanselman, D., and Littlefield, B., *The Student Edition of Matlab, Version 5, User's Guide,* Prentice-Hall, Upper Saddle River, NJ, 1997.
3. Croft, D., et al., Creep, hysteresis, and vibration compensation for piezoactuators: atomic force microscopy application, *ASME J. Dyn. Syst., Meas., Control,* 123, 35, 2001.
4. Dorny, C. N., *Understanding Dynamic Systems-Approaches to Modeling, Analysis, and Design,* Prentice-Hall, Englewood Cliffs, NJ, 1993.
5. Locatelli, M., et al., Easy method to characterize a piezoelectric ceramic tube as a displacer, *Rev. Sci. Instrum.,* 59, 4, 1988.
6. Dorf, R. C., and Bishop, R. H., *Modern Control Systems,* 9th ed., Prentice-Hall, Upper Saddle River, NJ, 2001.
7. Chen, T. C., *Linear System Theory and Design,* Oxford University Press, New York, 1999.
8. Friedland, B., *Control System Design: An Introduction to State-Space Methods,* McGraw-Hill, New York, 1986.
9. Gillis, J. T., State space, in *The Control Handbook,* Levine, W. S., CRC Press, Salem, MA, 1996, Chap. 5.
10. Khalil, H. K., *Nonlinear Systems,* 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 1996.
11. Ogata, K., *Discrete-Time Control Systems,* 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1995.
12. Franklin, G. F., et al., *Digital Control of Dynamic Systems,* 3rd ed., Addison-Wesley, Menlo Park, 1998.

3.4 Transfer Functions and Laplace Transforms

C. Nelson Dorny

We perceive a system primarily through its behavior. Therefore, our mental image of a system usually includes representative response **signals**. The *step response*, the behavior when we suddenly turn on the system, is such a system-characterizing signal. We should view the step response as a description of the system. The *impulse response* is another description of the system. For a system represented by linear differential equations, the unit-step response is the integral of the unit-impulse response.

Let us represent time differentiation (d/dt) by the *time-derivative operator, p*. Then we can denote the time derivative of a signal y by py, its second derivative by p^2y, its integral with respect to time by $(1/p)y$, and so on. This *operator notation* simplifies the expressions for differential equations. We shall use the expression *system equations* to mean a set of differential equations that determines fully the behaviors of the dependent variables that appear in those equations. We can reduce a *linear* set of system equations to a single **input-output system equation** by eliminating all but one dependent variable from the set. The *transfer function* associated with that dependent variable is a mathematical expression that contains all the essential information embodied in the system differential equation.

The Laplace transformation converts signals (functions of time) to functions of a *complex-frequency variable,* $s = \sigma + j\omega$. There is a one-to-one correspondence between a signal and its Laplace transform. We can retrieve the time function by inverse transformation. Laplace transformation produces images that have some properties that are more convenient than those of the original signals. In particular, time differentiating a signal corresponds to multiplying its Laplace transform by the complex-frequency variable s. Hence, the transformation converts linear constant-coefficient differential equations to linear algebraic equations. Such simplifications of time-domain operations make Laplace transformation useful.

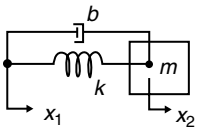


FIGURE 3.25 The lumped model of a mechanical system.

The Laplace transformation also converts the impulse response of a system variable to the transfer function for that variable. As a consequence, we can view the differential equation that represents a linear system as an expression of the response of that system to an impulsive input.

3.4.1 Transfer Functions

The node displacements x_1 and x_2 and the compressive forces f_1 and f_2 within the branches of the lumped model of Figure 3.25 are related to each other by the spring equation, the damper equation, and the balance of forces at node 2. The spring equation is $f_2 = k(x_1 - x_2)$. The equation for the damper is $f_1 = b(px_1 - px_2)$. The balance of forces requires that $f_1 + f_2 = mp^2x_2$. These equations describe fully the behavior of the system if the spring and mass are unenergized. (If the mass were moving and/or the spring were compressed, we would have to express separately their initial energy states to describe fully the future relations among the variables.)

Eliminate f_1 and f_2 from the equations to obtain the operational equation:

$$(mp^2 + bp + k)x_2 = (bp + k)x_1 \tag{3.152}$$

This differential equation describes fully the **zero-state** relation between x_1 and x_2. Rearrange Equation 3.152 to form the ratio

$$\frac{x_2}{x_1} = \frac{bp + k}{mp^2 + bp + k} \tag{3.153}$$

We call Equation 3.152 the *transfer function* from x_1 to x_2. The transfer function focuses attention on the mathematical operations that characterize the behavioral relationships rather than on the particular natures of the variables. (Note that the transfer function from v_1 to v_2, where $v_1 = px_1$ and $v_2 = px_2$, is the same as the transfer function given by Equation 3.153.)

In general, suppose that y_1 and y_2 are two variables related (in operator notation) by the linear differential equation

$$y_2 = G(p)y_1 \tag{3.154}$$

We formally define the *transfer function* from y_1 to y_2 by

$$G(p) = \left.\frac{y_2}{y_1}\right|_{\text{ZS}} \tag{3.155}$$

where the notation ZS means **zero state**. If y_1 is an independent variable, then $G(p)$ is the *input-output transfer function* for the variable y_2 and accounts fully for its behavior owing to the **input signal** y_1. We can determine from that transfer function the behavior of the system for any source waveform and any initial state.

3.4.2 The Laplace Transformation

The *one-sided Laplace transformation*, $\mathcal{L}$, is an integral operator that converts a signal $f(t)$ to a complex-valued function $F(s)$ in the following fashion:

$$+[f(t)] \equiv F(s) \triangleq \int_{0^-}^{\infty} f(t)e^{-st}\,dt \tag{3.156}$$

We refer to the transformed function $F(s)$ as the *Laplace transform* of the signal $f(t)$. Picture the lower limit 0^- of the integral as a *specific* instant prior to but infinitesimally close to $t = 0$. It is customary to use a lowercase symbol (f) to represent a signal waveform and an uppercase symbol (F) to represent its Laplace transform. (Although we speak here of time signals, there is nothing in Equation 3.156 that requires $f(t)$ to be a function of time. The transformation can be applied to functions of any quantity t.)

We shall use the Laplace transformation to transform the signals of **time-invariant** linear systems. The behavior of such a system for $t \geq 0$ depends only on the input signal for $t \geq 0$ and on the prior **state** of the output variable (at $t = 0^-$). Hence, it does not matter that the Laplace transformation ignores $f(t)$ for $t < 0^-$.

The process of finding the time function $f(t)$ that corresponds to a particular Laplace transform $F(s)$ is called *inverse Laplace transformation*, and is denoted by $\mathcal{L}^{-1}$. We also call $f(t)$ the *inverse Laplace transform* of $F(s)$. Since the one-sided Laplace transformation ignores $t < 0^-$, $F(s)$ contains no information about $f(t)$ for $t < 0^-$. Therefore, inverse Laplace transformation cannot reconstruct $f(t)$ for $t < 0^-$. We shall treat all signals as if they are defined only for $t \geq 0^-$. Then there is a one-to-one relation between $f(t)$ and $F(s)$.

To illustrate the Laplace transformation, we find the Laplace transform of the decaying exponential, $f(t) = e^{-\alpha t}$, $t \geq 0^-$. The transform is

$$\begin{aligned} F(s) &= \int_{0^-}^{\infty} e^{-\alpha t}e^{-st}\,dt = \left.\frac{e^{-(s+\alpha)t}}{-(s+\alpha)}\right|_{0^-}^{\infty} \\ &= \left.\frac{e^{-(\sigma+\alpha)t}e^{-j\omega t}}{-(s+\alpha)}\right|_{0^-}^{\infty} = \frac{1}{s+\alpha} \quad \text{for } \mathrm{Re}[s] > -\alpha \end{aligned} \tag{3.157}$$

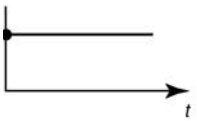
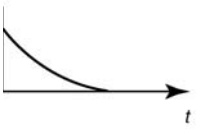
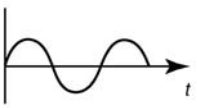
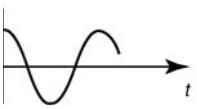
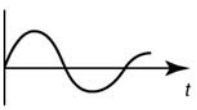
We must require $\sigma > -\alpha$, where σ is the real part of s, in order that the real-exponent factor converge to zero at the upper limit. (The magnitude of the complex-exponent factor remains 1 for all t.) Therefore, the Laplace transform of the decaying exponential is defined only for $\mathrm{Re}[s] > -\alpha$. This restriction on the domain of F in the complex s plane is comparable to the restriction $t \geq 0^-$ on the domain of f.

The significant features of the complex-frequency function $1/(s + \alpha)$ are the existence of a single pole and the location of that pole, $s = -\alpha$ [rad/s]. (The pole defines the left boundary of that region of the complex s plane over which the transform $1/(s + \alpha)$ is defined.) The significant features of the corresponding time function are the fact of decay and the rate of decay, with the exponent $-\alpha$ [rad/s]. There are clear parallels between the features of $f(t)$ and $F(s)$. We should think of the whole complex-valued function F as representing the whole time waveform f.

As a second transformation example, let $f(t) = \delta(t)$, the unit impulse, essentially a unit-area pulse of very short duration. It acts at $t = 0$, barely within the lower limit of the Laplace integral. It has value zero at $t = 0^-$. (Because we use 0^- as the lower limit of the defining integral, it does not matter whether the impulse straddles $t = 0$ or begins to rise at $t = 0$.) The impulse is nonzero only for $t \approx 0$, where $e^{-st} \approx 1$. Therefore, the Laplace transform is

$$\Delta(s) = \int_{0^-}^{\infty} \delta(t)e^{-st}\,dt \approx \int_{0^-}^{\infty} \delta(t)(1)\,dt = 1 \tag{3.158}$$

Table 3.11 Laplace Transform Pairs

$f(t) = \mathcal{L}^{-1}[\mathbf{F}(s)], t \geq 0^-$		$\mathbf{F}(s) = \mathcal{L}[f(t)]$
1. Unit impulse $\delta(t)$		1
2. Unit step $u_s(t)$		$\frac{1}{s}$
3. t^n, $n = 1, 2, \ldots$		$\frac{n!}{s^{n+1}}$
4. $e^{-\alpha t}$		$\frac{1}{s+\alpha}$
5. $t^n e^{-\alpha t}$, $n = 1, 2, \ldots$		$\frac{n!}{(s+\alpha)^{n+1}}$
6. $\sin(\omega_0 t)$		$\frac{\omega_0}{s^2+\omega_0^2}$
7. $\cos(\omega_0 t)$		$\frac{s}{s^2+\omega_0^2}$
8. $e^{-\alpha t}\sin(\omega_d t)$		$\frac{\omega_d}{(s+\alpha)^2+\omega_d^2}$
9. $e^{-\alpha t}\cos(\omega_d t)$		$\frac{s+\alpha}{(s+\alpha)^2+\omega_d^2}$

Source: Dorny, C. N. 1993. *Understanding Dynamic Systems*, p. 412. Prentice-Hall, Englewood Cliffs, NJ. With permission.

It is not necessary to derive the Laplace transform for each signal that we use in the study of systems. Table 3.11 gives the transforms for some signal waveforms that are common in dynamic systems.

3.4.3 Transform Properties

A number of useful properties of the Laplace transformation $\mathcal{L}$ are summarized in Table 3.12. According to the derivative property, the multiplier s acts precisely like the time-derivative operator, but in the domain of Laplace-transformed signals. When we Laplace transform the equation for an energy-storage element such as a mass or a spring, the derivative property automatically incorporates the prior energy

Table 3.12 Properties of the Laplace Transformation, ℒ

1. Magnification	$+[af(t)] = a\mathbf{F}(s)$
2. Addition	$+[f_1(t) + f_2(t)] = \mathbf{F}_1(s) + \mathbf{F}_2(s)$
3. Derivative	$+[\dot{f}(t)] = s\mathbf{F}(s) - f(0^-)$
4. Derivatives	$+[\ddot{f}(t)] = s^2\mathbf{F}(s) - sf(0^-) - \dot{f}(0^-)$
5. Integral	$+\left[\int_{0^-}^{t} f(t)\,dt\right] = \dfrac{\mathbf{F}(s)}{s}$
6. Convolution	$+\left[\int_{0^-}^{t} f_1(\lambda) f_2(t-\lambda)\,d\lambda\right] = \mathbf{F}_1(s)\mathbf{F}_2(s)$
7. Initial value	$f(0^+) = \lim_{t\to 0^+} f(t) = \lim_{s\to\infty} s\mathbf{F}(s)$
8. Final value	$f(\infty) = \lim_{t\to\infty} f(t) = \lim_{s\to 0} s\mathbf{F}(s)$ if finite
9. Definite integral	$\int_0^\infty f(t)\,dt = \lim_{s\to 0} s\mathbf{F}(s)$ if finite
10. Exponential decay	$+[e^{-\alpha t} f(t)] = \mathrm{F}(s+\alpha)$
11. Delay	$+[f(t-t_0)u_s(t-t_0)] = e^{-t_0 s}\mathbf{F}(s)$ for $t_0 \geq 0$
12. Time multiplication	$+[tf(t)] = -\dfrac{d\mathbf{F}(s)}{ds}$
13. Time division	$+\left[\dfrac{f(t)}{t}\right] = \int_s^\infty \mathbf{F}(s)\,ds$
14. Time scaling	$+[f(at)] = \dfrac{\mathbf{F}(s/a)}{a}$

Source: Dorny, C. N. 1993. *Understanding Dynamic Systems*, p. 413. Prentice-Hall, Englewood Cliffs, NJ. With permission.

state of the element—essentially the value of the variable at $t = 0^-$. When we Laplace transform the input–output system equation for a particular system variable, the derivative property automatically incorporates the whole prior system state. As a consequence, we can find the solution to the system equation without having to determine the initial conditions (at $t = 0^+$)—a considerable simplification of the solution process.

Since $F(s)$ contains all information about $f(t)$ for $t \geq 0^-$, it is possible to find some features of the signal $f(t)$ from the transform $F(s)$ without performing an inverse Laplace transformation. Properties 7–9 of Table 3.12 provide three of these features, namely the initial value ($t \to 0^+$), the final value ($t \to \infty$), and the area under the waveform. The remaining properties in the table show the effect on the transform of various changes in the signal waveform.

The usual approach to finding inverse transforms is to use a table of transform pairs. That table might be stored in a software package such as CC, MATLAB, MAPLE, and so on. Table 3.11 demonstrates that transforms of typical system signals are ratios of polynomials in s. A ratio of polynomials can be decomposed into a sum of *simple* polynomial fractions—a process referred to as *partial fraction expansion*. Hence, the inversion process can be accomplished by a computer program that incorporates a brief table of transforms.

3.4.4 Transformation and Solution of a System Equation

Suppose that an independent external source applies a specific velocity pattern $v_1(t)$ to node 1 of Figure 3.25. To obtain the input–output system equation that relates the velocity v_2 of node 2 to the input signal v_1, multiply Equation 3.153 by p and substitute v_1 for px_1 and v_2 for px_2. The result is

$$(mp^2 + bp + k)v_2 = (bp + k)v_1 \tag{3.159}$$

The two sides of Equation 3.159 are identical functions of time. Therefore, the Laplace transforms of the two sides of Equation 3.159 are equal. Since the Laplace transformation is linear (properties 1 and 2 of Table 3.12), and since the coefficients of the differential equation are constants, the Laplace transform can be applied separately to the individual terms of each side. The result is

$$m[s^2 V_2(s) - sv_2(0^-) - \dot{v}_2(0^-)] + b[sV_2(s) - v_2(0^-)] + kV_2(s) = b[sV_1(s) - v_1(0^-)] + kV_1(s) \tag{3.160}$$

where the derivative properties of the Laplace transformation (properties 3 and 4 of Table 3.12) introduce the prior values $v_1(0^-)$, $v_2(0^-)$, and $\dot{v}_2(0^-)$ into the equation. According to Equation 3.160, to fully determine the transform $V_2(s)$ of the behavior $v_2(t)$, we must specify these prior values and also $V_1(s)$. It can be shown that specifying the three prior values is equivalent to specifying the energy states of the spring and mass.

Let us assume that the independent source applies the constant velocity $v_1(t) = v_c$ beginning at $t = 0$. The corresponding transform, by item 2 of Table 3.11 and property 1 of Table 3.12, is $V_1(s) = v_c/s$. Substitute the transform $V_1(s)$ into Equation 3.160 and solve for

$$V_2(s) = \frac{(bs + k)v_c + ms\dot{v}_2(0^-) + bs[v_2(0^-) - v_1(0^-)] + ms^2 v_2(0^-)}{s(ms^2 + bs + k)} \tag{3.161}$$

We could find the output signal waveform $v_2(t)$ as a function of the model parameters m, k, b, the source-signal parameter v_c, and the prior state information $v_1(0^-)$, $v_2(0^-)$, and $\dot{v}_2(0^-)$, but the expression for the solution would be messy. Instead, we complete the solution process for specific numbers: $m =$ 2 kg, $b = 4$ N · s/m, $k = 10$ N/m, $\dot{v}_2(0^-) = 0$ m/s^2, $v_1(0^-) = 0$ m/s, $v_2(0^-) = -1$ m/s, and $v_c = 1$ m/s. The partial-fraction expansion of the transform and the inverse transform, both obtained by a commercial computer program, are

$$V_2(s) = \frac{1}{s} - \frac{2s + 2}{(s + 1)^2 + 2^2} \tag{3.162}$$

$$v_2(t) = 1 - 2e^{-t}\cos(2t), \quad \text{for } t \geq 0 \tag{3.163}$$

We can take Laplace transforms of the system equations at any stage in their development. We can even write the equations directly in terms of transformed variables if we wish. The process of eliminating variables can be carried out as well in one notation as in another. For example, the operator $G(p)$ in Equation 3.154 represents a ratio of polynomials in the time-derivative operator p. Therefore, Laplace transforming the differential equation, Equation 3.154, introduces the prior values of various derivatives of y_1 and y_2. If the prior values of all these derivatives are zero, then the Laplace-transformed equation is

$$Y_2(s) = G(s)Y_1(s) \tag{3.164}$$

where the operator p in Equation 3.154 is replaced by the complex-frequency variable s in Equation 3.164. It is appropriate, therefore, to define the transfer function directly in terms of Laplace-transformed signals:

$$G(s) = \left.\frac{Y_2(s)}{Y_1(s)}\right|_{\text{PV}=0} \tag{3.165}$$

where $Y_1(s)$ and $Y_2(s)$ are the Laplace transforms of the signals $y_1(t)$ and $y_2(t)$, and the notation PV = 0 means that the prior values (at $t = 0^-$) of $y_1(t)$ and $y_2(t)$ and the various derivatives mentioned above in connection with Equation 3.164 are set to zero. The *frequency domain* definition, Equation 3.165, is equivalent to the *time domain* definition, Equation 3.155.

Suppose that the input signal $y_1(t)$ is the unit impulse $\delta(t)$. Then the response signal $y_2(t)$ is the unit-impulse response of the system. Since the Laplace transform of the unit impulse is $Y_1(s) = \Delta(s) = 1$ by entry 1 of Table 3.11, Equation 3.164 shows that the Laplace transform $Y_2(s)$ of the unit-impulse response is identical to the zero-state transfer function (expressed in the transform domain).

The transfer function for a linear system has two interpretations. Both interpretations characterize the system. In the frequency domain, the transfer function $G(s)$ is the multiplier that produces the response—by multiplying the source-signal transform, as in Equation 3.164. In the time domain, we use a representative response signal—the impulse response—to characterize the system. The transfer function $G(s)$ is the Laplace transform of that characteristic response.

Defining Terms

Input: An independent variable.

Input-output system equation : A differential equation that describes the behavior of a single dependent variable as a function of time. The dependent variable is viewed as the system output. The independent variable(s) are the inputs.

Output: A dependent variable.

Signal: An observable variable; a quantity that reveals the behavior of a system.

State: The state of an nth-order linear system corresponds to the values of a dependent variable and its first $n - 1$ time derivatives.

Time invariant: A system that can be represented by differential equations with constant coefficients.

Zero state: A condition in which no energy is stored or in which all variables have the value zero.

References

Dorny, C. N. 1993. *Understanding Dynamic*, p. 413, Prentice-Hall, Englewood Cliffs, NJ.

Franklin, G. F., Powell, J. D., and Emami-Naeini, A. 1994. *Feedback Control of Dynamic Systems,* 3rd ed., Addison Wesley, Reading, MA.

Kuo, B. C. 1991. *Automatic Control Systems,* 6th ed., Prentice-Hall, Englewood Cliffs, NJ.

Nise, N. S. 1992. *Control Systems Engineering,* Benjamin Cumming, Redwood City, CA.

Further Information

A thorough mathematical treatment of Laplace transforms is presented in *Advanced Engineering Mathematics,* by C. Ray Wylie and Louis C. Barrett. *Understanding Dynamic Systems,* by C. Nelson Dorny, applies transfer functions and related concepts in a variety of contexts. The following journals publish papers that use transfer functions and Laplace transforms:

IEEE Transactions on Automatic Control. Published monthly by the Institute of Electrical and Electronics Engineers.

IEEE Transactions on Systems, Man, and Cybernetics. Published bimonthly.

Journal of Dynamic Systems, Measurement, and Control. Published quarterly by the American Society of Mechanical Engineers.

4

State Space Analysis and System Properties

Mario E. Salgado
Juan I. Yuz
Universidad Técnica Federico Santa María

4.1 Models: Fundamental Concepts

An essential connection between an engineer/scientist and a system relies on his/her ability to describe the system in a way which is useful to understand and to quantify its behavior.

Any description supporting that connection is a **model**. In system theory, models play a fundamental role, since they are needed to analyze, to synthesize, and to design systems of all imaginable sorts.

There is not a unique model for a given system. Firstly, the need for a model may obey different purposes. For instance, when dealing with an electric motor, we might be interested in the electro-mechanical energy conversion process, alternatively, we might be interested in modelling the motor either as a thermal system, or as a mechanical system to study vibrations, the strength of the materials, and so on.

A second source of that nonuniqueness is the fact that models are always inaccurate, since real systems are usually infinitely complex. One of the key decisions for an engineer when facing the task of **modelling** a system is to decide which are the essential features that the model should capture, and that decision is also closely related to the purpose of the model.

The theory supporting modelling is by itself a vast field, where first principles, signal theory, mathematics and numerical tools combine in different ways to generate rich methodologies. A model is rarely built in one go, the model building process is usually iterative, and it progresses according to the quality of the results obtained when using the model in a particular application. Iterations may also include changes in modelling methodology.

In this chapter we will deal with a special class of models to describe *dynamic systems*. Dynamic systems are those where the system variables are interdependent not only algebraically, but also in a way where we observe the intervention of accumulated effects and rate of change. Models for dynamic systems can be built in the continuous time domain, in the discrete time domain, or in a continuous-discrete time framework (for hybrid systems, involving sampled systems). We will cover the three situations.

In this quest we will put the emphasis on concepts, fundamental properties, physical interpretations, and examples. We will include neither proofs nor intricate theoretical developments. Sometimes we will sacrifice rigor for the sake of an easier understanding. To cover in depth the theory supporting our presentation we refer the interested reader to the specialized literature such as [6,8,10–14].

4.2 State Variables: Basic Concepts

4.2.1 Introduction

One of the most frequently used class of models is that defined by a set of equations on a set of system inner variables. These inner variables are known as **state variables**. The values they have at a specific time instant form a set known as the **system state**, although we will often use the expressions *state variables* and *system state* as synonyms.

The above definition is too vague since it would fit to any set of system variables. What is distinctive in the set of state variables is clarified in the following definition.

> *A set of state variables for the given system is a set of system inner variables such that any system variable can be computed as a function of the present state and the present and future system inputs.*

In this definition we have preferred to stress the physical meaning of state variables. However, a more abstract definition is also possible. The definition also implies that if we know the state at time t we can then compute the energy stored in the system at that instant. The energy stored in a system depends on some system variables (speed, voltage, current, position, temperature, pressure, etc.) and all of them, by definition, can be computed from the system state.

The above definition suggests that one can think of the state in a more general way: the state variables can be chosen as a **function** (e.g., a linear combination) of inner system variables. This generalization builds some distance between the state and its physical interpretation. However, it has the advantage of making the framework more general. It also makes more evident an interesting feature: the **choice of state variables is not unique**.

Another important observation is that the time evolution of the state, the state trajectory itself, can be computed from the present value of the state and the present and future inputs. Thus, the models involved are first order differential (continuous time) or one-step recursive (discrete time) equations.

4.2.2 Basic State Space Models

If we denote by $\mathbf{x}$ the vector corresponding to a particular choice of state variables, the general form of a state variable model is as follows:

For continuous-time systems

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{4.1}$$

$$\mathbf{y}(t) = \mathbf{G}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{4.2}$$

where $\mathbf{u}(t)$ is the system **input vector** and $\mathbf{y}(t)$ is an **output vector.**

For discrete-time systems

$$\mathbf{x}[t+1] = \mathbf{F}_d(\mathbf{x}[t], \mathbf{u}[t], t) \tag{4.3}$$

$$y[t] = \mathbf{G}_d(\mathbf{x}[t], \mathbf{u}[t], t) \tag{4.4}$$

Similarly to the continuous-time case, $\mathbf{u}[t]$ is the system **input vector** and $\mathbf{y}[t]$ is an **output vector.**

Note that throughout this chapter we will use the symbol t to denote continuous and discrete time, but the difference will be made on using [and] to enclose the argument in the discrete-time case, when $t \in \mathbb{Z}$.

To obtain a first glimpse at the concepts underlying the state space approach, we consider the following example.

Example 4.1

In Figure 4.1, an external force $f(t)$ is applied to a mass-spring system. The position $d(t)$ is measured with respect to the mass position when the spring is relaxed and no external force is applied. The mass movement is damped by a viscous friction force proportional to the mass velocity, $v(t)$.

From first principles we know that to be able to compute the mass position and the mass velocity we must know the initial mass velocity, and the initial spring stretching. Thus, the state vector must have two components, that is, $\mathbf{x}(t) = [x_1(t) \quad x_2(t)]^T$, and a natural state choice is

$$x_1(t) = d(t) \tag{4.5}$$

$$x_2(t) = v(t) = \dot{x}_1(t) \tag{4.6}$$

With this choice, one can apply Newton laws to obtain

$$f(t) = M\frac{dv(t)}{dt} + Kd(t) + Dv(t) = M\dot{x}_2(t) + Kx_1(t) + Dx_2(t) \tag{4.7}$$

where D is the viscous friction proportional constant. We are now in position to write the state equations as

$$\dot{x}_1(t) = x_2(t) \tag{4.8}$$

$$\dot{x}_2(t) = -\frac{K}{M}x_1(t) - \frac{D}{M}x_2(t) + \frac{1}{M}f(t) \tag{4.9}$$

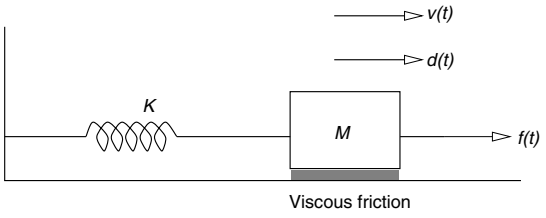


FIGURE 4.1 Mechanical system.

We also observe that the energy, $w(t)$, stored in the system is given by

$$w(t) = \frac{1}{2}K(d(t))^2 + \frac{1}{2}M(v(t))^2 = \mathbf{x}(t)^T \Lambda \mathbf{x}(t) \tag{4.10}$$

where Λ is a diagonal matrix: $\Lambda = \text{diag}\left\{\frac{K}{2}, \frac{M}{2}\right\}$.

Finally, the nonuniqueness of the state vector can be appreciated if, instead of the choices made in 4.8, we choose a new state $\bar{\mathbf{x}}(t)$ related to $\mathbf{x}(t)$ by a nonsingular matrix $\mathbf{T} \in \mathbb{R}^{2\times 2}$, that is,

$$\bar{\mathbf{x}}(t) = \mathbf{T}\mathbf{x}(t) \tag{4.11}$$

More on this will be said in Section 4.3.3.

4.2.3 Signals and State Space Description

The state space framework can also be used to describe a wide variety of signals using a model of the form

$$\frac{d\,\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t), \qquad y(t) = \mathbf{C}\mathbf{x}(t) \quad \text{for continuous-time signals} \tag{4.12}$$

$$\mathbf{x}[t+1] = \mathbf{A}_q\mathbf{x}[t], \qquad y[t] = \mathbf{C}_q\mathbf{x}[t] \quad \text{for discrete-time signals} \tag{4.13}$$

To illustrate the idea we consider a continuous-time signal given by

$$f(t) = 2 + 4\cos(5t) - \sin(5t) \tag{4.14}$$

This signal can be interpreted as the solution for the homogeneous differential equation

$$\frac{d^3 f(t)}{dt^3} + 25\frac{df(t)}{dt} = 0, \quad \text{subject to } f(0) = 6,\ \dot{f}(0) = -5 \text{ and } \ddot{f}(0) = -100 \tag{4.15}$$

If we now choose, as state variables, $x_1(t) = f(t)$, $x_2(t) = \dot{f}(t)$, and $x_3(t) = \ddot{f}(t)$, then the state space model for this signal is

$$\frac{d\mathbf{x}(t)}{dt} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -25 & 0 \end{bmatrix}\mathbf{x}(t), \qquad y(t) = [1 \quad 0 \quad 0]\,\mathbf{x}(t) \tag{4.16}$$

In this usage of state space models, the state variables have no particular physical meaning. However, this description is particularly useful in signal reconstruction theory and when dealing with disturbances in control system synthesis.

4.3 State Space Description for Continuous-Time Systems

In this section the state space description for continuous-time systems is presented. The analysis is focused on the class of **linear and time invariant** systems; to do that, we first show how to build a linear model from the nonlinear Equations 4.1 and 4.2.

An additional restriction is that, at this stage, the systems under study have no pure time delays. This feature generates an infinite dimensional state vector. However, we will see in section 4.4 that this class of systems can be successfully dealt with using sampled data models.

4.3.1 Linearization

Since we will concentrate on time invariant systems, (4.1) and (4.2) can be rewritten as

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.17}$$

$$\mathbf{y}(t) = \mathbf{G}(\mathbf{x}(t), \mathbf{u}(t)) \tag{4.18}$$

We assume that the model (4.17) and (4.18) has at least one *equilibrium point* given by $\{\mathbf{x}_Q, \mathbf{u}_Q, \mathbf{y}_Q\}$. This is a triad conformed by three constant vectors satisfying

$$\mathbf{0} = \mathbf{F}(\mathbf{x}_Q, \mathbf{u}_Q) \tag{4.19}$$

$$\mathbf{y}_Q = \mathbf{G}(\mathbf{x}_Q, \mathbf{u}_Q) \tag{4.20}$$

Note that the equilibrium point is defined by the state derivatives equal to zero.

If we now consider a neighborhood around the equilibrium point, then we can approximate the model (4.17) and (4.18) by a truncated Taylor's series having the form

$$\dot{\mathbf{x}}(t) \approx \mathbf{F}(\mathbf{x}_Q, \mathbf{u}_Q) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}} (\mathbf{x}(t) - \mathbf{x}_Q) + \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}} (\mathbf{u}(t) - \mathbf{u}_Q) \tag{4.21}$$

$$\mathbf{y}(t) \approx \mathbf{G}(\mathbf{x}_Q, \mathbf{u}_Q) + \left.\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}} (\mathbf{x}(t) - \mathbf{x}_Q) + \left.\frac{\partial \mathbf{G}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}} (\mathbf{u}(t) - \mathbf{u}_Q) \tag{4.22}$$

Equations 4.21 and 4.22 can then be written as

$$\frac{d\Delta\mathbf{x}(t)}{dt} = \mathbf{A}\Delta\mathbf{x}(t) + \mathbf{B}\Delta\mathbf{u}(t) \tag{4.23}$$

$$\Delta\mathbf{y}(t) = \mathbf{C}\Delta\mathbf{x}(t) + \mathbf{D}\Delta\mathbf{u}(t) \tag{4.24}$$

where

$$\Delta\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}_Q, \qquad \Delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}_Q, \qquad \Delta\mathbf{y}(t) = \mathbf{y}(t) - \mathbf{y}_Q \tag{4.25}$$

and

$$\mathbf{A} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}}, \qquad \mathbf{B} = \left.\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}}, \qquad \mathbf{C} = \left.\frac{\partial \mathbf{G}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}}, \qquad \mathbf{D} = \left.\frac{\partial \mathbf{G}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\mathbf{x}_Q \\ \mathbf{u}=\mathbf{u}_Q}} \tag{4.26}$$

The linearization ideas presented above are illustrated in the following example.

Example 4.2

Consider the magnetic levitation system shown in Figure 4.2.

The metallic sphere is subject to two forces: its own weight, mg, and the attraction force generated by the electromagnet, $f(t)$. The electromagnet is commanded through a voltage source, $e(t) > 0$, $\forall t$.

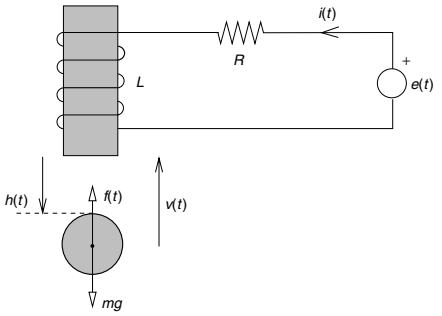


FIGURE 4.2 Magnetic levitation system.

The attraction force on the sphere, $f(t)$, depends on the distance $h(t)$ and the current, $i(t)$. This relation can be approximately described by

$$f(t) = \frac{K_1}{h(t) + K_2} i(t) \tag{4.27}$$

where K_1 and K_2 are positive constants.

Using first principles we can write

$$e(t) = Ri(t) + L\frac{di(t)}{dt} \tag{4.28}$$

$$v(t) = -\frac{dh(t)}{dt} \tag{4.29}$$

$$f(t) = \frac{K_1}{h(t) + K_2} i(t) = mg + m\frac{dv(t)}{dt} \tag{4.30}$$

We next choose as state variables: the current $i(t)$, the sphere position $h(t)$, and the sphere speed $v(t)$, that is,

$$x(t) = [x_1(t) \quad x_2(t) \quad x_3(t)]^T = [i(t) \quad h(t) \quad v(t)]^T \tag{4.31}$$

Then, from 4.28–4.30 we can set the system description as in 4.1 yielding

$$\frac{di(t)}{dt} = \frac{dx_1(t)}{dt} = -\frac{R}{L}x_1(t) + \frac{1}{L}e(t) \tag{4.32}$$

$$\frac{dh(t)}{dt} = \frac{dx_2(t)}{dt} = -x_3(t) \tag{4.33}$$

$$\frac{dv(t)}{dt} = \frac{dx_3(t)}{dt} = \frac{K_1}{m(x_2(t) + K_2)}x_1(t) - g \tag{4.34}$$

Before one can build the linearized model, an equilibrium point has to be computed. The driving input in this system is the source voltage $e(t)$. Say that the equilibrium point is obtained with $e(t) = E_Q$.

Hence, the state in equilibrium can be computed from (4.32) to (4.34), setting all the derivatives equal to zero, that is,

$$-\frac{R}{L}x_{1Q}+\frac{1}{L}E_Q = 0 \Rightarrow x_{1Q} = \frac{E_Q}{R} \tag{4.35}$$

$$-x_{3Q} = 0 \Rightarrow x_{3Q} = 0 \tag{4.36}$$

$$\frac{K_1}{m(x_{2Q}+K_2)}x_{1Q}-g = 0 \Rightarrow x_{2Q} = \frac{K_1}{mg}x_{1Q}-K_2 = \frac{K_1E_Q}{mgR}-K_2 \tag{4.37}$$

The setting now is adequate to build the linearized model in the incremental input $\Delta e(t)$ and the incremental state $\Delta\mathbf{x}(t) = [\Delta x_1(t) \quad \Delta x_2(t) \quad \Delta x_3(t)]^T$. The result is

$$\frac{d\Delta x_1(t)}{dt} = -\frac{R}{L}\Delta x_1(t)+\frac{1}{L}\Delta e(t) \tag{4.38}$$

$$\frac{d\Delta x_2(t)}{dt} = -\Delta x_3(t) \tag{4.39}$$

$$\frac{d\Delta x_3(t)}{dt} = \frac{Rg}{E_Q}\Delta x_1(t)-\frac{Rmg^2}{K_1E_Q}\Delta x_2(t) \tag{4.40}$$

If we define as the system output, the sphere position $h(t)$, we can then compare the above equations with (4.23) and (4.24) to obtain

$$\mathbf{A} = \begin{bmatrix} -\frac{R}{L} & 0 & 0 \\ 0 & 0 & -1 \\ \frac{Rg}{E_Q} & -\frac{Rmg^2}{K_1E_Q} & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \frac{1}{L} \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{D} = 0 \tag{4.41}$$

In the sequel we will drop the prefix Δ, but the reader should bear in mind that the model above is linear in the **incremental** components of the state, the inputs and the outputs around a chosen equilibrium point.

4.3.2 Linear State Space Models

Our starting point is now the linear time invariant state space model

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t)+\mathbf{B}\mathbf{u}(t) \tag{4.42}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t)+\mathbf{D}\mathbf{u}(t) \tag{4.43}$$

The solution to Equation 4.42, subject to $\mathbf{x}(t_o) = \mathbf{x_o}$, is given by

$$\mathbf{x}(t) = e^{\mathbf{A}(t-t_o)}\mathbf{x_o}+\int_{t_o}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\,d\tau \quad \forall t \geq t_o \tag{4.44}$$

where the **transition matrix** $e^{\mathbf{A}t}$ satisfies

$$e^{\mathbf{A}t} = \mathbf{I} + \sum_{k=1}^{\infty} \frac{1}{k!} \mathbf{A}^k t^k \tag{4.45}$$

The interested reader can check that (4.44) satisfies (4.43). To do that he/she should use the Leibnitz's rule for the derivative of an integral.

With the above result, the solution for (4.43) is given by

$$\mathbf{y}(t) = \mathbf{C}e^{\mathbf{A}(t-t_o)}\mathbf{x}_\mathbf{o} + \mathbf{C}\int_{t_o}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\, d\tau + \mathbf{D}\mathbf{u}(t) \tag{4.46}$$

4.3.2.1 System Dynamics

The state of the system has two components: the **unforced** component, $\mathbf{x}_\mathbf{u}(t)$, and the **forced** component, $\mathbf{x}_\mathbf{f}(t)$, where

$$\mathbf{x}_\mathbf{u}(t) = e^{\mathbf{A}(t-t_o)}\mathbf{x}_\mathbf{o} \tag{4.47}$$

$$\mathbf{x}_\mathbf{f}(t) = \int_{t_o}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\, d\tau \tag{4.48}$$

To gain insight into the state space model and its solution, consider the case when $t_o = 0$ and $u(t) = 0$ $\forall t \geq 0$, that is, the state has only the **unforced part**. Then

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_\mathbf{o} \tag{4.49}$$

Further assume that $\mathbf{A} \in \mathbb{R}^n$ and that, for simplicity, it has distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ with n (linearly independent) eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. Then there always exists a set of constants $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that

$$\mathbf{x}_\mathbf{o} = \sum_{\ell=1}^{n} \alpha_\ell \mathbf{v}_\ell, \quad \alpha_\ell \in \mathbb{C} \tag{4.50}$$

A well-known result from linear algebra tells us that the eigenvalues of A^k are $\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k$ with corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$. The application of this result yields

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_\mathbf{o} = \mathbf{I} + \sum_{\ell=1}^{n} \alpha_\ell \sum_{k=1}^{\infty} \frac{1}{k!} \underbrace{\mathbf{A}^k \mathbf{v}_\ell}_{\lambda_\ell^k \mathbf{v}_\ell} t^k = \sum_{\ell=1}^{n} \alpha_\ell e^{\lambda_\ell t}\mathbf{v}_\ell \tag{4.51}$$

This equation shows that the unforced component of the state is a linear combination of **natural modes**, $\{e^{\lambda_\ell t}\}$, each of which is associated with an eigenvalue of **A**. Hence the matrix **A** determines:

- Structure of the unforced response
- Stability (or otherwise) of the system
- Speed of response

When the matrix **A** does not have a set of n independent eigenvectors, Jordan forms can be used (see, e.g., [9,10]).

4.3.2.2 Structure of the Unforced Response

In the absence of input, the state evolves as a combination of natural modes which belong to a defined class of functions: all those generated by exponentials with either real or complex exponents. Hence these modes include constants, real exponentials, pure sine waves, exponentially modulated sine waves, and some other specials functions arising from repeated eigenvalues.

To illustrate these ideas and their physical interpretation consider the system in Example 4.1. For that system

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{D}{M} \end{bmatrix} \tag{4.52}$$

Hence, the system eigenvalues are solutions to the equation

$$\det(\lambda \mathbf{I} - \mathbf{A}) = \lambda^2 + \frac{D}{M}\lambda + \frac{K}{M} = 0 \tag{4.53}$$

that is,

$$\lambda_{1,2} = -\frac{D}{2M} \pm \sqrt{\frac{D^2}{4M^2} - \frac{K}{M}} \tag{4.54}$$

Hence, when the damping is zero ($D = 0$), the system eigenvalues are a couple of conjugate imaginary numbers, and the two natural (complex) modes combine to yield a sustained oscillation with angular frequency $\omega_o = \sqrt{K/M}$. This is in agreement with our physical intuition, since we expect a sustained oscillation to appear when the system has nonzero initial conditions even if the external force, $f(t)$, is zero.

When the system is slightly damped ($D^2 < 4KM$), the matrix eigenvalues are conjugate complex numbers, and the associated complex natural modes combine to yield an exponentially damped sine wave. This also agrees with intuition, since the energy initially stored in the mass and the spring will periodically go from the mass to the spring and vice versa but, at the end, it will completely dissipate, as heat, in the viscous friction.

Finally if the damping is high ($D^2 > 4KM$), the matrix eigenvalues are a couple of negative real numbers, and the natural modes are two decaying exponentials. The heavy damping will preclude oscillations and the initial energy will dissipate quickly.

The three different situation are illustrated in Figure 4.3. For this simulation we have used three different values of the viscous friction constant D and

$$M = 2 \text{ kg}, \quad K = 0.1 \text{ N/m}, \quad d(0) = 0.3 \text{ m}, \quad v(0) = 0.05 \text{ m/s} \tag{4.55}$$

Note that, except when there is no friction ($D = 0$), the mass comes to rest asymptotically.

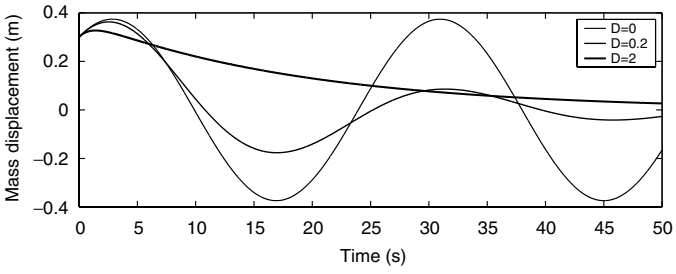


FIGURE 4.3 Unforced response of a mass-spring system.

4.3.2.3 Structure of the Forced Response

When the initial state is zero, the state will exhibit only the forced component. The forced component of the state will include natural modes and some additional **forced** or **particular** modes, which depend on the nature of the system input $\mathbf{u}(t)$. In general the forcing modes in the input will also appear in the state. However, some special cases arise when some of the forcing modes in $\mathbf{u}(t)$ coincide with some system natural modes.

4.3.2.4 System Stability

Stability in linear, time-invariant systems can also be analyzed using the state matrix **A**.

All systems variables can be expressed as linear functions of the state and the system input. When the system input $\mathbf{u}(t)$ is a vector of bounded time functions, then the boundeness of the system variables depends on the state to be bounded.

We then have the following result:

Theorem 4.1 *Consider a system with the state description* (4.42) *and* (4.43) *where* **B**, **C**, *and* **D** *have bounded elements. Then the system state (and hence the system output) is bounded for all bounded inputs if and only if the eigenvalues of* **A** *have negative real parts.*

To illustrate this theorem we again consider the magnetic levitation system from Example 4.2. For that system the matrix **A** (in the linearized model) is given by

$$\mathbf{A} = \begin{bmatrix} -\frac{R}{L} & 0 & 0 \\ 0 & 0 & -1 \\ \frac{Rg}{E_Q} & -\frac{Rmg^2}{K_1 E_Q} & 0 \end{bmatrix} \tag{4.56}$$

and its eigenvalues are the roots of $\det(\lambda\mathbf{I} - \mathbf{A}) = 0$, where

$$\det(\lambda\mathbf{I} - \mathbf{A}) = \left(\lambda + \frac{R}{L}\right)\left(\lambda - \sqrt{\frac{Rmg^2}{K_1 E_Q}}\right)\left(\lambda + \sqrt{\frac{Rmg^2}{K_1 E_Q}}\right) \tag{4.57}$$

One can then see that the set of matrix eigenvalues includes one which is **real and greater than zero**. This implies that the system is **unstable**. This is in agreement with physical reasoning. Indeed, at least theoretically, we can position the sphere in equilibrium (this is described by x_{2Q} in (4.37)). However, this is an unstable equilibrium point, since as soon as we slightly perturb the sphere, it accelerates either towards the ground or towards the magnet.

4.3.2.5 Speed of Response and Resonances

Even if the system is stable there are still some questions regarding other fundamental properties.

To start with, in stable systems the real part of the eigenvalues determines the speed at which the associated mode converges to zero. The slowest modes, the *dominant* modes, determine the speed at which the system output settles at its steady state value, that is, determine the system speed of response. For example, if the system dominant eigenvalues are $\lambda_{1,2} = -\sigma \pm j\omega_o$, $\sigma > 0$, the combined natural modes generate an exponentially damped sine wave $y(t) = Ae^{-\sigma t}\sin(\omega_o t + \alpha)$. We then observe that this signal decays faster for a larger σ.

A second issue, of special importance for flexible structures, is the presence of resonances, which have associated complex eigenvalues. In physical systems, the existence of complex eigenvalues is intimately connected to the presence of two forms of energy. The resonance describes the (poorly damped) oscillation between those two forms of energy. In electric circuits those energies are the electrostatic energy in capacitors and the electromagnetic energy in inductors. In mechanical systems we have the kinetic energy of moving masses and the potential energy in springs. Flexible structures may have many resonant modes. One of the main problems with resonances occurs when the input contains energy at a frequency

close to the resonant frequency. For example, if a system has eigenvalues $\lambda_{1,2} = -0.05 \pm j$, that is, the resonant frequency is 1 rad/s and, additionally, one of the **input** components is a sine wave of frequency 0.9 rad/s, then the system output exhibits a very large (forced) oscillation with amplitude initially growing almost linearly and later, stabilizing to a constant value. In real situations this phenomenon may destroy the system (recall the Tacoma bridge case).

4.3.3 State Similarity Transformation

We have already said that the choice of state variables is nonunique. Say that we have a system with input $\mathbf{u}(t)$, output $\mathbf{y}(t)$, and two different choices of state vectors: $\mathbf{x}(t) \in \mathbb{R}^n$ with an associated 4-tuple ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$), and $\bar{\mathbf{x}}(t) \in \mathbb{R}^n$ with an associated 4-tuple ($\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, $\bar{\mathbf{C}}$, $\bar{\mathbf{D}}$). Then there exists a nonsingular matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$ such that

$$\bar{\mathbf{x}}(t) = \mathbf{T}\mathbf{x}(t) \Leftrightarrow \mathbf{x}(t) = \mathbf{T}^{-1}\bar{\mathbf{x}}(t) \tag{4.58}$$

This leads to the following equivalences:

$$\bar{\mathbf{A}} = \mathbf{T}\mathbf{A}\mathbf{T}^{-1}, \quad \bar{\mathbf{B}} = \mathbf{T}\mathbf{B}, \quad \bar{\mathbf{C}} = \mathbf{C}\mathbf{T}^{-1} \tag{4.59}$$

Different choices of state variables may or may not respond to different phenomenological approaches to the system analysis. Sometimes it is just a question of mathematical simplicity, as we shall see in Section 4.6. In other occasions, the decision is made considering relative facility to measure certain system variables. However, what is important is that, no matter which state description is chosen, certain fundamental system characteristics do not change. They are related to the fact that the system eigenvalues are invariant with respect to similarity transformations, since

$$\det(\lambda\mathbf{I} - \bar{\mathbf{A}}) = \det(\lambda\mathbf{T}\mathbf{T}^{-1} - \mathbf{T}\mathbf{A}\mathbf{T}^{-1}) = \det(\mathbf{T})\det(\lambda\mathbf{I} - \mathbf{A})\det(\mathbf{T}^{-1}) \tag{4.60}$$

$$= \det(\lambda\mathbf{I} - \mathbf{A}) \tag{4.61}$$

Hence, stability, nature of the unforced response, and speed of response are invariants with respect to similarity transformations.

Example 4.3

Consider the electric network shown in Figure 4.4

We choose the state vector $\mathbf{x}(t) = [x_1(t) \quad x_2(t)]^T = [i_L(t) \quad v_c(t)]^T$. Also $u(t) = v_f(t)$. Using first principles we have that

$$\frac{d\mathbf{x}(t)}{dt} = \begin{bmatrix} 0 & \frac{1}{L} \\ -\frac{1}{C} & -\frac{R_1 + R_2}{R_1 R_2 C} \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{R_1 C} \end{bmatrix} u(t) \tag{4.62}$$

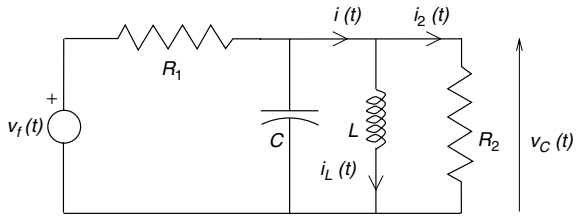


FIGURE 4.4 Electric network.

An alternative state vector is $\bar{\mathbf{x}}(t) = [x_1(t) \quad x_2(t)]^T = [i(t) \quad i_2(t)]^T$. It is straightforward to show that

$$\bar{\mathbf{x}}(t) = \underbrace{\frac{1}{R_2}\begin{bmatrix} R_2 & 1 \\ 0 & 1 \end{bmatrix}}_{\mathbf{T}} \mathbf{x}(t) \tag{4.63}$$

4.3.4 State Space and Transfer Functions

The state space description of linear time invariant systems is an alternative description to that provided by transfer functions. Strictly speaking, the state space description has a wider scope, as we shall see in this subsection.

For a linear time invariant system with input $\mathbf{u}(t) \in \mathbb{R}^m$ and output $\mathbf{y}(t) \in \mathbb{R}^p$, the transfer function, $\mathbf{H}(s) \in C^{p \times m}$, is defined by the equation

$$\mathbf{Y}(s) = \mathbf{H}(s)\mathbf{U}(s), \quad \text{where} \quad [\mathbf{H}(s)]_{ij} = \frac{Y_i(s)}{U_j(s)} \tag{4.64}$$

that is, the (i, j) element in matrix $\mathbf{H}(s)$ is the Laplace transformation of the response in the i^{th} output when a unit impulse is applied at the j^{th} input, with zero initial conditions and with the remaining inputs equal to zero for all $t \geq 0$.

On the other hand, if we Laplace-transform (4.42) and (4.43) with zero initial conditions, we obtain

$$\mathbf{X}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s) \tag{4.65}$$

$$\mathbf{Y}(s) = \mathbf{C}\mathbf{X}(s) + \mathbf{D}\mathbf{U}(s) = \underbrace{(\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D})}_{\mathbf{H}(s)}\mathbf{U}(s) \tag{4.66}$$

For simplicity, and to be able to go deeper into the analysis, **in the remaining part of this section we will focus our attention on the class of scalar systems**, that is, systems with a single input and a single output (SISO systems). This means that $m = p = 1$, $\mathbf{B}$ becomes a column vector, $\mathbf{C}$ is a row vector, and $D = H(\infty)$ (in real systems it usually holds that $D = H(\infty) = 0$). For SISO systems, $H(s)$ is a quotient of polynomials in s, that is,

$$H(s) = \frac{\mathbf{C}\,\text{Adj}(s\mathbf{I} - \mathbf{A})\mathbf{B} + \mathbf{D}\,\det(s\mathbf{I} - \mathbf{A})}{\det(s\mathbf{I} - \mathbf{A})} \tag{4.67}$$

where Adj(o) denotes the adjoint matrix of (o).

A key issue is that the transfer function poles are eigenvalues of matrix $\mathbf{A}$. However, it is not true, in general, that the set of transfer function poles is identical to the set of matrix $\mathbf{A}$ eigenvalues. This can be appreciated through the following example.

Example 4.4

Let

$$\mathbf{A} = \begin{bmatrix} -2 & 1 \\ 0 & -3 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}, \qquad \mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix}, \qquad D = 0 \tag{4.68}$$

Then

$$H(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} = \frac{1}{(s+2)(s+3)}\begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}s+3 & 1\\ 0 & s+2\end{bmatrix}\begin{bmatrix}1\\ 0.5\end{bmatrix} \tag{4.69}$$

$$= \frac{0.5(s+2)}{(s+2)(s+3)} = \frac{0.5}{(s+3)} \tag{4.70}$$

Therefore, the transfer function has only one pole, although matrix **A** has two eigenvalues. We observe that there is a pole–zero cancellation in $H(s)$. This phenomenon is closely connected to the question of system properties, which is the central topic in Section 4.6.

To acquire a phenomenological feeling on this issue, consider again the magnetic levitation system in Example 4.2. If we define the current $i(t)$ as the system output we can immediately see that the transfer function from the input $e(t)$ to this output has only one pole. This contrasts with the fact that the dimension of the state is equal to three. The explanation for this is that, in our simplified physical model, the current $i(t)$ is unaffected by the position and the speed of the metallic sphere (note that we have neglected the changes in the inductance due to changes in the sphere position).

The key result is that **the transfer function may not provide the same amount of information than the state space model** for the same system.

An interesting problem is to obtain a state space description from a given transfer function. The reader must be aware that the resulting state space model does not reveal pole-zero cancellations; for that reason, the obtained description is known as a **minimal realization**.

There are many methods to go from the transfer function to a state space model. We present below one of those methods.

Consider a transfer function given by

$$H_T(s) = \frac{B_o(s)}{A_o(s)} + H_T(\infty) = \frac{b_{n-1}s^{n-1} + b_{n-2}s^{n-2} + \cdots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0} + H_T(\infty) \tag{4.71}$$

We first recall that $D = H_T(\infty)$. We can thus concentrate on the transfer function $H(s) = H_T(s) - H_T(\infty)$, which is a strictly proper transfer function.

Consider next a variable $v_\ell(t) \in \mathbb{R}$ whose Laplace transform, $V_\ell(s)$, satisfies

$$V_\ell(s) = \frac{s^{\ell-1}}{A_o(s)}U(s), \quad \ell \in \{1, 2, \ldots, n\} \tag{4.72}$$

This implies that

$$v_\ell(t) = \frac{dv_{\ell-1}(t)}{dt}, \quad \ell \in \{2, \ldots, n\} \tag{4.73}$$

$$Y(s) = \sum_{\ell=1}^{n} b_{\ell-1}V_\ell(s) \tag{4.74}$$

$$U(s) = \frac{A_o(s)}{A_o(s)}U(s) = \underbrace{\frac{s^n}{A_o(s)}U(s)}_{sV_n(s)} + \sum_{\ell=1}^{n} a_{\ell-1}\underbrace{\frac{s^{\ell-1}}{A_0(s)}U(s)}_{V_\ell(s)} \tag{4.75}$$

Now choose as state variables,

$$x(t) = v(t) \tag{4.76}$$

The above equations yield

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \mathrm{A} & \mathrm{A} & \mathrm{A} & & \mathrm{A} & \mathrm{A} \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{A} \\ 0 \\ 1 \end{bmatrix} \tag{4.77}$$

$$\mathbf{C} = [b_0 \quad b_1 \quad b_2 \quad \cdots \quad b_{n-1}], \qquad \mathbf{D} = H_T(\infty) \tag{4.78}$$

Example 4.5

The transfer function of a system is given by

$$H(s) = \frac{4s-10}{(s+2)^2(s-1)} = \frac{4s-10}{s^3+3s^2-4} \tag{4.79}$$

Then a minimal realization for this system is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 4 & 0 & -3 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.80}$$

$$\mathbf{C} = \begin{bmatrix} -10 & 4 & 0 \end{bmatrix}, \qquad \mathbf{D} = 0 \tag{4.81}$$

A key result is that a **system transfer function is invariant with respect to state similarity transformations**.

4.4 State Space Description for Discrete-Time and Sampled Data Systems

In this section we will present an overview of the state space description for discrete time systems, mainly based on the results presented for the continuous time case.

Discrete time models may arise from two different sources:

- From a *pure* discrete-time system, usually nonlinear, whose variables are defined only at specific time instants t_k. Systems like that can be found in economic systems, stochastic process theory, etc.
- From a discretization of a continuous-time system. In this case, we are only concerned with the value of some system variables at specific time instants. These models are useful when digital systems, such as microcontrollers, computers, PLCs, or others, interact with continuous-time *real systems* such as mechanical structures, valves, tanks, analog circuits or a whole industrial process*. These are called **sampled data systems**.

In both cases our analysis will be focused on the class of **linear and time invariant** models.

*Through *digital-to-analog* and *analog-to-digital* converters (DAC and ADC, respectively).

4.4.1 Linearization of Discrete Time-Systems

The discrete time equivalents to (4.3) and (4.4) are given by the nonlinear equations

$$\mathbf{x}[t+1] = \mathbf{F}_d(\mathbf{x}[t], \mathbf{u}[t]) \tag{4.82}$$

$$\mathbf{y}[t] = \mathbf{G}_d(\mathbf{x}[t], \mathbf{u}[t]) \tag{4.83}$$

The linearization of models for discrete time systems follows along the same lines to that for continuous ones. Consider firstly an equilibrium point given by $\{\mathbf{x}_Q, \mathbf{u}_Q, \mathbf{y}_Q\}$:

$$\mathbf{x}_Q = \mathbf{F}_d(\mathbf{x}_Q, \mathbf{u}_Q) \tag{4.84}$$

$$\mathbf{y}_Q = \mathbf{G}_d(\mathbf{x}_Q, \mathbf{u}_Q) \tag{4.85}$$

Note that an equilibrium point is defined by a set of **constant** values of the state and **constant** values of the input which satisfy (4.82) and (4.83). This yields a constant system output. The discrete model can then be linearized around this equilibrium point. Defining

$$\Delta\mathbf{x}[t] = \mathbf{x}[t] - \mathbf{x}_Q, \qquad \Delta\mathbf{u}[t] = \mathbf{u}[t] - \mathbf{u}_Q, \qquad \Delta\mathbf{y}[t] = \mathbf{y}[t] - \mathbf{y}_Q \tag{4.86}$$

we have the state space model

$$\Delta\mathbf{x}[t+1] = \mathbf{A}_d\Delta\mathbf{x}[t] + \mathbf{B}_d\Delta\mathbf{u}[t] \tag{4.87}$$

$$\Delta\mathbf{y}[t] = \mathbf{C}_d\Delta\mathbf{x}[t] + \mathbf{D}_d\Delta\mathbf{u}[t] \tag{4.88}$$

where

$$\mathbf{A}_d = \left.\frac{\partial \mathbf{F}_d}{\partial \mathbf{x}}\right|_{\substack{x=x_Q \\ u=u_Q}}, \qquad \mathbf{B}_d = \left.\frac{\partial \mathbf{F}_d}{\partial \mathbf{u}}\right|_{\substack{x=x_Q \\ u=u_Q}}, \qquad \mathbf{C}_d = \left.\frac{\partial \mathbf{G}_d}{\partial \mathbf{x}}\right|_{\substack{x=x_Q \\ u=u_Q}}, \qquad \mathbf{D}_d = \left.\frac{\partial \mathbf{G}_d}{\partial \mathbf{u}}\right|_{\substack{x=x_Q \\ u=u_Q}} \tag{4.89}$$

4.4.2 Sampled Data Systems

As we have already said, discrete time models are frequently obtained by sampling inputs and outputs in continuous-time systems. When a digital device is to be used to act upon a continuous-time system, the command signals need only to be defined at specific instants, and not at *all* time. However, to be able to act upon the continuous-time system, we need a continuous-time signal. This is usually built with a *zero order hold*, which generates a staircase signal. Also, when we want to digitally measure a system variable this is done at some specific time instants. This means that we must **sample** the output signals. Figure 4.5 illustrates these concepts. If we assume a periodic sampling, with period Δ, we are only interested in the signals at time $k\Delta$. In the sequel we will drop Δ from the arguments, using $\mathbf{u}(k\Delta) = \mathbf{u}[t]$ for the input, $\mathbf{y}(k\Delta) = \mathbf{y}[t]$ for the output, and $\mathbf{x}(k\Delta) = \mathbf{x}[t]$ for the system state.

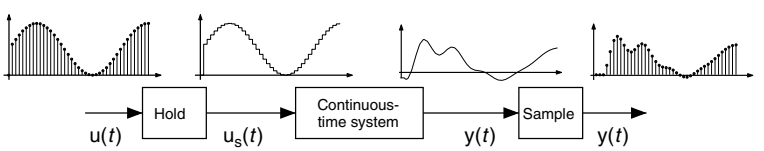


FIGURE 4.5 Schematic representation of a sampled data system.

If we consider the continuous, time-invariant, and linear state space model defined by Equations 4.42 and 4.43, with initial state $\mathbf{x}(k_0\Delta) = \mathbf{x}_0$, we can use Equation 4.44 to calculate the next value of the state:

$$\mathbf{x}(k_0\Delta + \Delta) = e^{\mathbf{A}(k_0\Delta+\Delta-k_0\Delta)}\mathbf{x}(k_0\Delta) + \int_{k_0\Delta}^{k_0\Delta+\Delta} e^{\mathbf{A}(k_0\Delta+\Delta-\tau)}\mathbf{B}\,\mathbf{u}(\tau)\,d\tau \tag{4.90}$$

Furthermore, if a zero order hold is used, that is, $\mathbf{u}(\tau) = \mathbf{u}(k_0\Delta)$ for $k_0\Delta \le \tau < k_0\Delta + \Delta$, we obtain

$$\mathbf{x}(k_0\Delta + \Delta) = e^{\mathbf{A}\Delta}\mathbf{x}(k_0\Delta) + \int_0^{\Delta} e^{\mathbf{A}\eta}d\eta\mathbf{B}\,\mathbf{u}(k_0\Delta) \tag{4.91}$$

And, if we know the state and the input at time $k_0\Delta$, the output is defined by Equation 4.43:

$$\mathbf{y}(k_0\Delta) = \mathbf{C}\mathbf{x}(k_0\Delta) + \mathbf{D}\,\mathbf{u}(k_0\Delta) \tag{4.92}$$

We can now conclude that given a continuous-time model with state space matrices {**A**, **B**, **C**, **D**}, and we sample inputs and outputs every Δ seconds then, the equivalent sampled data systems will be described by the discrete-time state space model:

$$\mathbf{x}(k\Delta + \Delta) = \mathbf{A}_\mathbf{d}\mathbf{x}(k\Delta) + \mathbf{B}_\mathbf{d}\mathbf{u}(k\Delta) \tag{4.93}$$

$$\mathbf{y}(k\Delta) = \mathbf{C}_\mathbf{d}\mathbf{x}(k\Delta) + \mathbf{D}_\mathbf{d}\mathbf{u}(k\Delta) \tag{4.94}$$

where

$$\mathbf{A}_\mathbf{d} = e^{\mathbf{A}\Delta}, \quad \mathbf{B}_\mathbf{d} = \int_0^{\Delta} e^{\mathbf{A}\eta}d\eta\mathbf{B}, \quad \mathbf{C}_\mathbf{d} = \mathbf{C}, \quad \mathbf{D}_\mathbf{d} = \mathbf{D} \tag{4.95}$$

There are different methods to obtain $\mathbf{A}_\mathbf{d}$ defined in (4.95), but a simple way to calculate this matrix is to use Laplace transformation. This yields

$$\mathbf{A}_\mathbf{d} = e^{\mathbf{A}\Delta} = L^{-1}\{(s\mathbf{I} - \mathbf{A})^{-1}\}\big|_{t=\Delta} \tag{4.96}$$

Example 4.6

Consider the mechanical system of Example 4.1 on p. 4, that was described by the state space model:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{D}{M} \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{M} \end{bmatrix} f(t) \tag{4.97}$$

where $f(t)$ is the external force, and where we can choose either the mass position, $x_1(t)$, or the mass velocity, $x_2(t)$, of the mass, as the system output.

For the purpose of a numerical illustration, we set $M = 1$ kg, $D = 1.2$ N s/m, and $K = 0.32$ N/m.

The matrix $\mathbf{A}_\mathbf{d}$ is obtained from (4.96), applying inverse Laplace transformation

$$\mathbf{A}_\mathbf{d} = L^{-1}\left\{\begin{bmatrix} s & -1 \\ 0.32 & s+1.2 \end{bmatrix}^{-1}\right\}\Bigg|_{t=\Delta} = \begin{bmatrix} 2e^{-0.4\Delta} - e^{-0.8\Delta} & 2.5(e^{-0.4\Delta} - e^{-0.8\Delta}) \\ 0.8(e^{-0.4\Delta} - e^{-0.8\Delta}) & -e^{-0.4\Delta} + 2e^{-0.8\Delta} \end{bmatrix} \tag{4.98}$$

and the $\mathbf{B}_d$ matrix is obtained from (4.95):

$$\begin{aligned}\mathbf{B}_d &= \int_0^{\Delta} \begin{bmatrix} 2e^{-0.4\eta} - e^{-0.8\eta} & 2.5(e^{-0.4\eta} - e^{-0.8\eta}) \\ 0.8(e^{-0.4\eta} - e^{-0.8\eta}) & -e^{-0.4\eta} + 2e^{-0.8\eta} \end{bmatrix} d\eta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \Rightarrow \mathbf{B}_d &= \begin{bmatrix} -6.25e^{-0.4\Delta} + 3.125e^{-0.8\Delta} + 3.125 \\ 2.5(e^{-0.4\Delta} - e^{-0.8\Delta}) \end{bmatrix}\end{aligned} \tag{4.99}$$

Note that both, $\mathbf{A}_d$ and $\mathbf{B}_d$ are functions of Δ. Thus, the sampling period, Δ, has a strong presence in the dynamic behavior of the sampled system, as we shall observe in the following subsections.

4.4.3 Linear State Space Models

We will analyze the linear time invariant state space model

$$\mathbf{x}[t+1] = \mathbf{A}_d\mathbf{x}[t] + \mathbf{B}_d\mathbf{u}[t] \tag{4.100}$$

$$\mathbf{y}[t] = \mathbf{C}_d\mathbf{x}[t] + \mathbf{D}_d\mathbf{u}[t] \tag{4.101}$$

This can be a linearized discrete time model like (4.87) and (4.88), or a sampled data system like (4.93) and (4.94) where Δ has been dropped from the time argument.

The solution to Equations 4.100 and 4.101, subject to $\mathbf{x}[t_o] = \mathbf{x}_o$, is given by

$$\mathbf{x}[t] = \mathbf{A}_d^{(t-t_o)}\mathbf{x}_o + \sum_{i=0}^{(t-t_o)-1} \mathbf{A}_d^{(t-t_o)-i-1}\mathbf{B}_d\mathbf{u}[i+t_o] \quad \forall t \geq t_o \tag{4.102}$$

where $\mathbf{A}_d^{(t-t_o)}$ is the **transition matrix**.

The reader can check easily that (4.102) satisfies (4.100). With the above result, the solution for (4.101) is given by

$$\mathbf{y}[t] = \mathbf{C}_d\mathbf{A}_d^{(t-t_o)}\mathbf{x}_o + \mathbf{C}_d \sum_{i=0}^{(t-t_o)-1} (\mathbf{A}_d^{(t-t_o)-i-1}\mathbf{B}_d\mathbf{u}[i+t_o]) + \mathbf{D}_d\mathbf{u}[t] \tag{4.103}$$

4.4.3.1 System Dynamics

The state of the system has two components: the **unforced** component, $\mathbf{x}_u[t]$, and the **forced** component, $\mathbf{x}_f[t]$, where

$$\mathbf{x}_u[t] = \mathbf{A}_d^{(t-t_o)}\mathbf{x}_o \tag{4.104}$$

$$\mathbf{x}_f[t] = \sum_{\tau=0}^{(t-t_o)-1} \mathbf{A}_d^{(t-t_o)-i-1}\mathbf{B}_d\mathbf{u}[i+t_o] \tag{4.105}$$

To gain insight into the state space model and its solution consider the case when $t_o = 0$ and $u[t] = 0$, $\forall t \geq 0$, that is, the state has only the **unforced part**. Then

$$\mathbf{x}[t] = \mathbf{A}_d^t\mathbf{x}_o \tag{4.106}$$

Further assume that $\mathbf{A}_d \in \mathbf{R}^{n\times n}$ and that, for simplicity, it has n distinct eigenvalues η, with n linearly independent eigenvectors $\mathbf{v}$. Then there always exists a set of n constants α such that

$$\mathbf{x}_o = \sum_{\ell=1}^{n} \alpha_\ell \mathbf{v}_\ell, \quad \alpha_\ell \in \mathbb{C} \tag{4.107}$$

A well known result from linear algebra tells us that the eigenvalues of $\mathbf{A}_d^k$ are η_ℓ^k, for $k \in \mathrm{N}$, with corresponding eigenvectors $\mathbf{v}$. The application of this result yields

$$\mathbf{x}[t] = \mathbf{A}_d^t \mathbf{x}_o = \mathbf{A}_d^t \sum_{\ell=1}^{n} \alpha_\ell \mathbf{v}_\ell = \sum_{\ell=1}^{n} \alpha_\ell \underbrace{\mathbf{A}_d^t \mathbf{V}_\ell}_{\eta_\ell^t \mathbf{v}_\ell} \tag{4.108}$$

$$\mathbf{x}[t] = \sum_{\ell=1}^{n} \alpha_\ell \eta_\ell^t \mathbf{v}_\ell \tag{4.109}$$

This equation shows that the unforced component of the state is a linear combination of **natural modes**, $\{\eta_\ell^t\}$, and each one is associated with an eigenvalue of $\mathbf{A}_d$, which are also known as **natural frequencies** of the model. Thus, we again have that the matrix $\mathbf{A}_d$ determines:

- Structure of the unforced response
- Stability (or otherwise) of the system
- Speed of response

4.4.3.2 Structure of the Unforced Response

In the absence of input, the state evolves as a combination of natural modes which belong to a defined class of functions: the powers of the model eigenvalues, either real or complex. These modes are discrete functions related to constants, real exponentials, pure sine waves, exponentially modulated sine waves, and some other specials functions arising from repeated eigenvalues.

To illustrate these ideas and their physical interpretation consider the sampled system in Example 4.6. If $\Delta = 1$, the state space matrices are

$$\mathbf{A}_d = \begin{bmatrix} 0.8913 & 0.5525 \\ -0.1768 & 0.2283 \end{bmatrix}, \qquad \mathbf{B}_d = \begin{bmatrix} 0.3397 \\ 0.5525 \end{bmatrix} \tag{4.110}$$

Hence, the system eigenvalues are solutions to the equation

$$\det(\eta\mathbf{I} - \mathbf{A}_d) = \det\left(\begin{bmatrix} \eta - 0.8913 & -0.5525 \\ 0.1768 & \eta - 0.2283 \end{bmatrix}\right) \tag{4.111}$$

$$= (\eta - 0.6703)(\eta - 0.4493) = 0 \tag{4.112}$$

that is, $\eta_1 = 0.6703$, $\eta_2 = 0.4493$, and the unforced response is

$$\mathbf{x}_u[t] = C_1(0.6702)^t + C_2(0.4493)^t \tag{4.113}$$

where $C_1$and C_2 depend on the initial conditions only. We can observe that, when t tends to infinity, $\mathbf{x}_u[t]$ decays to zero, because $|\eta_{1,2}| < 1$. Also these eigenvalues are positive real numbers, so there is no oscillation

in the natural modes. This last observation is consistent with the parameter choice in Example 1.6, which made the mass-spring system to be overdamped.

4.4.3.3 Structure of the Forced Response

Consider the Equation 4.102. Then, when the initial state is zero, the state will only exhibit the forced component. However, the forced component will still include natural modes plus some additional **forced** or **particular** modes, which depend on the nature of the system input $\mathbf{u}[t]$. In general, the forcing modes in the input will also appear in the state. However, special cases arise when a forcing mode in $\mathbf{u}[t]$ coincides with a system natural mode.

4.4.3.4 System Stability

Stability in linear time-invariant systems can also be analyzed using the state matrix $\mathbf{A}_d$. As we said, all systems variables can be expressed as linear functions of the state and the system input. When the system input $\mathbf{u}[t]$ is a vector of bounded time functions, then the boundedness of the system variables depends on the state to be bounded. We then have the following result:

Theorem 4.2 *Consider a system with the state description* (4.100) *and* (4.101) *where* $\mathbf{B}_d$, $\mathbf{C}_d$, *and* $\mathbf{D}_d$ *have bounded elements. Then the system state is bounded for all bounded inputs if and only if the eigenvalues of* $\mathbf{A}_d$ *lies inside the unit disc, that is,* $|\eta| < 1, \ \forall, .$

4.4.3.5 Speed of Response and Resonances

We recall that the natural modes of discrete-time systems are the powers of the eigenvalues η. Since those eigenvalues can always be described as complex quantities, we can then write the natural modes as

$$(\eta_\ell)^t = (|\eta_\ell| e^{j\theta_\ell})^t = |\eta_\ell|^t e^{j\theta_\ell t}, \quad \text{where} \quad \theta_\ell = \angle\eta_\ell \tag{4.114}$$

Therefore, we have that

- $0 < |\eta| < \infty$ determines the *speed* at which the mode decays to zero for stable systems ($|\eta| < 1$), or grows to infinity for unstable systems ($|\eta| > 1$)
- $-\pi < \theta \leq \pi$ determines the *frequency* of the natural mode, measured in radians

Although the natural modes of stable systems decay to zero, their nature determines the system transient response.

To illustrate these issues the **step response**, with zero initial conditions, is frequently used.

Example 4.7

Consider the first order, single-input single-output discrete-time system

$$\mathbf{x}[t+1] = \eta \mathbf{x}[t] + \mathbf{u}[t] \tag{4.115}$$

$$\mathbf{y}[t] = (1-\eta)\mathbf{x}[t] \tag{4.116}$$

To obtain the step response, we can use the Equation 4.103, where $\mathbf{x}_o = 0$, $\mathbf{u}[t] = 1$, $\forall t \geq 0$.

$$\mathbf{y}[t] = \mathbf{C}_d \left(\sum_{i=0}^{t-1} \mathbf{A}_d^{t-i-1} \right) \mathbf{B}_d \tag{4.117}$$

$$= (1-\eta_\ell) \left(\sum_{i=0}^{t-1} \eta_\ell^{t-i-1} \right) = (1-\eta_\ell)\eta_\ell^{t-1} \frac{1-\eta_\ell^{-t}}{1-\eta_\ell^{-1}} \tag{4.118}$$

$$= 1 - \eta^t \tag{4.119}$$

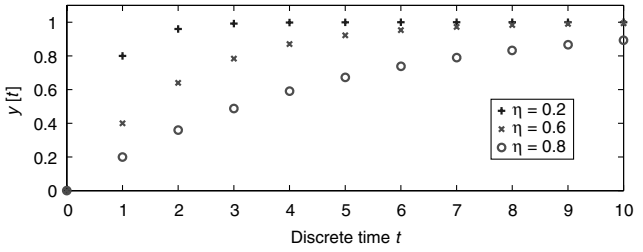


FIGURE 4.6 Step response of the system for different eigenvalues.

The output signal, $y[t] = y_h[t] + y_p[t]$, is shown in Figure 4.6, for different values of the eigenvalue η. The transient is given by $\mathrm{y_h}[t] = -\eta^t$, and the steady state response by $\mathrm{y_p}[t] = 1$.

We observed in Equation 4.114 that the system eigenvalues define the damping of its transient response, but also determine its frequency of oscillation (when the eigenvalues have a nonzero imaginary part). The potential problem when resonant modes exist is the same problem we found in the context of continuous-time systems, that is, the system input contains a sine wave or another kind of signal, with energy at a frequency close to one of the natural frequencies of the system. The system output still remains bounded, although it grows to undesirable amplitudes.

Example 4.8

Consider the discrete-time system described by the state space model

$$\mathbf{x}[t+1] = \begin{bmatrix} 1.2796 & -0.81873 \\ 1 & 0 \end{bmatrix} \mathbf{x}[t] + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u[t] \tag{4.120}$$

$$y[t] = \begin{bmatrix} 0 & 0.5391 \end{bmatrix} \mathbf{x}[t] \tag{4.121}$$

The eigenvalues of the system are obtained from $\mathbf{A_d}$:

$$\eta_{1,2} = 0.6398 \pm j0.6398 = 0.9048\ (e^{j\pi/4}) \tag{4.122}$$

And the associated natural modes, present in the transient response, are

$$\eta_{1,2}^t = 0.9048^t e^{j\frac{\pi}{4}t} = 0.9048^t \left[\cos\left(\frac{\pi}{4}t\right) \pm j\sin\left(\frac{\pi}{4}t\right)\right] \tag{4.123}$$

The natural modes are slightly damped, because $|\eta_{1,2}|$ is close to 1, and they show an oscillation of frequency $\pi/4$.

In the plots shown in Figure 4.7 we appreciate a strongly resonant output. The upper plot corresponds to an input $u[t] = \sin(\frac{\pi}{4}t)$, that is, the input frequency coincides with the frequency of the natural modes. In the lower plot the input is a square wave of frequency input signal $\pi/12$. In this case, the input **third** harmonic has a frequency equal to the frequency of the natural modes.

4.4.3.6 Effect of Different Sampling Periods

We observe in Equation 4.95 that $\mathbf{A_d}$ and $\mathbf{B_d}$ depend on the choice of the sampling period Δ. This choice determines the position of the eigenvalues of the system too. If we look at the Equation 4.96, assuming that **A** has been diagonalized, we have that

$$\mathbf{A_d} = e^{\mathrm{diag}\{\lambda_1,\ldots,\lambda_n\}\Delta} = \mathrm{diag}\{e^{\lambda_1\Delta},\ldots,e^{\lambda_n\Delta}\} \tag{4.124}$$

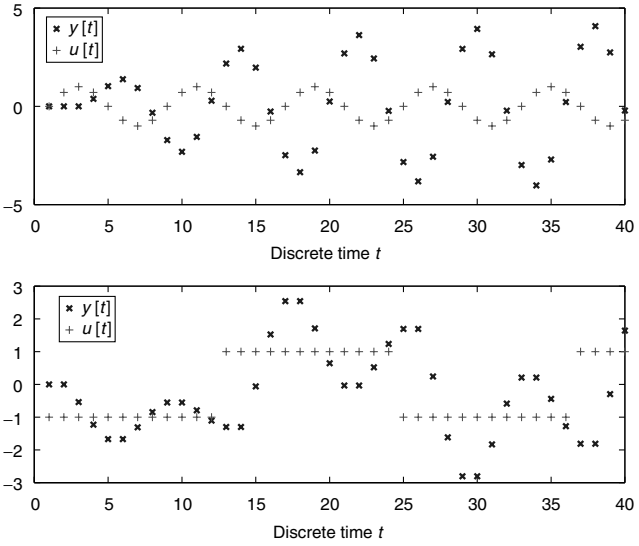


FIGURE 4.7 Resonant effect in the system output.

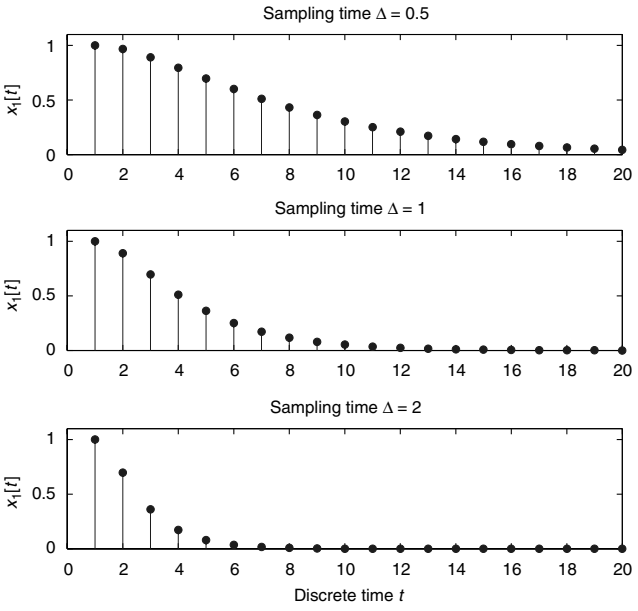


FIGURE 4.8 Effect of sampling in natural modes.

where $\{\lambda_1, \ldots, \lambda_n\}$ are the eigenvalues of the underlying continuous-time systems. Then, these eigenvalues are mapped to the eigenvalues of the sampled-data system by equation:

$$\eta_\ell = e^{\lambda_\ell \Delta} \tag{4.125}$$

In Figure 4.8 we observe the response of the sampled system of Example 4.6, choosing $x_1[t]$ as the system output, when the initial condition is $\mathbf{x}_o = [1 \quad 0]^T$, for different values of Δ. Observe that the horizontal axis corresponds to t, so the *real* instants times are $t\Delta$.

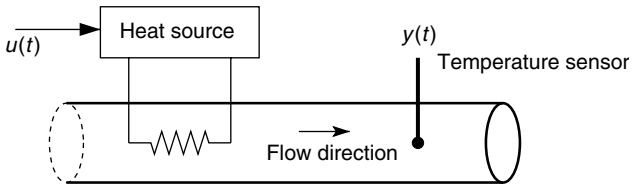


FIGURE 4.9 Heating system with time delay.

A fundamental issue regarding sampling of continuous-time signals is that the sampling period has to be chosen small enough to capture the essential nature of the signal to be sampled. To exemplify an ill-chosen Δ, assume that the signal $f(t) = A\sin(\omega_o t)$ is sampled every Δ seconds, with $\Delta = 2\ \pi/\omega_o, \quad \in \mathbb{N}$. Then the resulting discrete time signal is $f[t] = 0,\ \forall t \in \mathrm{Z}$.

4.4.3.7 Sampled Data Systems and Time Delays

We said in Section 4.3 that one cannot use continuous-time state space models to describe systems with time delays, because they are infinite dimensional systems. It was also said there that we would be able to tackle this problem using sampled signals. This is done using the following example.

Example 4.9

Consider the heating system sketched in Figure 4.9.

The measured temperature, $y(t)$, of the flow depends on the power injected by the heat source. This source is commanded by a control signal $u(t)$. Changes in $u(t)$ yield changes in the temperature $y(t)$, but with a significant time delay. The linearized system can thus be represented by the transfer function:

$$\frac{Y(s)}{U(s)} = H(s) = \frac{e^{-\tau s}K}{s+\lambda} \tag{4.126}$$

where $U(s)$ and $Y(s)$ are the Laplace transforms of $u(t)$ and $y(t)$, respectively.

We next assume that the input and output signals are sampled every $\Delta[s]$. The time delay τ, in seconds, is a function of the flow velocity and we can assume, for simplicity, that τ it is a multiple of the sampling interval Δ, that is, $\tau = m\Delta,\ m \in \mathrm{Z}^+$. These delays translate in a factor z^m in the denominator of the Z-transform transfer function. In other words, the delay gives rise to a set of m poles at the origin. Furthermore, the continuous-time system eigenvalue at $s = -\lambda$ becomes a discrete-time system eigenvalue at $z = e^{-\lambda\Delta}$ (see Equation 4.125). The resulting transfer function is

$$\frac{Y[z]}{U[z]} = H[z] = \frac{K}{\lambda}\frac{1-e^{-\lambda\Delta}}{z^m(z-e^{-\lambda\Delta})} \tag{4.127}$$

And this transfer function can be expressed as the discrete state space model

$$x_1[t+1] = x_2[t] \tag{4.128}$$

$$x_2[t+1] = x_3[t] \tag{4.129}$$

$$\mathrm{A} \qquad \mathrm{A}$$

$$x_m[t+1] = x_{m+1}[t] \tag{4.130}$$

$$x_{m+1}[t+1] = e^{-\lambda\Delta}x_{m+1}[t] + \frac{K}{\lambda}(1-e^{-\lambda\Delta})\,u[t] \tag{4.131}$$

$$y[t] = x_1[t] \tag{4.132}$$

We can then think of the states variables $x_{m+1}[t], \ldots, x_1[t]$ as the temperature at equally spaced points, between the heat source and the temperature sensor.

When the time delay τ is not a multiple of the sampling period Δ, an additional pole at the origin and an additional zero appear in the discrete transfer function. The details can be found elsewhere, for example, in [7].

4.4.4 State Similarity Transformation

The idea of transforming the state via a similarity transformation equally applies to discrete-time systems. The system properties also remain unchanged.

4.4.5 State Space and Transfer Functions

For discrete-time systems the relation between state space and transfer function models is basically the same as in the continuous-time case. As we said then, the state space description of linear time invariant systems is an alternative description to that provided by transfer functions, although in some situations it provides more information on the system.

For a linear discrete-time invariant system with input $\mathbf{u}[t] \in \mathbb{R}^m$ and output $\mathbf{y}[t] \in \mathbb{R}^p$, the transfer function, $\mathbf{H}[z] \in C^{p\times m}$, is defined by the equation

$$\mathbf{Y}[z] = \mathbf{H}[z]\mathbf{U}[z], \quad \text{where} \quad [\mathbf{H}[z]]_{ij} = \frac{Y_i[z]}{U_j[z]} \tag{4.133}$$

that is, the (i, j) element in matrix $\mathbf{H}[z]$ is the Zeta transformation of the response in the i^{th} output when a unit Kronecker's delta is applied at the j^{th} input, with zero initial conditions and with the remaining inputs equal to zero for all $t \geq 0$.

On the other hand, if we apply Zeta transform to the discrete time state space model (4.100) and (4.101), with zero initial conditions, we have

$$\mathbf{X}[z] = (z\mathbf{I} - \mathbf{A}_\mathrm{d})^{-1}\mathbf{B}_\mathrm{d}\mathbf{U}[z] \tag{4.134}$$

$$\mathbf{Y}[z] = \mathbf{C}_\mathrm{d}\mathbf{X}[z] + \mathbf{D}_\mathrm{d}\mathbf{U}[z] \tag{4.135}$$

leading to

$$\mathbf{C}_\mathrm{d}(z\mathbf{I} - \mathbf{A}_\mathrm{d})^{-1}\mathbf{B}_\mathrm{d} + \mathbf{D}_\mathrm{d} = \mathbf{H}[z] \tag{4.136}$$

In the following analysis, we will focus on the class of scalar systems, that is, $m = p = 1$, $\mathbf{B}_\mathrm{d}$, $\mathbf{C}_\mathrm{d}^T$ are column vectors, and $\mathbf{D}_\mathrm{d} = H[\infty]$. We can then see that $H[z]$ is a quotient of polynomials in z, that is,

$$H[z] = \frac{\mathbf{C}_\mathrm{d}\,\mathrm{Adj}(z\mathbf{I} - \mathbf{A}_\mathrm{d})\mathbf{B}_\mathrm{d} + \mathbf{D}_\mathrm{d}\det(z\mathbf{I} - \mathbf{A}_\mathrm{d})}{\det(z\mathbf{I} - \mathbf{A}_\mathrm{d})} \tag{4.137}$$

where Adj(o) denotes the adjoint matrix of (o).

We have again, paralleling the continuous-time case, that the transfer function poles are eigenvalues of $\mathbf{A}_\mathrm{d}$. However, it is not true in general that the set of transfer function poles is identical to the set of eigenvalues of the matrix. It is important to realize that transfer function models can hide cancellations between poles and zeros, with the consequences described in Section 4.6.1 and Section 4.6.2.

A key result for discrete-time system is the same for continuous-time systems: **the transfer function may not provide the same amount of information than the state space model** for the same system.

One way to obtain the state space model is to use the same method proposed in Section 4.3.4 "State Space and Transfer Functions," applying Zeta transformation instead of Laplace transformation, and using the fact that

$$F[z] = \mathcal{Z}\{f[t]\} \Leftrightarrow zF[z] = \mathcal{Z}\{f[t+1]\} \tag{4.138}$$

Example 4.10

The transfer function of a system is given by

$$H[z] = \frac{2z^2 - z + 1}{(z-0.8)(z-0.6)} = \frac{1.8z + 0.04}{z^2 - 1.4z + 0.48} + 2 \tag{4.139}$$

Then a minimal realization for this system is

$$\mathbf{A}_d = \begin{bmatrix} 0 & 1 \\ -0.48 & -1.4 \end{bmatrix}, \qquad \mathbf{B}_d = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.140}$$

$$\mathbf{C}_d = [0.04 \quad 1.8], \qquad \mathbf{D}_d = 2 \tag{4.141}$$

In discrete-time models it also happens that the **system transfer function is invariant with respect to state similarity transformations.**

4.5 State Space Models for Interconnected Systems

To build state space models for complex systems it is sometimes useful (and possible) to describe them as the interconnection of simpler systems. That interconnection is usually a combination of three basic interconnection structures: series, parallel, and feedback. In those three basic cases our aim is to obtain a state space model for the composite system.

In the following analysis we will use two systems, which are defined by

$$\text{System 1:} \qquad \frac{d\mathbf{x}_1(t)}{dt} = \mathbf{A}_1\mathbf{x}_1(t) + \mathbf{B}_1\mathbf{u}_1(t) \tag{4.142}$$

$$\mathbf{y}_1(t) = \mathbf{C}_1\mathbf{x}_1(t) + \mathbf{D}_1\mathbf{u}_1(t) \tag{4.143}$$

$$\text{System 2:} \qquad \frac{d\mathbf{x}_2(t)}{dt} = \mathbf{A}_2\mathbf{x}_2(t) + \mathbf{B}_2\mathbf{u}_2(t) \tag{4.144}$$

$$\mathbf{y}_2(t) = \mathbf{C}_2\mathbf{x}_2(t) + \mathbf{D}_2\mathbf{u}_2(t) \tag{4.145}$$

4.5.1 Series Connection

The system interconnection shown in Figure 4.10 is known as a series or cascade connection. To build the desired state space model, we first observe that $\mathbf{y}_2(t) = \mathbf{u}_1(t)$. Also, the composite system input is $\mathbf{u}(t) = \mathbf{u}_2(t)$,

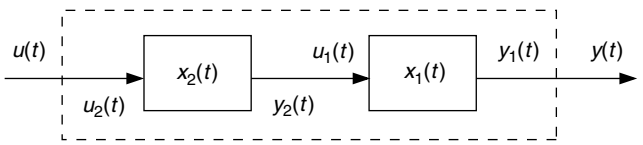


FIGURE 4.10 Series connection.

and the composite system output is $\mathbf{y}(t) = \mathbf{y}_1(t)$. We thus obtain

$$\begin{bmatrix} \dot{\mathbf{x}}_1(t) \\ \dot{\mathbf{x}}_2(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1\mathbf{C}_2 \\ 0 & \mathbf{A}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1\mathbf{D}_2 \\ \mathbf{B}_2 \end{bmatrix} \mathbf{u}(t) \tag{4.146}$$

$$\mathbf{y}(t) = [\mathbf{C}_1 \;\; \mathbf{D}_1\mathbf{C}_2] \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} + [\mathbf{D}_1\mathbf{D}_2]\, \mathbf{u}(t) \tag{4.147}$$

4.5.2 Parallel Connection

The system interconnection shown in Figure 4.11 is known as a parallel connection. To obtain the desired state space model we observe that the input is $\mathbf{u}(t) = \mathbf{u}_1(t) = \mathbf{u}_2(t)$ and the output for the whole system is $\mathbf{y}(t) = \mathbf{y}_1(t) + \mathbf{y}_2(t)$. We obtain

$$\begin{bmatrix} \dot{\mathbf{x}}_1(t) \\ \dot{\mathbf{x}}_2(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} \mathbf{u}(t) \tag{4.148}$$

$$\mathbf{y}(t) = [\mathbf{C}_1 \;\; \mathbf{C}_2] \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} + [\mathbf{D}_1 + \mathbf{D}_2]\, \mathbf{u}(t) \tag{4.149}$$

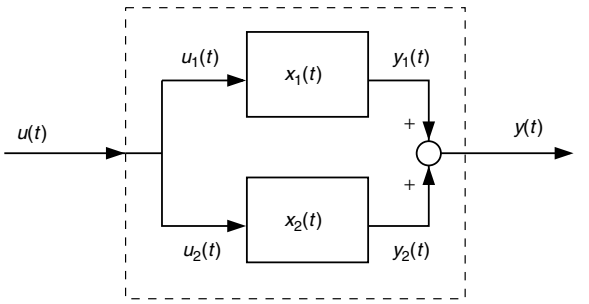


FIGURE 4.11 Parallel connection.

4.5.3 Feedback Connection

The system interconnection shown in Figure 4.12 is known as feedback connection (with unit negative feedback), and it corresponds to the basic structure of a control loop, where S_1 is the *plant* and S_2 is the *controller*. To build the composite state space model we observe that the overall system input satisfies the equation $\mathbf{u}(t) = \mathbf{u}_2(t) + \mathbf{y}_1(t)$, and the overall system output is $\mathbf{y}(t) = \mathbf{y}_1(t)$. Furthermore, we assume

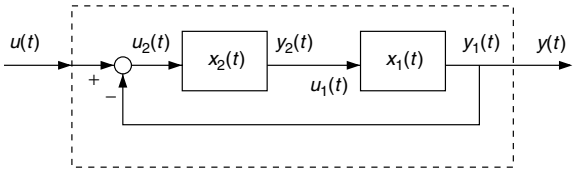


FIGURE 4.12 Feedback connection.

that the system S_1 (the plant) is strictly proper, that is, $\mathbf{D}_1 = 0$. We then obtain

$$\begin{bmatrix}\dot{\mathbf{x}}_1(t)\\ \dot{\mathbf{x}}_2(t)\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1 - \mathbf{B}_1\mathbf{D}_2\mathbf{C}_1 & \mathbf{B}_1\mathbf{C}_2\\ -\mathbf{B}_2\mathbf{C}_1 & \mathbf{A}_2\end{bmatrix}\begin{bmatrix}\mathbf{x}_1(t)\\ \mathbf{x}_2(t)\end{bmatrix} + \begin{bmatrix}\mathbf{B}_1\mathbf{D}_2\\ \mathbf{B}_2\end{bmatrix}\mathbf{u}(t) \tag{4.150}$$

$$\mathbf{y}(t) = [\mathbf{C}_1 \quad \mathbf{0}]\begin{bmatrix}\mathbf{x}_1(t)\\ \mathbf{x}_2(t)\end{bmatrix} \tag{4.151}$$

The same results apply, mutatis mutandis, to discrete-time interconnected systems. More details can be found elsewhere, for example, in [15].

4.6 System Properties

4.6.1 Controllability, Reachability, and Stabilizability

A very important question that we must be interested in regarding control systems using state space models is whether or not we can steer the state via the control input to certain locations in the state space. We must remember that the states of a system frequently are internal variables like temperature, pressure, level of tanks, or others. These are sometimes critical variables that we want to keep between specific values.

4.6.1.1 Controllability

The issue of controllability is concerned with whether or not a given initial state $\mathbf{x}_0$ can be steered to the origin in finite time using the input $\mathbf{u}(t)$.

Example 4.11

If we examine the model defined in (4.152), we note that the input $u(t)$ has no effect over the state $x_2(t)$.

$$\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}u(t) \tag{4.152}$$

Given an initial state $[x_1(0), x_2(0)]^T$, the input $u(t)$ can be chosen to steer $x_1(t)$ to zero, while $x_2(t)$ remains unchanged.

Formally, we have the following definition:

Definition 4.1 A state x_o is said to **controllable** if there exists a finite interval $[0, T]$ and an input $\{u(t), t \in [0, T]\}$ such that $x(T) = 0$. If all states are controllable, then the system is said to be **completely controllable**.

4.6.1.2 Reachability

A related concept is that of **reachability**, used sometimes in discrete-time systems. It is formally defined as follows:

Definition 4.2 A state $\bar{\mathbf{x}} \neq 0$ is said to be **reachable**, from the origin, if given $\mathbf{x}(0) = 0$, there exists a finite time interval $[0, T]$ and an input $\{\mathbf{u}(t), t \in [0, T]\}$ such that $\mathbf{x}(T) = \bar{\mathbf{x}}$. If all states are reachable the system is said to be **completely reachable**.

For continuous, time-invariant, linear systems, there is no distinction between complete controllability and reachability. However, the following example illustrates that there is a subtle difference in the

discrete-time case. Consider the system and the output

$$\mathbf{x}[t+1] = \underbrace{\begin{bmatrix} 0.5 & 1 \\ -0.25 & -0.5 \end{bmatrix}}_{\mathbf{A}_d} \mathbf{x}[t] \Rightarrow \mathbf{x}[t] = \begin{bmatrix} 0.5 & 1 \\ -0.25 & -0.5 \end{bmatrix}^t \mathbf{x}[0] \tag{4.153}$$

We can see that this system is completely controllable since $\mathbf{x}[t] = 0$, $\forall t \geq 2$ and $\forall \mathbf{x}[0] \in \mathrm{R}^2$. This implies that every initial state is controllable. However, no nonzero state is reachable.

In view of the distinction between controllability and reachability in discrete time, we will use the term *controllability* in the sequel to cover the stronger of the two concepts.

Usually, in the context of linear time invariant systems, controllability and reachability are used interchangeably.

4.6.1.3 Controllability Test

We now present a systematic way to determine the complete controllability of a system.

Theorem 4.3 *Consider the linear, time-invariant, state space model where* $\mathbf{A} \in \mathrm{R}^{n \times n}$*:*

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{4.154}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{4.155}$$

i) *The set of all controllable states is the* ***range space*** *of the controllability matrix* $\Gamma_c[\mathbf{A}, \mathbf{B}]$ *where*

$$\Gamma_c[\mathbf{A}, \mathbf{B}] \triangleq [\mathbf{B} \quad \mathbf{AB} \quad \mathbf{A}^2\mathbf{B} \quad \cdots \quad \mathbf{A}^{n-1}\mathbf{B}] \tag{4.156}$$

ii) *The model is completely controllable if and only if* $\Gamma_c[\mathbf{A}, \mathbf{B}]$ *has* ***full row rank.***

Example 4.12

Consider the state space model given in (4.152), with state space matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.157}$$

The controllability matrix for this system, is given by

$$\Gamma_c[\mathbf{A}, \mathbf{B}] = [\mathbf{B} \quad \mathbf{AB}] = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.158}$$

Clearly, rank $\Gamma_c[\mathbf{A}, \mathbf{B}] = 1$, thus the system is **not** completely controllable.

The result above applies to continuous-time models, and it holds equally well for reachability of discrete-time models.

Also we can see that the controllability of a system is a property that does not depend on the choice of state variables. To see that, consider the similarity transformation defined in Section 4.4.4. Then, observing that $\overline{\mathbf{A}}^i = \mathbf{T}^{-1}\mathbf{A}^i\mathbf{T}$, we have

$$\Gamma_c[\overline{\mathbf{A}}, \overline{\mathbf{B}}] = \mathbf{T}^{-1}\Gamma_c[\mathbf{A}, \mathbf{B}] \tag{4.159}$$

which implies that $\Gamma_c[\overline{\mathbf{A}}, \overline{\mathbf{B}}]$ and $\Gamma_c[\mathbf{A}, \mathbf{B}]$ have the same rank.

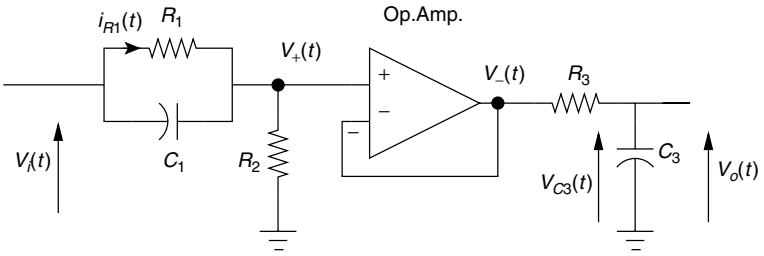


FIGURE 4.13 Electronic circuit.

The reader may wish to check that the state space models used to describe signals in Section 4.2.3 are uncontrollable. Indeed, it is always true **that any state space model where B = 0 is completely uncontrollable.**

4.6.1.4 Loss of Controllability

Lack of controllability is sometimes a structural feature. However, in some other cases, it depends on the numerical value of certain parameters. We illustrate this in the following example.

Example 4.13

Consider the electronic circuit shown in Figure 4.13.

We first build a state space model for the circuit. We choose, as state variables, $x_1(t) = i_{R1}(t)$ and $x_2(t) = v_{C3}(t)$. Using first principles on the left half of the circuit we have that

$$i_{C1} = C_1 \frac{d}{dt}(v_i - v_+), \qquad i_{R1} = \frac{v_i - v_+}{R_1}, \qquad i_{R2} = \frac{v_+}{R_2}, \qquad i_{C1} = i_{R2} - i_{R1} \tag{4.160}$$

This yields

$$\frac{di_{R1}(t)}{dt} = -\frac{(R_1 + R_2)}{C_1 R_1 R_2} i_{R1}(t) + \frac{1}{C_1 R_1 R_2} v_i(t) \tag{4.161}$$

$$v_+(t) = -R_1 i_{R1}(t) + v_i(t) \tag{4.162}$$

And, similarly, from the right half of the circuit we obtain

$$\frac{dv_{C3}(t)}{dt} = -\frac{1}{R_3 C_3} v_{C3}(t) + \frac{1}{R_3 C_3} v_-(t) \tag{4.163}$$

$$v_o(t) = v_{C3}(t) \tag{4.164}$$

The (ideal) operational amplifier ensures that $v_+(t) = v_-(t)$, so we can combine the state space models given in Equations 4.161–4.164 to obtain

$$\begin{bmatrix} \frac{di_{R1}(t)}{dt} \\ \frac{dv_{C3}(t)}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{(R_1 + R_2)}{C_1 R_1 R_2} & 0 \\ -\frac{R_1}{R_3 C_3} & -\frac{1}{R_3 C_3} \end{bmatrix} \begin{bmatrix} i_{R1}(t) \\ v_{C3}(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{C_1 R_1 R_2} \\ \frac{1}{C_3 R_3} \end{bmatrix} v_i(t) \tag{4.165}$$

$$v_o(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i_{R1}(t) \\ v_{C3}(t) \end{bmatrix} \tag{4.166}$$

The controllability matrix is then given by

$$\Gamma_c[\mathbf{A}, \mathbf{B}] = [\mathbf{B} \quad \mathbf{AB}] = \begin{bmatrix} \frac{1}{R_1 R_2 C_1} & \frac{-(R_1 + R_1)}{(R_1 R_2 C_1)^2} \\ \frac{1}{R_3 C_3} & \frac{-(R_2 C_1 + R_3 C_3)}{(R_3 C_3)^2 R_2 C_1} \end{bmatrix} \tag{4.167}$$

and

$$\det(\Gamma_c[\mathbf{A}, \mathbf{B}]) = \frac{R_2}{(R_1 R_2 R_3 C_1 C_2)^2}(-R_1 C_1 + R_3 C_3) \tag{4.168}$$

where we can observe that the system is completely controllable if, and only if, $R_1 C_1 \neq R_3 C_3$.

This issue has a very important interpretation if we analyze it from the transfer function point of view. Applying Laplace transform to Equations 4.161–4.164, the transfer function from $v_i(t)$ to $v_o(t)$ (recall that $V_+(s) = V_-(s)$) is given by

$$\frac{V_o(s)}{V_i(s)} = \frac{V_o(s)}{V_-(s)}\frac{V_+(s)}{V_i(s)} = \frac{\frac{1}{R_3 C_3}}{\left(s + \frac{1}{R_3 C_3}\right)} \cdot \frac{\left(s + \frac{1}{R_1 C_1}\right)}{\left(s + \frac{R_1 + R_2}{R_1 R_2 C_1}\right)} \tag{4.169}$$

where we can observe that the loss of complete controllability, when $R_1 C_1 = R_3 C_3$ obtained from (4.168), means that there is a **zero-pole cancellation** in the transfer function, that is, the zero from the left half of the circuit in Figure 4.13 is cancelled by the pole from the other part of the circuit. This issue will be discussed in more detail in Section 4.6.3.

4.6.1.5 Controllability Gramian

The test of controllability gives us a *yes or no* answer about the controllability of a system model. However, to conclude that a system is completely controllable says nothing about the *degree* of controllability. For **stable** systems, we can quantify the effort to control the system state through the energy involved in the input signal $\mathbf{u}(t)$ applied from $t = -\infty$ to reach the state $\mathbf{x}(0) = \mathbf{x}_0$ at $t = 0$:

$$J(\mathbf{u}) = \int_{-\infty}^{0} \|\mathbf{u}(t)\|^2 dt = \int_{-\infty}^{0} \mathbf{u}(t)^T \mathbf{u}(t)\, dt \tag{4.170}$$

It can be shown that the minimal *control energy* is

$$J(\mathbf{u}_{\text{opt}}) = \mathbf{x}_o^T \mathbf{P}^{-1} \mathbf{x}_o \tag{4.171}$$

where

$$\mathbf{P} = \int_0^{\infty} e^{\mathbf{A}t} \mathbf{B}\mathbf{B}^T e^{\mathbf{A}^T t} dt \tag{4.172}$$

The matrix $\mathbf{P}$ is called the **controllability gramian**, and it measures the controllability of the state vector $\mathbf{x}(0)$. If this matrix is *small*, it means that we need a lot of energy in the control input $\mathbf{u}(t)$ to steer the state vector to $\mathbf{x}_0$. Indeed, we can appreciate the necessary effort for each one of the state variables, making, for example $\mathbf{x}_0 = [0,\ldots, 0, 1, 0,\ldots, 0]^T$.

It is important to emphasize that the existence of the integral defined in (4.172) is guaranteed only if the eigenvalues of **A** have negative real part, that is, the system must be stable.

Also, the controllability gramian P defined in (4.172) satisfies the Lyapunov equation

$$\mathbf{AP} + \mathbf{PA}^T + \mathbf{BB}^T = 0 \tag{4.173}$$

For discrete-time systems we have the following equations for the controllability gramian:

$$\mathbf{P}_\mathbf{d} = \sum_{k=0}^{\infty} \mathbf{A}_\mathbf{d}^k \mathbf{B}_\mathbf{d} \mathbf{B}_\mathbf{d}^T (\mathbf{A}_\mathbf{d}^T)^k \tag{4.174}$$

which satisfies

$$\mathbf{A}_\mathbf{d} \mathbf{P}_\mathbf{d} \mathbf{A}_\mathbf{d}^T - \mathbf{P}_\mathbf{d} + \mathbf{B}_\mathbf{d} \mathbf{B}_\mathbf{d}^T = 0 \tag{4.175}$$

The sum defined in (4.174) is bounded if and only if the discrete-time system is stable, that is, its eigenvalues lie inside the unit disc.

Example 4.14

We can analyze the model of the Example 4.13, where the electronic circuit was described by the state space models (4.165) and (4.166). If we want to appreciate the information that we can obtain from the controllability gramian, defined in (4.172), when the model is close to losing complete controllability, we can choose suitable values of the parameters that ensure $R_1 C_1 \approx R_3 C_3$.

If we choose

$$R_1 = R_2 = R_3 = 10^3\ \Omega, \qquad C_1 = 0.9 \times 10^3\ \mu\text{F}, \qquad C_3 = 10^3\ \mu\text{F} \tag{4.176}$$

the model will be described by

$$\begin{bmatrix} \dot{i}_{R1}(t) \\ \dot{v}_{C3}(t) \end{bmatrix} = \begin{bmatrix} -\frac{20}{9} & 0 \\ -10^3 & -1 \end{bmatrix} \begin{bmatrix} i_{R1}(t) \\ v_{C3}(t) \end{bmatrix} + \begin{bmatrix} \frac{0.01}{9} \\ 1 \end{bmatrix} v_i(t) \tag{4.177}$$

$$v_o(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i_{R1}(t) \\ v_{C3}(t) \end{bmatrix} \tag{4.178}$$

If we look at the relative magnitude of the elements of **B**, we can *a priori* say that the effect of the input $u(t)$ upon the state $i_{R1}(t)$ will be much weaker than its effect upon the state $v_{C3}(t)$. To verify this we can compute the controllability gramian defined in (4.172), solving

$$\mathbf{0} = \mathbf{AP} + \mathbf{PA}^T + \mathbf{BB}^T \tag{4.179}$$

$$\mathbf{0} = \begin{bmatrix} -\frac{20}{9} & 0 \\ -10^3 & -1 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} + \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} \begin{bmatrix} -\frac{20}{9} & -10^3 \\ 0 & -1 \end{bmatrix} + \begin{bmatrix} \frac{0.01}{9} \\ 1 \end{bmatrix} \begin{bmatrix} \frac{0.01}{9} & 1 \end{bmatrix} \tag{4.180}$$

We have

$$\mathbf{P} = \begin{bmatrix} 0.28\times10^{-6} & 0.000258620 \\ 0.000258620 & 0.99999948 \end{bmatrix}, \qquad \mathbf{P}^{-1} = \begin{bmatrix} 4736624.0 & -1224.9 \\ -1224.9 & 1.3 \end{bmatrix} \tag{4.181}$$

So we can obtain the minimal control energy to steer the state $\mathbf{x}(t)$, from 0 in $t = -\infty$ to $\mathbf{x}_0$ in $t = 0$, from Equation 4.171.

$$\mathbf{x_0} = [1, 0]^T \Rightarrow J(u_{\text{opt}}) = 4736624.0 \tag{4.182}$$

$$\mathbf{x_0} = [0, 1]^T \Rightarrow J(u_{\text{opt}}) = 1.3 \tag{4.183}$$

We can thus verify that the control energy to attain $i_{R1}(0) = 1$ is six orders of magnitude greater than the necessary energy to attain $v_{C3}(0) = 1$.

Also, if we substitute the parameter values in Equation 4.169, we have that the transfer function is given by

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{s+1} \cdot \frac{s+1+\frac{1}{9}}{s+\frac{20}{95}} \tag{4.184}$$

from where we observe a **zero-pole quasi cancellation.**

The idea of gramian has been extended to include the unstable case; see [16].

4.6.1.6 Canonical Decomposition and Stabilizability

If we have a system which is not completely controllable, it can be **decomposed** into a controllable subsystem and a completely uncontrollable subsystem in the following way.

Lemma 4.1 *Consider a system having rank* $\{\Gamma_c[\mathbf{A}, \mathbf{B}]\} = k < \text{n}$. *Then there exists a similarity transformation* T *such that* $\boldsymbol{x} = \mathbf{T}^{-1}\boldsymbol{x}$,

$$\overline{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}, \qquad \overline{\mathbf{B}} = \mathbf{T}^{-1}\mathbf{B} \tag{4.185}$$

and $\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$ *have the form*

$$\overline{\mathbf{A}} = \begin{bmatrix} \overline{\mathbf{A}}_c & \overline{\mathbf{A}}_{12} \\ 0 & \overline{\mathbf{A}}_{nc} \end{bmatrix}, \qquad \overline{\mathbf{B}} = \begin{bmatrix} \overline{\mathbf{B}}_c \\ 0 \end{bmatrix} \tag{4.186}$$

where $\overline{\mathbf{A}}_c$ *has dimension k and* $(\overline{\mathbf{A}}_c, \overline{\mathbf{B}}_c)$ *is completely controllable.*

The above result tells us what states we can and what states we cannot steer to zero. To appreciate this, we express the state and output equations in the form

$$\begin{bmatrix} \dot{\overline{\mathbf{x}}}_c \\ \dot{\overline{\mathbf{x}}}_{nc} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{A}}_c & \overline{\mathbf{A}}_{12} \\ \mathbf{0} & \overline{\mathbf{A}}_{nc} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{x}}_c \\ \overline{\mathbf{x}}_{nc} \end{bmatrix} + \begin{bmatrix} \overline{\mathbf{B}}_c \\ \mathbf{0} \end{bmatrix} \mathbf{u} \tag{4.187}$$

$$\mathbf{y} = \begin{bmatrix} \overline{\mathbf{C}}_c & \overline{\mathbf{C}}_{nc} \end{bmatrix} \begin{bmatrix} \overline{\mathbf{x}}_c \\ \overline{\mathbf{x}}_{nc} \end{bmatrix} + \mathbf{D}\mathbf{u} \tag{4.188}$$

The **controllable subspace** of a state space model is composed of all states generated through every possible linear combination of the states in $\bar{\mathbf{x}}_c$. The stability of this subspace is determined by the location of the eigenvalues of $\bar{\mathbf{A}}_c$.

On the other hand, the **uncontrollable subspace** is composed of all states generated through every possible linear combination of the states in $\bar{\mathbf{x}}_{nc}$. The stability of this subspace is determined by the location of the eigenvalues of $\bar{\mathbf{A}}_{nc}$.

Hence, the input will have no effect over the uncontrollable subspace, so the best we can hope is that this uncontrollable subspace is stable, since then the state in this subspace will go to the origin. In this case the state space model is said to be **stabilizable**.

A key feature of the descriptions (4.187) and (4.188) arises from the fact that the transfer function is given by

$$\mathbf{H}(s) = \bar{\mathbf{C}}_c(s\mathbf{I} - \bar{\mathbf{A}}_c)^{-1}\bar{\mathbf{B}}_c + \mathbf{D} \tag{4.189}$$

Equation 4.189 says that the eigenvalues of the uncontrollable subspace do not belong to the set of poles of the system transfer function. This implies that there is a cancellation of all poles corresponding to the roots of $(s\mathbf{I} - \bar{\mathbf{A}}_{nc})$.

4.6.1.7 Controllability Canonical Form

Lemma 4.2 *Consider a completely reachable state space model for a SISO system. Then, there exists a similarity transformation which converts the state space model into the following* ***controllability canonical form:***

$$\mathbf{A}' = \begin{bmatrix} 0 & 0 & \dots & 0 & -\alpha_0 \\ 1 & 0 & \dots & 0 & -\alpha_1 \\ 0 & 1 & \dots & 0 & -\alpha_2 \\ \mathrm{A} & \mathrm{A} & \ddots & \mathrm{A} & \mathrm{A} \\ 0 & 0 & \dots & 1 & -\alpha_{n-1} \end{bmatrix}, \qquad \mathbf{B}' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \mathrm{A} \\ 0 \end{bmatrix} \tag{4.190}$$

where $\lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0 = \det(\lambda\mathbf{I} - \mathbf{A})$ *is the characteristic polynomial of* **A**.

Lemma 4.3 *Consider a completely controllable state space model for a SISO system. Then, there exists a similarity transformation which converts the state space model into the following* ***controller canonical form:***

$$\mathbf{A}'' = \begin{bmatrix} -\alpha_{n-1} & -\alpha_{n-2} & \dots & -\alpha_1 & -\alpha_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \mathrm{A} & \mathrm{A} & \ddots & \mathrm{A} & \mathrm{A} \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}, \qquad \mathbf{B}'' = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \mathrm{A} \\ 0 \end{bmatrix} \tag{4.191}$$

where $\lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0 = \det(\lambda\mathbf{I} - \mathbf{A})$ *is the characteristic polynomial of* **A**.

4.6.2 Observability, Reconstructibility, and Detectability

If we consider the state space model of a system, one might conjecture that if one observes the output over some time interval then this might tell us some information about the state. The associated model property is called observability (or reconstructibility).

4.6.2.1 Observability

Observability is concerned with the issue of what can be said on the state if we measure the plant output.

Example 4.15

If we look at the system defined by state space model

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}, \qquad y(t) = [1 \quad 0] \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{4.192}$$

we can see that the output $y(t)$ only is determined by $x_1(t)$, and the other state variable $x_2(t)$ has no influence on the output. So the system is not completely observable.

A formal definition is as follows:

Definition 4.3 The state $\mathbf{x}_o \neq \mathbf{0}$ is said to be **unobservable** if given $\mathbf{x}(\mathbf{0}) = \mathbf{x}_o$, and $\mathbf{u}(\mathbf{t}) = \mathbf{0}$ for $t \geq 0$, then $\mathbf{y}(t) = \mathbf{0}$ for $t \geq 0$, that is, we cannot see any effect of $\mathbf{x}_o$ on the system output.

The system is said to be **completely observable** if there exists no nonzero initial state that it is unobservable.

4.6.2.2 Reconstructibility

There is another concept, closely related to observability, called **reconstructibility.** Reconstructibility is concerned with what can be said about $\mathbf{x}(T)$, having observed the **past** values of the output, $\mathbf{y}$, for $0 \leq t \leq T$. For linear time invariant, continuous-time systems, the distinction between observability and reconstructibility is unnecessary. However, the following example illustrates that in discrete time, the two concepts are different. Consider

$$\mathbf{x}[t+1] = 0, \qquad \mathbf{x}[0] = \mathbf{x}_o \tag{4.193}$$

$$\mathbf{y}[t] = 0 \tag{4.194}$$

This system is clearly reconstructible for all $T \geq 1$, since we know for certain that $\mathbf{x}[T] = 0$ for $T \geq 1$. However, it is completely unobservable since $\mathbf{y}[t] = 0$, $\forall k$ irrespective of $\mathbf{x}_o$.

In view of the subtle difference between observability and reconstructibility, we will use the term *observability* in the sequel to cover the stronger of the two concepts.

4.6.2.3 Observability Test

A test for observability of a system is established in the following theorem.

Theorem 4.4 *Consider the linear, continuous, time-invariant, state space model where* $\mathbf{A} \in \mathrm{R}^{n \times n}$

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{4.195}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{4.196}$$

i) *The set of all unobservable states is equal to the null space of the observability matrix* $\Gamma_o[\mathbf{A}, \mathbf{C}]$ where

$$\Gamma_o[\mathbf{A}, \mathbf{C}] \triangleq \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \tag{4.197}$$

ii) *The system is completely observable if and only if* $\Gamma_o[\mathbf{A}, \mathbf{C}]$ *has* ***full column rank*** *n.*

Example 4.16

Consider the following state space model:

$$\mathbf{A} = \begin{bmatrix} -3 & -2 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \mathbf{C} = [1 \quad -1] \tag{4.198}$$

The observability matrix is given by

$$\Gamma_o[\mathbf{A}, \mathbf{C}] = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -4 & -2 \end{bmatrix} \tag{4.199}$$

Hence rank $\Gamma_o[\mathbf{A}, \mathbf{C}] = 2$, which says that the system is completely observable.

Example 4.17

If we look at the model defined in (4.192), we have

$$\mathbf{A} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{C} = [1 \quad 0] \tag{4.200}$$

The observability matrix is

$$\Gamma_o[\mathbf{A}, \mathbf{C}] = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \tag{4.201}$$

Hence rank $\Gamma_o[\mathbf{A}, \mathbf{C}] = 1 < 2$ and the system is not completely observable.

The above result also applies to discrete-time models.

The observability is a system property that does not depend on the choice of state variables. It can be proved that the rank of the matrix defined in Equation 4.197 does not change when a similarity transformation T is used (see Section 4.3.3).

4.6.2.4 Loss of Observability

Lack of observability may arise from structural system features. However, it is also possible that lack of observability occurs when certain system parameters take some specific numerical values. This is the same phenomenon, for controllability, we analyzed in Section 4.6.1. We expect that those parameters will affect the complete observability of the model in a similar way. Let us look at the following example.*

Example 4.18

Consider the electronic circuit in Figure 4.14. We can see this is the same as that in Figure 4.13 where the left and right halves were swaped, so we can use similar equations to obtain a state space model. The state variables have been chosen to be $x_1(t) = v_{C3}(t)$ and $x_2(t) = i_{R1}(t)$.

For the left half of the circuit, we have

$$\frac{dv_{C3}(t)}{dt} = -\frac{1}{R_3 C_3} v_{C3}(t) + \frac{1}{R_3 C_3} v_i(t) \tag{4.202}$$

$$v_+(t) = v_{C3}(t) \tag{4.203}$$

*Which is the *dual* of Example 4.13.

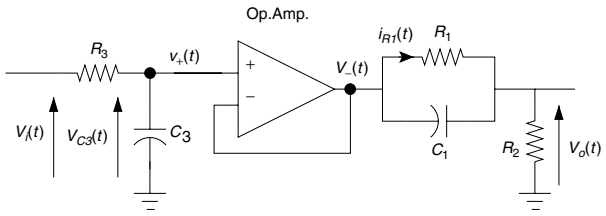


FIGURE 4.14 Electronic circuit.

And for the right half, we have

$$\frac{di_{R1}(t)}{dt} = -\left(\frac{R_1 + R_2}{C_1 R_1 R_2}\right) i_{R1}(t) + \frac{1}{C_1 R_1 R_2} v_-(t) \tag{4.204}$$

$$v_o(t) = -R_1 i_{R1}(t) + v_-(t) \tag{4.205}$$

The operational amplifier, in voltage follower connection, ensures that $v_+(t) = v_-(t)$, so we can combine the state space models given in Equations 4.202–4.205:

$$\begin{bmatrix} \frac{dv_{C3}(t)}{dt} \\ \frac{di_{R1}(t)}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{1}{R_3 C_3} & 0 \\ \frac{1}{C_1 R_1 R_2} & -\frac{R_1 + R_2}{C_1 R_1 R_2} \end{bmatrix} \begin{bmatrix} v_{C3}(t) \\ i_{R1}(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{R_3 C_3} \\ 0 \end{bmatrix} v_i(t) \tag{4.206}$$

$$v_o(t) = [1 \; -R_1] \begin{bmatrix} v_{C3}(t) \\ i_{R1}(t) \end{bmatrix} \tag{4.207}$$

The observability matrix is given by

$$\Gamma_c[\mathbf{C}, \mathbf{A}] = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} = \begin{bmatrix} 1 & -R_1 \\ -\frac{1}{R_3 C_3} - \frac{1}{R_2 C_1} & \frac{R_1 + R_2}{R_2 C_1} \end{bmatrix} \tag{4.208}$$

To determine the complete observability, or otherwise, we need to compute the matrix determinant

$$\det(\Gamma_c[\mathbf{C}, \mathbf{A}]) = \frac{1}{R_3 C_3 C_1}(-R_1 C_1 + R_3 C_3) \tag{4.209}$$

from where we conclude that the model system is completely observable if and only if, $R_1 C_1 \neq R_3 C_3$, which is the same condition we obtained in Example 4.13.

Applying Laplace transform to Equations 4.204–4.203 we obtain the transfer function from $V_i(s)$ to $V_o(s)$:

$$\frac{V_o(s)}{V_i(s)} = \frac{V_+(s)}{V_i(s)} \frac{V_o(s)}{V_-(s)} = \frac{s + \frac{1}{R_1 C_1}}{s + \frac{R_1 + R_2}{R_1 R_2 C_1}} \cdot \frac{\frac{1}{R_3 C_3}}{s + \frac{1}{R_3 C_3}} \tag{4.210}$$

The condition $R_1C_1 = R_3C_3$ produces the loss of complete observability, leading to a **pole-zero cancellation** in the model transfer function, i.e., the pole from the left half of the circuit in Figure 4.14 is cancelled by the zero from the right half. There is subtle difference between the transfer functions in (4.210) and (4.169). The final result is the same, but the order the cancellation is different in each case. The **zero-pole cancellation** is connected to the loss of complete observability and the **pole-zero cancellation** is connected to the loss of complete controllability. These issues will be discussed in more detail in Section 4.6.3.

4.6.2.5 Observability Gramian

The observability test in Theorem 4.4 answers *yes* or *no* to the question about completely observability of a model. However, sometimes we are interested in the *degree* of observability for a particular model. So we can quantify the energy of the output signal $\mathbf{y}(t)$, when there is no input ($\mathbf{u}(t) = \mathbf{0}$) and the state is $\mathbf{x}(0) = \mathbf{x}_0$ at $t = 0$

$$E(\mathbf{x}_0) = \int_0^\infty \mathrm{i}\mathbf{y}(t)\mathrm{i}^2 dt = \int_0^\infty \mathbf{y}(t)^T\mathbf{y}(t)dt \tag{4.211}$$

It can be proved that the *ouput energy* is

$$E(\mathbf{x}_0) = \int_0^\infty \mathrm{i}\mathbf{y}(t)\mathrm{i}^2 dt = \mathbf{x}_0^T\mathbf{Q}\mathbf{x}_0 \tag{4.212}$$

where

$$\mathbf{Q} = \int_0^\infty e^{\mathbf{A}^T t}\mathbf{C}^T\mathbf{C}\, e^{\mathbf{A}t}dt \tag{4.213}$$

The matrix $\mathbf{Q}$ is called **observability gramian**, and it measures the observability of the state vector $\mathbf{x}(0)$. If this matrix is *small*, it means that we have a weak contribution of the initial state $\mathbf{x}_0$ in the energy of the output $\mathbf{y}(t)$. Indeed, we can appreciate the effect of each one of the state variables taking, for example, $\mathbf{x}_0 = [0,\ldots,0,1,0,\ldots,0]^T$.

Note that the existence of the integral defined in (4.213) is guaranteed if and only if the system is stable, i.e., if and only if the eigenvalues of $\mathbf{A}$ have negative real part.

Also, the observability gramian $\mathbf{Q}$ defined in (4.213) satisfies the Lyapunov equation

$$\mathbf{A}^T\mathbf{Q} + \mathbf{Q}\mathbf{A} + \mathbf{C}^T\mathbf{C} = 0 \tag{4.214}$$

For stable discrete-time systems, the controllability gramian is defined by

$$\mathbf{Q}_\mathrm{d} = \sum_{k=0}^{\infty} (\mathbf{A}_\mathrm{d}^T)^k\mathbf{C}_\mathrm{d}^T\mathbf{C}_\mathrm{d}\mathbf{A}_\mathrm{d}^k \tag{4.215}$$

which satisfies

$$\mathbf{A}_\mathrm{d}^T\mathbf{Q}_\mathrm{d}\mathbf{A}_\mathrm{d} - \mathbf{Q}_\mathrm{d} + \mathbf{C}_\mathrm{d}^T\mathbf{C}_\mathrm{d} = 0 \tag{4.216}$$

Example 4.19

We will use the model of Example 4.18, described by the state space models (4.206) and (4.207), to appreciate the utility of the observability gramian (4.213), especially when the model is close to losing complete observability, i.e., when $R_1C_1 \approx R_3C_3$.

Assuming the same component values as in Example 4.14 for R_1, R_2, R_3, C_1, and C_3 we have

$$\begin{bmatrix} v_{C3}(t) \\ \dot{i}_{R1}(t) \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 10^{-3} & -\frac{20}{9} \end{bmatrix} \begin{bmatrix} v_{C3}(t) \\ i_{R1}(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} v_i(t) \tag{4.217}$$

$$v_o(t) = [1 \quad -10^3] \begin{bmatrix} v_{C3}(t) \\ i_{R1}(t) \end{bmatrix} \tag{4.218}$$

If we look at the relative magnitude of the components of C matrix, we can foretell *a priori* that the output $v_o(t)$ will be mainly determined by state $i_{R1}(t)$. To verify this we compute the observability gramian defined in (4.172), solving

$$\mathbf{0} = \mathbf{A}^T\mathbf{Q} + \mathbf{Q}\mathbf{A} + \mathbf{C}^T\mathbf{C} \tag{4.219}$$

$$\mathbf{0} = \begin{bmatrix} -1 & 10^{-3} \\ 0 & -\frac{20}{9} \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} + \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 10^{-3} & -\frac{20}{9} \end{bmatrix} + \begin{bmatrix} 1 \\ -10^3 \end{bmatrix} [1 \quad 10^{-3}] \tag{4.220}$$

We have

$$\mathbf{Q} = \begin{bmatrix} 0.57 & 69.83 \\ 69.83 & 225000 \end{bmatrix} \tag{4.221}$$

From there we can compute the contribution of each state to the total energy in the output. Doing this, we verify that the state variable $i_{R1}(t)$ has an effect over the output greater than the effect of $v_{C3}(t)$, as defined in Equation 4.212:

$$\mathbf{x}_0 = [1, 0]^T \Rightarrow E(\mathbf{x}_0) = 0.57 \tag{4.222}$$

$$\mathbf{x}_0 = [0, 1]^T \Rightarrow E(\mathbf{x}_0) = 225000 \tag{4.223}$$

The transfer function is

$$\frac{V_o(s)}{V_i(s)} = \frac{s + 1 + \frac{1}{9}}{s + \frac{20}{9}} \cdot \frac{1}{s + 1} \tag{4.224}$$

We observe that there is a pole-zero **quasi-cancellation**.

4.6.2.6 Duality Principle

We observe a remarkable similarity between the results in Theorem 4.3 and in Theorem 4.4, and also for the definitions of the gramians (4.172) and (4.213). This is known as the **duality principle**, and it can be formalized as follows:

Theorem 4.5 (Duality) *Consider a state space model described by the 4-tuple* (**A**, **B**, **C**, **D**). *Then the system is completely controllable if and only if the dual system* ($\mathbf{A}^T$, $\mathbf{C}^T$, $\mathbf{B}^T$, $\mathbf{D}^T$) *is completely observable.*

4.6.2.7 Canonical Decomposition and Detectability

The above theorem can often be used to go from a result on controllability to one on observability and vice versa. The *dual* of Lemma 4.1 is:

Lemma 4.4 *If rank* $\{\Gamma_o[\mathbf{A}, \mathbf{C}]\} = k < n$, *there exists a similarity transformation* $\mathbf{T}$ *such that with* $\bar{\mathbf{x}} = \mathbf{T}^{-1}\mathbf{x}$, $\bar{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$, $\bar{\mathbf{C}} = \mathbf{C}\mathbf{T}$, *then* $\bar{\mathbf{C}}$ *and* $\bar{\mathbf{A}}$ *take the form*

$$\bar{\mathbf{A}} = \begin{bmatrix} \bar{\mathbf{A}}_o & 0 \\ \bar{\mathbf{A}}_{21} & \bar{\mathbf{A}}_{no} \end{bmatrix}, \qquad \bar{\mathbf{C}} = [\bar{\mathbf{C}}_o \quad 0] \tag{4.225}$$

where $\bar{\mathbf{A}}_o$ *has dimension k and the pair* $(\bar{\mathbf{C}}_o, \bar{\mathbf{A}}_o)$ *is completely observable.*

This result has a relevance similar to that of the controllability property and the associated decomposition. To appreciate this, we apply the dual of Lemma 4.1 to express the (transformed) state and output equations in partitioned form as

$$\begin{bmatrix} \dot{\bar{\mathbf{x}}}_o(t) \\ \dot{\bar{\mathbf{x}}}_{no}(t) \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{A}}_o & 0 \\ \bar{\mathbf{A}}_{21} & \bar{\mathbf{A}}_{no} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_o(t) \\ \bar{\mathbf{x}}_{no}(t) \end{bmatrix} + \begin{bmatrix} \bar{\mathbf{B}}_o \\ \bar{\mathbf{B}}_{no} \end{bmatrix} \mathbf{u}(t) \tag{4.226}$$

$$\mathbf{y}(t) = \begin{bmatrix} \bar{\mathbf{C}}_o & 0 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_o(t) \\ \bar{\mathbf{x}}_{no}(t) \end{bmatrix} + \mathbf{D}\mathbf{u}(t) \tag{4.227}$$

The above description reveals why one can be in trouble when trying to control a system using only the system output. The output has no information on the state $\bar{\mathbf{x}}_{no}$.

The **observable subspace** of a model is the space composed of all states generated through every possible linear combination of the states in $\bar{\mathbf{x}}_o$. The stability of this subspace is determined by the location of the eigenvalues of $\bar{\mathbf{A}}_o$.

The **unobservable subspace** of a model is the space composed of all states generated through every possible linear combination of the states in $\bar{\mathbf{x}}_{no}$. The stability of this subspace is determined by the location of the eigenvalues of $\bar{\mathbf{A}}_{no}$.

If the unobservable subspace is stable we say that the system is **detectable.**

A key feature of the descriptions (4.226) and (4.227) arises from the fact that the transfer function is given by

$$\mathbf{H}(s) = \bar{\mathbf{C}}_0(s\mathbf{I} - \bar{\mathbf{A}}_o)^{-1}\bar{\mathbf{B}}_o + \mathbf{D} \tag{4.228}$$

Equation (4.228) says that the eigenvalues of the unobservable subspace do not belong to the set of poles of the system transfer function. This implies that there is a cancellation of all poles corresponding to the roots of $(s\mathbf{I} - \bar{\mathbf{A}}_{no})$.

4.6.2.8 Observability Canonical Form

There are also duals of the canonical forms given in Lemmas 4.2 and 4.3. For example, the dual of Lemma 4.3 is:

Lemma 4.5 *Consider a completely observable SISO system. Then there exists a similarity transformation that converts the model to the* ***observer canonical form****:*

$$\dot{x}(t) = \begin{bmatrix} -\alpha_{n-1} & 1 & & \\ \mathrm{A} & & \ddots & \\ \mathrm{A} & & & 1 \\ -\alpha_0 & 0 & & 0 \end{bmatrix} x(t) + \begin{bmatrix} b_{n-1} \\ \mathrm{A} \\ \mathrm{A} \\ b_0 \end{bmatrix} u(t) \tag{4.229}$$

$$y(t) = [1 \quad 0 \quad \cdots \quad 0]x(t) + \mathbf{D}u(t) \tag{4.230}$$

4.6.3 Canonical Decomposition

Further insight into the structure of linear dynamical systems is obtained by considering those systems which are only partially observable or controllable. These systems can be separated into completely observable and completely controllable systems.

The two results of Lemmas 4.1 and 4.4 can be combined for those systems, which are neither completely observable nor completely controllable. We can see it as follows.

Theorem 4.6 (Canonical Decomposition Theorem) *Consider a system described in state space form. Then, there always exists a similarity transformation* $\mathbf{T}$ *such that the transformed model for* $\boldsymbol{x} = \mathbf{T}^{-1}\boldsymbol{x}$ *takes the form*

$$\overline{\mathbf{A}} = \begin{bmatrix} \overline{\mathbf{A}}_{co} & \mathbf{0} & \overline{\mathbf{A}}_{13} & \mathbf{0} \\ \overline{\mathbf{A}}_{21} & \overline{\mathbf{A}}_{22} & \overline{\mathbf{A}}_{23} & \overline{\mathbf{A}}_{24} \\ \mathbf{0} & \mathbf{0} & \overline{\mathbf{A}}_{33} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \overline{\mathbf{A}}_{34} & \overline{\mathbf{A}}_{44} \end{bmatrix}, \quad \overline{\mathbf{B}} = \begin{bmatrix} \overline{\mathbf{B}}_1 \\ \overline{\mathbf{B}}_2 \\ 0 \\ 0 \end{bmatrix}, \quad \overline{\mathbf{C}} = [\overline{\mathbf{C}}_1 \;\; \mathbf{0} \;\; \overline{\mathbf{C}}_2 \;\; \mathbf{0}] \tag{4.231}$$

where

i) *The subsystem* $[\overline{\mathbf{A}}_{co}, \overline{\mathbf{B}}_1, \overline{\mathbf{C}}_1]$ *is both completely controllable and completely observable and has the same transfer function as the original system (see Lemma 4.6).*

ii) *The subsystem*

$$\begin{bmatrix} \overline{\mathbf{A}}_{co} & \mathbf{0} \\ \overline{\mathbf{A}}_{21} & \overline{\mathbf{A}}_{22} \end{bmatrix}, \quad \begin{bmatrix} \overline{\mathbf{B}}_1 \\ \overline{\mathbf{B}}_2 \end{bmatrix}, \quad [\overline{\mathbf{C}}_1 \;\; \mathbf{0}] \tag{4.232}$$

is completely controllable.

iii) *The subsystem*

$$\begin{bmatrix} \overline{\mathbf{A}}_{co} & \overline{\mathbf{A}}_{13} \\ 0 & \overline{\mathbf{A}}_{33} \end{bmatrix}, \quad \begin{bmatrix} \overline{\mathbf{B}}_1 \\ 0 \end{bmatrix}, \quad [\overline{\mathbf{C}}_1 \;\; \overline{\mathbf{C}}_2] \tag{4.233}$$

is completely observable.

The canonical decomposition described in Theorem 4.6 leads to an important consequence for the transfer function of the model, which will take only the completely observable and completely controllable subspace.

Lemma 4.6 *Consider the transfer function matrix* $\mathbf{H}(s)$ *given by*

$$Y(s) = \mathbf{H}(s)U(s) \tag{4.234}$$

Then

$$\mathbf{H} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \overline{\mathbf{C}}_1(s\mathbf{I} - \overline{\mathbf{A}}_{co})^{-1}\overline{\mathbf{B}}_1 + \mathbf{D} \tag{4.235}$$

where $\overline{\mathbf{C}}_1$, $\overline{\mathbf{A}}_{co}$, *and* $\overline{\mathbf{B}}_1$ *are as in Equation 4.231. This state description is a minimal realization of the transfer function.*

If $\mathbf{M}$ is any square matrix and we denote by $\Lambda\{\mathbf{M}\}$ the set of eigenvalues of $\mathbf{M}$, then

$$\Lambda\{\bar{\mathbf{A}}\} = \Lambda\{\bar{\mathbf{A}}_{co}\} \cup \Lambda\{\bar{\mathbf{A}}_{22}\} \cup \Lambda\{\bar{\mathbf{A}}_{33}\} \cup \Lambda\{\bar{\mathbf{A}}_{44}\} \tag{4.236}$$

where

$\Lambda\{\bar{\mathbf{A}}\}$ = eigenvalues of the system,
$\Lambda\{\bar{\mathbf{A}}_{co}\}$ = eigenvalues of the controllable and observable subsystem,
$\Lambda\{\bar{\mathbf{A}}_{22}\}$ = eigenvalues of the controllable but unobservable subsystem,
$\Lambda\{\bar{\mathbf{A}}_{33}\}$ = eigenvalues of the uncontrollable but observable subsystem,
$\Lambda\{\bar{\mathbf{A}}_{44}\}$ = eigenvalues of the uncontrollable and unobservable subsystem.

We observe that controllability for a given system depends on the structure of the input ports, that is, where, in the system, the manipulable inputs are applied. Thus, the states of a given subsystem may be uncontrollable for a given input, but completely controllable for another. This distinction is of fundamental importance in control system design since not all plant inputs can be manipulated (consider, for example, disturbances) and, therefore, cannot be used to steer the plant to reach certain states.

Similarly, the observability property depends on which outputs are being considered. Certain states may be unobservable from a given output, but they may be completely observable from some other output. This also has a significant impact on output feedback control systems, since some states may not appear in the plant output being measured and feeded back. However, they may appear in crucial internal variables and thus be important to the control problem.

4.6.4 PBH Test

An alternative test for controllability and observability is provided by the following lemma known as **PBH** test.

Lemma 4.7 *Consider a state space model* $(\mathbf{A}, \mathbf{B}, \mathbf{C})$. *Then*

(i) *The system is not completely observable if and only if there exists a nonzero vector* $\mathbf{x} \in \mathbb{C}^n$ *and a scalar* $\lambda \in \mathbb{C}$ *such that*

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \quad \mathbf{C}\mathbf{x} = 0 \tag{4.237}$$

(ii) *The system is not completely controllable if and only if there exists a nonzero vector* $\mathbf{x} \in \mathbb{C}^n$ *and a scalar* $\lambda \in \mathbb{C}$ *such that*

$$\mathbf{x}^T\mathbf{A} = \lambda\mathbf{x}^T, \quad \mathbf{x}^T\mathbf{B} = 0 \tag{4.238}$$

4.7 State Observers

4.7.1 Basic Concepts

When the state variables have to be measured for monitoring, implementing control systems, or other purposes, there are hard technical and economical issues to face. Observers are a way to estimate the state variables based upon a system model, measurements of the plant output $\mathbf{y}(t)$, and measurements of the plant input $\mathbf{u}(t)$. This problem is a generalization of that of indirectly measuring a system variable using a system model and the measurement of some other easier-to-measure variable.

4.7.2 Observer Dynamics

Assume that the system has a state space model given by (4.42) and (4.43) with $\mathbf{D} = \mathbf{0}$ (a strictly proper system has been assumed). Then, the general structure of a classic observer for the system state is as shown in Figure 4.15, where the matrix $\mathbf{J}$ is the **observer gain**.

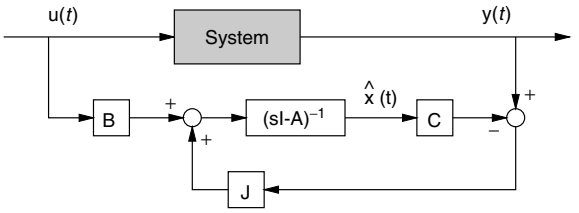


FIGURE 4.15 Classic state observer.

Therefore, the observer equation is

$$\frac{d\hat{\mathbf{x}}(t)}{dt} = \mathbf{A}\hat{\mathbf{x}}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{J}(\mathbf{y}(t) - \mathbf{C}\hat{\mathbf{x}}(t)) \tag{4.239}$$

An obvious question is: if we know an exact system model and the system input, why do we need to feed the system output? The answer is that **we need the output measurement since we do not know the system initial state**. This can be appreciated from the equation for the state estimation error, $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) - \hat{\mathbf{x}}(t)$. That equation can be obtained subtracting (4.239) from (4.42). This leads to

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = (\mathbf{A} - \mathbf{J}\mathbf{C})\tilde{\mathbf{x}}(t) \tag{4.240}$$

From (4.240) we observe that the estimation error will converge to zero for a nonzero initial error if and only if all the eigenvalues of the matrix $\mathbf{A} - \mathbf{J}\mathbf{C}$ have negative real parts, that is, if the **observer polynomial** $E(s) = \det(sI - \mathbf{A} + \mathbf{J}\mathbf{C})$ is strictly Hurwitz.

Discussion

- Equation (4.240) is valid only if the model is a perfect representation of the system under study. Modelling errors will impact the observer. This will normally lead to nonzero state estimation errors.
- If the pair $(\mathbf{A}, \mathbf{C})$ is completely observable, then the eigenvalues of $\mathbf{A} - \mathbf{J}\mathbf{C}$ can be arbitrarily located (in the stability region). Thus, the speed of the estimation convergence is a designer's choice. Those eigenvalues are known as the **observer poles**.
- If the pair $(\mathbf{A}, \mathbf{C})$ is detectable, then the observer will yield zero steady state error asymptotically, although not all the eigenvalues of $\mathbf{A} - \mathbf{J}\mathbf{C}$ can be placed at will.
- If the system is not completely observable, and the unobservable subspace contains unstable modes, then the observer will never converge.

To illustrate the observer techniques we refer to Example 4.5.

Example 4.20

Assume that we want the observer poles for the state model in Example 4.5 to be located at $s = -4$, $s = -6$, and $s = -8$. We can then compute the observer gain, $\mathbf{J}$, using a software such as MATLAB. This yields

$$J = [-4.5247 \quad -7.5617 \quad -4.1543]^T \tag{4.241}$$

To appreciate the observer dynamics, assume that the initial system state is $\mathbf{x}(0) = [-1 \quad 2 \quad 1]^T$ and that the system input is a square wave of amplitude 1, and frequency equal to 1 rad/s. The observer is initialized with $\hat{\mathbf{x}}(0) = 0$. Then the norm of the estimation error, $\|\hat{\mathbf{x}}(t)\|$, evolves as shown in Figure 4.16. It is

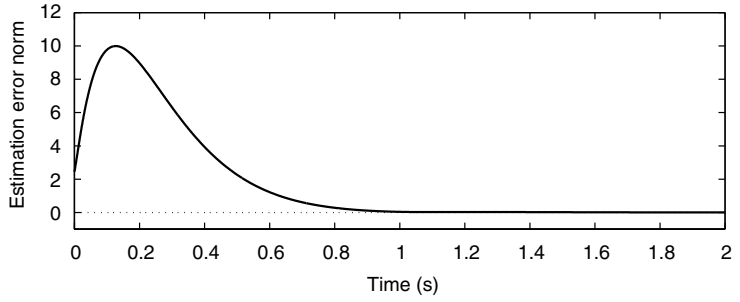


FIGURE 4.16 State estimation error.

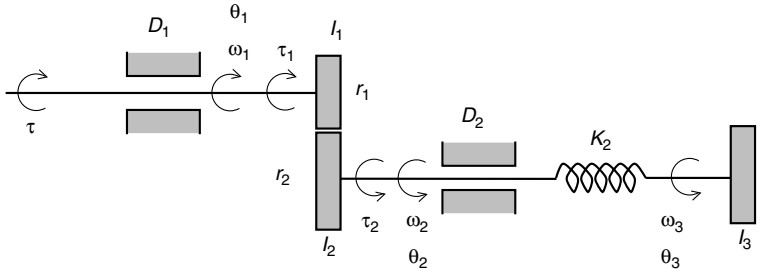


FIGURE 4.17 Rotational system.

important to point out that, in this example, the plant is unstable. This means that the state and the state estimation grow unbounded. However, under the assumption of perfect modelling, the estimation error converges to zero.

To gain physical insight into the observer philosophy, we consider the following application.

Example 4.21

Figure 4.17 shows the schematics of a rotational system driven by a torque $\tau(t)$. The system power is transmitted through a gear system built with two wheels with radii r_1 and r_2 and inertias I_1 and I_2, respectively. The rotation of both shafts is damped by viscous friction with coefficients D_1 and D_2, and a significant torsional spring in shaft 2 has also been modelled. The system load is modelled as an inertia I_3. We want to estimate the load speed ω_3 based on the measurement of the speed in shaft 1, ω_1.

We first need to build a state space model. To do that we choose a minimum set of system variables, which quantify the energy stored in the system. The system has four components able to store energy: three inertias and a spring. Nevertheless, the energy stored in I_1 and I_2 can be computed either from ω_1 or from ω_2, that is, we need only one of these speeds, since they satisfy

$$\frac{\omega_1(t)}{\omega_2(t)} = \frac{r_2}{r_1} \quad \text{and} \quad \tau_1(t)\omega_1(t) = \tau_2(t)\omega_2(t) \tag{4.242}$$

Thus, a physically oriented choice of state variables is

$$x_1(t) = \omega_1(t) \tag{4.243}$$

$$x_2(t) = \theta_2(t) - \theta_3(t) \tag{4.244}$$

$$x_3(t) = \omega_3(t) \tag{4.245}$$

From first principles we have

$$\tau(t) = D_1\omega_1(t) + I_1\frac{d\omega_1(t)}{dt} + \tau_1(t) \tag{4.246}$$

$$\tau(t) = \frac{r_2}{r_1}\tau_1(t) = D_2\omega_2(t) + I_2\frac{d\omega_2(t)}{dt} + K_2(\theta_2(t) - \theta_3(t)) \tag{4.247}$$

$$0 = K_2(\theta_3(t) - \theta_2(t)) + I_3\frac{d\omega_3(t)}{dt} \tag{4.248}$$

Since we have chosen $\omega_1(t)$ as the measurable system variable, we finally obtain

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{\begin{bmatrix} -\frac{r_1^2D_2 + r_2^2D_1}{r_1^2I_2 + r_2^2I_1} & -\frac{r_1r_2K_2}{r_1^2I_2 + r_2^2I_1} & 0 \\ \frac{r_1}{r_2} & 0 & -1 \\ 0 & \frac{K_2}{I_3} & 0 \end{bmatrix}}_{\mathbf{A}} \mathbf{x}(t) + \underbrace{\begin{bmatrix} \frac{r_2^2}{r_1^2I_2 + r_2^2I_1} \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{B}} \tau(t) \tag{4.249}$$

$$\omega_1(t) = \underbrace{\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}}_{\mathbf{C}} \mathbf{x}(t) \tag{4.250}$$

To evaluate the observability properties of this system, numerical values for the parameters are chosen as follows:

$$r_1 = 0.25 \text{ m}, \qquad r_2 = r_3 = 0.50 \text{ m}, \qquad D_1 = D_2 = 10 \text{ Nms/rad} \tag{4.251}$$

$$K_2 = 30 \text{ Nm/rad}, \qquad I_1 = 2.39 \text{ Nms}^2\text{/rad}, \qquad I_2 = I_3 = 38.29 \text{ Nms}^2\text{/rad} \tag{4.252}$$

With these values we have that

$$\mathbf{A} = \begin{bmatrix} -1.045 & -1.254 & 0 \\ 0.5 & 0 & -1 \\ 0 & 0.784 & 0 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0.084 \\ 0 \\ 0 \end{bmatrix} \tag{4.253}$$

We next use the test presented in Section 4.6.2. This yields

$$\Gamma_o = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \end{bmatrix} = \begin{bmatrix} 1.0000 & 0 & 0 \\ -1.0450 & -1.2540 & 0 \\ 0.4650 & 1.3104 & 1.2540 \end{bmatrix} \tag{4.254}$$

From this expression we see that Γ_o is a full rank matrix. Thus, the system state is completely observable from $\omega_1(t)$.

Once we have a state estimate, $\hat{\mathbf{x}}(t)$, an estimate, $\omega_3(t)$ for ω_3, is obtained from

$$\omega_3(t) = \underbrace{[0 \quad 0 \quad 1]}_{\mathbf{K}_3^T} \hat{x}(t) \tag{4.255}$$

where $\hat{\omega}_3(t)$ can be obtained from (4.239). This yields

$$\frac{d\hat{\omega}_3(t)}{dt} = \mathbf{K}_3^T \frac{d\hat{\mathbf{x}}(t)}{dt} = \mathbf{K}_3^T(\mathbf{A} - \mathbf{JC})\hat{\mathbf{x}}(t) + \underbrace{\mathbf{K}_3^T\mathbf{B}}_{0} \tau(t) + \mathbf{K}_3^T\mathbf{J}\omega_1(t) \tag{4.256}$$

4.7.3 Observers and Measurement Noise

In the theory above we have assumed that both the system input, $\mathbf{u}(t)$, and the system output, $\mathbf{y}(t)$, are available with no errors. This assumption is usually correct with regard to $\mathbf{u}(t)$, since the same equipment generating $\mathbf{u}(t)$ is normally used to estimate the state. However, that assumption is not usually valid with respect to $\mathbf{y}(t)$, since the measurement of this variable is normally corrupted with noise. To analyze the effect of this error, let us denote by $\mathbf{y}_m(t)$ the noisy measurement, that is, $\mathbf{y}_m(t) = \mathbf{y}(t) + \mathbf{v}(t)$, where $\mathbf{v}(t)$ is the additive measurement noise. Therefore, the state estimation error satisfies

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = (\mathbf{A} - \mathbf{JC})\tilde{\mathbf{x}}(t) + \mathbf{J}\mathbf{v}(t) \tag{4.257}$$

We then have that

$$\tilde{X}(s) = (s\mathbf{I} - \mathbf{A} + \mathbf{JC})^{-1}\tilde{\mathbf{x}}(0) + (s\mathbf{I} - \mathbf{A} + \mathbf{JC})^{-1}\mathbf{J}V(s) \tag{4.258}$$

Hence, the error is small if the transfer function $(s\mathbf{I} - \mathbf{A} + \mathbf{JC})^{-1}\mathbf{J}$ filters out the noise. Consider the following example.

Example 4.22

A system has a state space model given by

$$\mathbf{A} = \begin{bmatrix} -2 & 1 \\ 1 & -3 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}, \qquad \mathbf{C} = [1 \quad -1], \qquad D = 0 \tag{4.259}$$

Assume that we want to estimate a system variable $z(t) = \gamma^T\mathbf{x}(t)$, where $\gamma^T = [1 \quad 1]$. Then, a suitable observer-based estimate is $\hat{z}(t)$, which is given by

$$\hat{z}(t) = \gamma^T\hat{\mathbf{x}}(t) \tag{4.260}$$

Then, the noise term in the estimation of $z(t)$ is $z_v(t)$, whose Laplace transform satisfies

$$Z_v(s) = H_v(s)V(s), \quad \text{where} \quad H_v(s) = \gamma^T(s\mathbf{I} - \mathbf{A} + \mathbf{JC})^{-1}\mathbf{J} \tag{4.261}$$

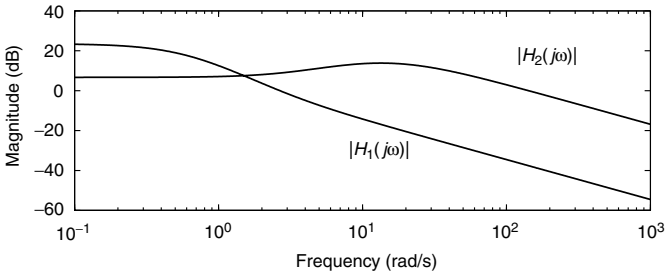


FIGURE 4.18 Observer filtering characteristics.

We next consider two different choices for the observer polynomial $E(s)$. They are

$$E_1(s) = (s+0.5)(s+0.75) \quad \text{and} \quad E_2(s) = (s+10)(s+20) \tag{4.262}$$

The reader can appreciate that the resulting observers will have very different speeds, the first observer being much slower than the second one.

With those choices we compute the observer gains, J_1 and J_2, and the corresponding filter functions

$$H_1(s) = \gamma^T(s\mathbf{I} - \mathbf{A} + \mathbf{J}_1\mathbf{C})^{-1}\mathbf{J}_1 = \frac{1.875s + 5.625}{s^2 + 1.25s + 0.375} \tag{4.263}$$

$$H_2(s) = \gamma^T(s\mathbf{I} - \mathbf{A} + \mathbf{J}_2\mathbf{C})^{-1}\mathbf{J}_2 = \frac{144s + 432}{s^2 + 30s + 200} \tag{4.264}$$

To compare both cases we compute and plot the frequency response of each filter. The result is shown in Figure 4.18.

From Figure 4.18 we observe that for a high frequency noise, the slowest filter is more immune to noise than the fast filter.

The above case exemplifies the trade-off between observer speed and noise immunity. A systematic way to face this dilemma is to use an optimal filter theory, such as Kalman–Bucy filtering. The interested reader is referred to [2].

4.8 State Feedback

4.8.1 Basic Concepts

When all the system states can be measured, and the system is completely reachable (in the sense explained in Section 4.6.1), we can control the system using state feedback to achieve full command of the loop dynamics. This idea is captured in Figure 4.19.

Figure 4.19 shows the most basic form of state feedback: the plant input has a component that is proportional to the state (the other component is an external signal $\bar{r}(t)$).

State feedback is a very simple, almost naive idea. A careful analysis shows that this idea has some shortcomings and potentially dangerous features, such as

- It requires as many sensors as state variables. This is not only very expensive but also, in some cases, its implementation may become impossible.
- Each state measurement is a source of error because of its limited accuracy.
- Each measurement introduces noise, which has deleterious effect on the control system performance.

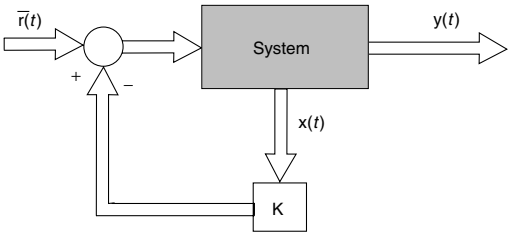


FIGURE 4.19 State feedback.

- The correct overall performance relies on the correct functioning of a complex set of equipments. This poses several questions regarding performance degradation and system integrity.

In spite of these weak points, state feedback is by itself a powerful concept, since it works as a basis for more sophisticated and robust control schemes. The key reason for this is that any linear controller can be explained as the combination of a state observer and state feedback.

4.8.2 Feedback Dynamics

Assume that the system to be controlled has a transfer function $\mathbf{H}(s)$ and a state space representation given by (4.42) and (4.43), with $\mathbf{D} = \mathbf{0}$. If the plant input is generated according to

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t) + \bar{\mathbf{r}}(t) \tag{4.265}$$

then the state space representation for the complete control loop is given by

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}(-\mathbf{K}\mathbf{x}(t) + \bar{\mathbf{r}}(t)) \tag{4.266}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \tag{4.267}$$

It can be shown that the relationship between $\bar{\mathbf{R}}(s)$ and $\mathbf{Y}(s)$ is given by

$$\mathbf{Y}(s) = \underbrace{\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}}_{\mathbf{H}(s)} (\mathbf{I} + \mathbf{K}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B})^{-1}\bar{\mathbf{R}}(s) \tag{4.268}$$

This shows that the state feedback loop preserves the system zeros and shifts the poles to the roots of $\det(s\mathbf{I} - \mathbf{A} + \mathbf{B}\mathbf{K})$.

4.8.3 Optimal State Feedback—The Optimal Regulator

Consider a linear time invariant system having a state space representation given by (4.42) and (4.43), with $\mathbf{D} = \mathbf{0}$, subject to the initial state $\mathbf{x}(0) = \mathbf{x}_o$.

Assume that the control objective is to steer the plant from the the initial state, $\mathbf{x}_o$, to the smallest possible value as soon as possible in the interval $[0, t_f]$. We additionally require that the steering process does not demand too much control effort. Then, the optimal regulator problem is defined as the problem of finding an optimal control $\mathbf{u}(t)$ over the interval $[0, t_f]$ such that a quadratic cost function is minimized. This cost function is chosen as

$$J_u(\mathbf{x}_o) = \int_0^{t_f} [\mathbf{x}(t)^T\mathbf{Q}x(t) + \mathbf{u}(t)^T\mathbf{R}\mathbf{u}(t)]\,dt + \mathbf{x}(t_f)^T\mathbf{Q}_f\mathbf{x}(t_f) \tag{4.269}$$

where $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\mathbf{Q}_f \in \mathbb{R}^{n\times n}$ are symmetric nonnegative definite matrices and $\mathbf{R} \in \mathbb{R}^{m\times m}$ is a symmetric positive definite matrix. The requirements on the weighting matrices are set so that the cost function makes sense. For instance, if $\mathbf{Q}$ is allowed to be negative, then the *optimal* cost could even be negative while the state could grow unbounded in magnitude. Also, if we allow $\mathbf{R}$ to have eigenvalues at the origin (that is, $\mathbf{R}$ is allowed to be a nonnegative definite matrix, instead of requiring it to be a strictly positive definite matrix) then the control $\mathbf{u}(t)$ could also grow unbounded (in the directions of the associated eigenvectors) without that situation being revealed by the cost function.

A time invariant linear control law is asymptotically obtained when $t_f \to \infty$. Under this condition, the optimal control law is given by

$$\mathbf{u}^{\mathrm{o}}(t) = -\mathbf{K}^{\mathrm{o}} x(t) \tag{4.270}$$

with

$$\mathbf{K}^{\mathrm{o}} = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}_\infty \tag{4.271}$$

and where $\mathbf{P}_\infty$ is the only nonnegative solution of the algebraic Riccati equation

$$\mathbf{0} = \mathbf{Q} - \mathbf{P}_\infty\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}_\infty + \mathbf{P}_\infty\mathbf{A} + \mathbf{A}^T\mathbf{P}_\infty \tag{4.272}$$

For this solution to exist, it is necessary that certain technical conditions are satisfied (for a detailed discussion of these issues see, for instance, [5]).

Discussion

- The solution for the LQR problem minimizes the cost function (4.269) and, when $t_f \to \infty$, always stabilizes the plant.
- A key issue is how to choose the weighting matrices $\mathbf{Q}$ and $\mathbf{R}$. A frequent choice for $\mathbf{Q}$ is $\mathbf{Q} = \mathbf{C}^T\mathbf{C}$. With this choice, the magnitude of the plant output is directly introduced into the cost function.
- For a given $\mathbf{Q}$, the *size* of $\mathbf{R}$ strongly influences the location of the closed loop poles. The larger $\mathbf{R}$ is, the slower is the control loop.

Further reading on optimal quadratic regulators can be found in the literature. See, for example, [1,3,4,8,9].

4.9 Observed State Feedback

4.9.1 Separation Strategy

Due to the drawbacks inherent in the measuring of the state, feedback of the estimated state can be used instead. The resulting control system integrates an observer and a feedback mechanism for the observed states.

The combination of a state observer and the feedback of the estimated state conform the structure shown in Figure 4.20.

In Figure 4.20, the (matrix) transfer functions $\mathbf{T}_1(s)$ and $\mathbf{T}_2(s)$ can be obtained from Figure 4.15. This yields

$$\mathbf{T}_1(s) = (s\mathbf{I} - \mathbf{A}_{\mathrm{o}} + \mathbf{J}\mathbf{C}_{\mathrm{o}})^{-1}\mathbf{B}_{\mathrm{o}} \tag{4.273}$$

$$\mathbf{T}_2(s) = (s\mathbf{I} - \mathbf{A}_{\mathrm{o}} + \mathbf{J}\mathbf{C}_{\mathrm{o}})^{-1}\mathbf{J} \tag{4.274}$$

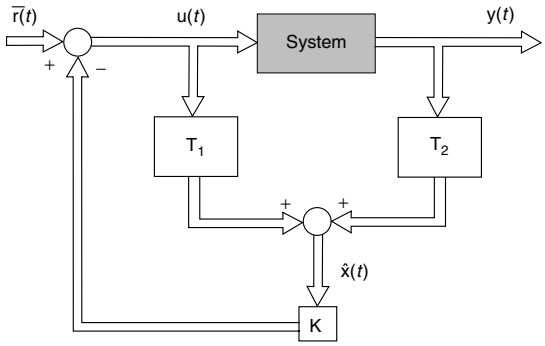


FIGURE 4.20 Estimated state feedback.

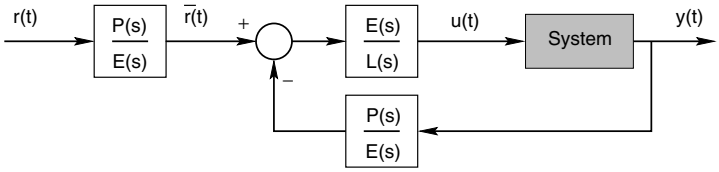


FIGURE 4.21 Equivalent control loop.

4.9.2 Transfer Function Interpretation for the Single-Input Single-Output Case

Consider a SISO plant having transfer function

$$G_o(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A_o})^{-1}\mathbf{B} = \frac{N_o(s)}{M_o(s)} \tag{4.275}$$

where $M_o(s)$ and $N_o(s)$ are polynomials in s.

First, a state feedback gain, **K**, is chosen to obtain a closed loop polynomial $F(s)$, where $F(s) = \det(s\mathbf{I} - \mathbf{A_o} + \mathbf{B_o K})$. Next, an observer gain, **J**, is computed to obtain an observer polynomial $E(s) = \det(s\mathbf{I} - \mathbf{A_o} + \mathbf{JC_o})$.

If the observer and the observed state feedback are combined, the resulting control loop can be made equivalent (by a suitable choice of $\bar{r}(t)$) to the classical control loop shown in Figure 4.21.

In Figure 4.21 the polynomials $P(s)$ and $L(s)$ satisfy the Diophantine equation

$$M_o(s)L(s) + N_o(s)P(s) = E(s)F(s) \tag{4.276}$$

This result says that the set of closed loop poles is the union of the set of observer poles and the set of state feedback poles.

References

1. Anderson, B.D.O. and Moore, J., *Linear optimal Control*. Prentice-Hall, Englewood Cliffs, NJ, 1971.
2. Anderson, B.D.O. and Moore, J., *Optimal Filtering*. Prentice-Hall, Englewood Cliffs, NJ, 1979.
3. Athans, M. and Falb, P., *Optimal Control*. McGraw HIll, 1966.
4. Dennis Bernstein and Wassim Haddad. LQG control with an H_∞ performance bound: A Riccati equation approach. *IEEE Transactions on Automatic Control*, 34(3): L293–305, 1989.
5. Bittanti, S., Laub, A.J., and Willems, J.C., *The Riccati Equation*. Springer Verlag, Berlin, 1996.

6. Dorf, R.C. and Bishop, R., *Modern Control Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1997.
7. Franklin, G.F. and Powel, J.D., *Digital Control of Dynamics Systems*. 2nd ed., Addison-Wesley, 1990.
8. Goodwin, G.C., Graebe, S., and Salgado, M.E., *Control System Design*. Prentice-Hall, NJ, 2001.
9. Kwakernaak, H. and Sivan, R., *Linear Optimal Control System*. Wiley–Interscience, New York, 1972.
10. Ogata, K., *State Space Analysis of Control Systems*. Prentice-Hall, Englewood Cliffis, NJ, 1967.
11. Rosenbrock, H.H., *State Space and Multivariable Theory*. John Wiley and Sons, New York, 1970.
12. Schultz, D.G. and Melsa, J.L., *State Function and Linear Control Systems*. McGraw Hill, New York, 1967.
13. Wiberg, D.W., *Theory and Problems of State Space and Linear Systems*. McGraw Hill, New York, 1971.
14. Zadeh, L.A. and Desoer, C.A., *Linear System Theory: The State Space Approach*. McGraw Hill, New York, 1963.
15. Zhou, K., *Essentials of Robust Control*. Prentice-Hall, Englewood Cliffs, NJ, 1998.
16. Zhou, K., Salomon, G., and Wu, E., Balanced realization and model reduction for unstable systems. *International Journal of Robust and Nonlinear Control*, 9:183–198, 1999.

5

Response of Dynamic Systems

Raymond A. de Callafon
University of California

5.1 System and Signal Analysis

In dynamic system design and analysis it is important to predict and understand the dynamic behavior of the system. Examining the dynamic behavior can be done by using a mathematical model that describes the relevant dynamic behavior of the system in which we are interested. Typically, a model is formulated to describe either continuous or discrete time behavior of a system. The corresponding equations that govern the model are used to predict and understand the dynamic behavior of the system.

A rigorous analysis can be done for relatively simple models of a dynamic system by actually computing solutions to the equations of the model. Usually, this analysis is limited to linear first and second order models. Although limited to small order models, the solutions tend to give insight in the typical responses of a dynamic system. For more complicated, higher order and possibly nonlinear models, numerical simulation tools provide an alternative for the dynamic system analysis.

In the following we review the analysis of linear models of discrete and continuous time dynamic systems. The equations that describe and relate continuous and discrete time behavior are presented. For the analysis of continuous time systems extensive use is made of the Laplace transform that converts linear differential equations into algebraic expressions. For similar purposes, a *z*-transform is used for discrete time systems.

5.1.1 Continuous Time Systems

Models that describe the linear continuous time dynamical behavior of a system are usually given in the form of differential equations that relate an input signal $u(t)$ to an output signal $y(t)$. The differential equation of a time invariant linear continuous time model has the general format

$$\sum_{j=0}^{n_a} a_j \frac{d^j}{dt^j} y(t) = \sum_{j=0}^{n_b} b_j \frac{d^k}{dt^j} u(t) \tag{5.1}$$

in which a linear combination is taken using the jth order time derivatives d^j/dt^j of a single output $y(t)$ and a single input $u(t)$. In (5.1), the scalar real valued numbers a_j for $j = 0,\ldots n_a$, $a_{n_a} \neq 0$ and b_j for $j = 0,\ldots,n_b$, $b_{n_b} \neq 0$, respectively, are called the denominator and numerator coefficients. The input $u(t)$ is distinguished from the output $y(t)$ in (5.1) by requiring $n_a \geq n_b$. As a result, the n_ath derivative is the highest derivative of the output $y(t)$ and n_a is used to indicate the order of the differential equation.

An alternative representation of a model of a continuous time system can be obtained by rewriting the n_ath order differential equation in (5.1) into a set of (coupled) first order differential equations. This can be done by introducing a state variable $x(t)$ and rewriting the higher order differential equation into

$$\begin{aligned} \frac{d}{dt}x(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{5.2}$$

where A, B, C, and D are real valued matrices. The set of first order differential equations given in (5.2) is referred to as a state space representation. The state variable $x(t)$ is a column vector and contains n_a variables, where n_a is the order of the differential equation.

The size of the matrices in (5.2) corresponds to the order of differential equation from which the state space realization is derived. For generalization purposes, consider multiple inputs and outputs rearranged in $m \times 1$ input column vector $u(t)$ and a $p \times 1$ output column vector $y(t)$. Given the $n_a \times 1$ size of the state vector, the state matrix A has size $n_a \times n_a$, the input matrix has size $n_a \times m$, the output matrix C has size $p \times n_a$, and the feedthrough matrix D has size $m \times p$. From these size considerations it can be observed that the state space realization in (5.2) easily generalizes the model description of multi-input multi-output systems.

To illustrate the concepts, consider the differential equation

$$m\frac{d^2}{dt^2}y(t) + c\frac{d}{dt}y(t) + ky(t) = u(t) \tag{5.3}$$

that describes the dynamical behavior of the one cart system given in Figure 5.1. The differential Equation 5.3 is found by writing Newton's second law for the cart mass m with position output $y(t)$, spring force $ky(t)$, damper force $c(d/dt)y(t)$, and force input $u(t)$. Comparing with (5.1) it can be seen that $n_a = 2 \geq n_b = 0$, making (5.3) a second order differential equation. The differential equation can be rewritten into a state space representation (5.2) by defining the state variable

$$x(t) := \begin{bmatrix} y(t) \\ \frac{d}{dt}y(t) \end{bmatrix}$$

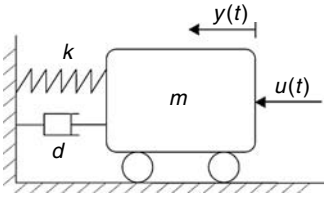


FIGURE 5.1 One cart system representing a single mass dynamical system with cart mass m, spring constant k, and damping constant c.

that consists the position and velocity of the mass. With this state variable (5.3) can be rewritten into

$$\frac{d}{dt}x(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{d}{m} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + 0\,u(t)$$

which yields a state space model similar to (5.2). In this case, the size of the state matrix A is 2×2, the input matrix B is 2×1, the output matrix C is 1×2, and the feedthrough matrix $D = 0$ is scalar.

5.1.2 Discrete Time Systems

Discrete time models approximate and describe the sampled data behavior of a continuous time dynamical system. In some applications, such as digital control, the dynamical control system is inherently discrete time. In these situations, analysis with discrete time equivalent models in necessary.

For analysis purposes, both input $u(t)$ and output $y(t)$ are assumed to be sampled on a regular discrete time interval

$$t = k\Delta T, \qquad k = 0,1,2,\ldots$$

where ΔT indicates the sampling time. To maintain uniform notation throughout the analysis, the sampling time ΔT is normalized to $\Delta T = 1$ and the time dependency t is assumed to be discrete with $t = k = 0,1,2,\ldots$.

Given sampled or discrete time input/output data, a linear discrete time model can be formulated in the form of a difference equation

$$\sum_{i=0}^{n_c} c_k y(k+j) = \sum_{i=0}^{n_d} d_k u(k+j) \tag{5.4}$$

in which a linear combination is taken of positive time shifted inputs $u(k)$ and outputs $y(k)$. To distinguish the differential equation from the differential Equation 5.1, different scalar real valued numbers c_j for $j = 0,\ldots,n_c$, $c_{n_c} \neq 0$ and d_j for $j = 0,\ldots,n_d$, $d_{n_d} \neq 0$ are used. The input $u(k)$ is distinguished from the output $y(k)$ in (5.1) by requiring $n_c \geq n_d$ for causality purposes. As a result, the n_c is the largest time shift of the output $y(k)$ and n_c is used to indicate the order of the difference equation.

The simplicity with which the difference equation can be represented also allows an algebraic representation of (5.4). Introducing the time shift operator

$$qu(k) := u(k+1) \tag{5.5}$$

allows (5.4) to be rewritten into the algebraic expression

$$y(k)\sum_{i=0}^{n_c} c_j q^j = u(k)\sum_{i=0}^{n_d} d_j q^j$$

Following this analysis, the discrete time output $y(k)$ can be represented by the difference model

$$y(k) = G(q)u(k), \qquad \text{with } G(q) = \frac{\sum_{j=0}^{n_d} d_j q^j}{\sum_{j=0}^{n_c} c_j q^j} \tag{5.6}$$

where the scalar real valued numbers c_j for $j = 0,\ldots,n_c$, $c_{n_c} \neq 0$ and d_j for $j = 0,\ldots,n_d$, $d_{n_d} \neq 0$, respectively, indicate the denominator and numerator coefficients.

Similar to the continuous time system representation, the higher order difference Equation 5.4 can also be rewritten into a set of (coupled) first order difference equations for analysis purposes. This can be done by introducing a state variable $x(k)$ and rewriting the higher order difference equation into

$$\begin{aligned} qx(k) &= Fx(k) + Gu(k) \\ y(k) &= Hx(k) + Ju(k) \end{aligned} \tag{5.7}$$

where $qx(k) = x(k+1)$, according to (5.5). The state variable $x(k)$ is a column vector and contains n_c variables, where n_c is the order of the difference equation. The state space matrices in (5.7) are labeled differently to distinguish them from the continuous time state space model.

5.1.3 Laplace and z Transform

An important mathematical concept for the analysis of models described by linear differential equations such as (5.1) and (5.2) is the Laplace transform. As indicated before, the Laplace transform converts linear differential equations into algebraic expressions. With this conversion, proper algebraic manipulation can be used to recover solutions of the differential equation. In a similar manner, the z transform is used for discrete time models described by difference equations. Although it was shown in (5.6) that a difference equation can be written as an algebraic expression, the z transform allows complex analysis of the discrete time models.

The Laplace transform of a signal $u(t)$ is defined to be

$$L\{u(t)\} := u(s) = \int_{t=0}^{\infty} u(t)e^{-st}\,dt \tag{5.8}$$

where the integration over t eliminates the time dependency and the transform $u(s)$ is a function of the Laplace variable only. This is indicated in the transform $u(s)$ where the dependency of t has been dropped, and $u(s)$ is a function of the (complex valued) Laplace variable s only.

The integral (5.8) exists for most commonly used signals $u(t)$, provided certain conditions on s are imposed. To illustrate the transform, consider a (unity) step signal

$$u(t) := \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$

where the shape of $u(t)$ resembles a stepwise change of an input signal. With the definition of the Laplace transform in (5.8) the transform of the step signal becomes

$$u(s) = \int_{t=0}^{\infty} u(t)e^{-st}\,dt = \int_{t=0}^{\infty} e^{-st}\,dt = -\frac{e^{-st}}{s}\bigg|_0^{\infty} = \frac{1}{s} \tag{5.9}$$

where it is assumed that the real part of s is greater than zero so that $\lim_{t\to\infty} e^{-st} = 0$.

If a signal $u(k)$ is given at discrete time samples $k = 0, 1, 2\ldots$, the integral expression of (5.8) cannot be applied. Instead, a transform similar to the Laplace transform can be used and denoted by the z transform. The z-transform of a discrete time signal $u(k)$ is defined as

$$L\{u(k)\} := u(z) = \sum_{k=0}^{\infty} u(k)z^{-k} \tag{5.10}$$

The series (5.10) converges if it is assumed that there exist values r_l and r_u with $r_l < |z| < r_u$ as bounds on the magnitude of the complex variable z.

The z transform has the same role in discrete time systems that the Laplace transform has in continuous time systems. In case of sampling, the complex variable z of the z transform is related to the complex variables s in the Laplace transform via

$$z = e^{s\Delta T} \tag{5.11}$$

where ΔT is the sampling time used for sampling. Both the Laplace and z-transform are linear operators and satisfy

$$L\{\alpha u(t) + \beta y(t)\} = \alpha L\{u(t)\} + \beta\{y(t)\} \tag{5.12}$$

Using the definition in (5.8) and the linearity property in (5.12), the transform of most commonly used functions has been precalculated and tabulated.

Of particular interest for the analysis of linear differential equations such as (5.1) and (5.2) is the Laplace transform of a derivative:

$$\begin{aligned} L\left\{\frac{d}{dt}u(t)\right\} &= \int_{t=0}^{\infty} \frac{d}{dt}u(t)e^{-st}\,dt \\ &= u(t)e^{-st}\Big|_0^{\infty} + s\int_{t=0}^{\infty} u(t)e^{-st}\,dt \\ &= su(s) - u(0) \end{aligned}$$

With $u(0) = 0$ it can be seen that the Laplace transform of the derivative of $u(t)$ is simply s times the Laplace transform of $u(s)$. This result can be extended to higher order derivatives and the result for the nth derivative is given by

$$L\left\{\frac{d^n}{dt^n}u(t)\right\} = s^n u(s) - \sum_{j=1}^{n} s^{n-j}\frac{d^{j-1}}{dt^{j-1}}u(t)\Big|_{t=0}$$

In case the signal $u(t)$ satisfies the initial zero conditions

$$\frac{d^{j-1}}{dt^{j-1}}u(t)\Big|_{t=0} = 0 \quad \text{for } j = 1, \ldots, n$$

the formula reduces to

$$L\left\{\frac{d^n}{dt^n}u(t)\right\} = s^n u(s)$$

and the Laplace transform of an nth order derivative is simply s^n times the transform $u(s)$.

For discrete time systems the interest lies in the z-transform of a time-shifted signal. Similar to the Laplace transform, the z transform of an n time-shifted signal can be computed and is given by

$$L\{q^n u(k)\} = z^n u(z) - \sum_{j=0}^{n-1} z^{n-j}u(j)$$

In case the discrete time signal $u(k)$ satisfies the initial zero conditions $u(j) = 0$ for $j = 0,\ldots,n-1$, the formula reduces to

$$L\{q^n u(k)\} = z^n u(z)$$

and the z transform of an n time-shifted discrete time signal is simply z^n times the transform $u(z)$.

5.1.4 Transfer Function Models

The results of the Laplace and z-transform can be used to reduce linear differential Equations 5.1 and difference Equation 5.4 to the algebraic expressions. Starting with the differential equations for continuous time models and assuming zero initial conditions for both the input $u(t)$ and output signal $y(t)$, the Laplace transform of (5.1) yields

$$y(s)\sum_{j=0}^{n_a} a_j s^j = u(s)\sum_{j=0}^{n_b} b_j s^j$$

which can be written in transfer function format

$$y(s) = G(s)u(s), \quad \text{with } G(s) = \frac{\sum_{j=0}^{n_b} b_j s^j}{\sum_{j=0}^{n_a} a_j s^j} \tag{5.13}$$

In (5.13), the transfer function $G(s)$ is the ratio of the numerator polynomial $\sum_{j=0}^{n_b} b_j s^j$ and the denominator polynomial $\sum_{j=0}^{n_a} a_j s^j$. As indicated before, the scalar real valued numbers a_j for $j = 0,\ldots,n_a$, $a_{n_a} \neq 0$ and b_j for $j = 0, \ldots, n_b$, $b_{n_b} \neq 0$, respectively, are called the denominator and numerator coefficients.

Similarly for the discrete time model, assuming zero initial conditions for both the input $u(k)$ and output signal $y(k)$, the z-transform of (5.4) yields

$$y(z)\sum_{j=0}^{n_c} c_j z^j = u(z)\sum_{j=0}^{n_d} b_j z^j$$

which can be written in transfer function format

$$y(z) = G(z)u(z), \quad \text{with } G(z) = \frac{\sum_{j=0}^{n_c} c_j z^j}{\sum_{j=0}^{n_a} a_j z^j} \tag{5.14}$$

From the transfer function representations, poles and zeros of the dynamic system can be computed for dynamic system analysis. The poles of the system are defined as the roots of the denominator polynomial. The zeros of the system are defined as the roots of the numerator polynomial.

The Laplace and z-transform can also be used to reduce the state space representation to a set of algebraic expressions that consists of (coupled) first order polynomials. Assuming zero initial conditions for the state vector $x(t)$, application of the Laplace transform to (5.2) yields

$$\begin{aligned} sx(s) &= Ax(s) + Bu(s) \\ y(s) &= Cx(s) + Du(s) \end{aligned}$$

in which the state vector $x(s)$ can be eliminated. Solving for $x(s)$ gives $x(s) = (sI - A)^{-1} Bu(s)$ and the above transform can be rewritten into a transfer function representation

$$y(s) = G(s)u(s), \quad \text{with } G(s) = D + C(sI - A)^{-1}B \tag{5.15}$$

Under mild technical conditions involving controllability and observability of the state space model, the transfer function representations in (5.13) and (5.15) are similar in case the state space model in (5.2) is derived from the differential Equation 5.1 and vice versa.

5.2 Dynamic Response

The Laplace and z transform offer the possibility to compute the dynamic response of a dynamic system by means of algebraic manipulations. The analysis of the dynamic response gives insight into the dynamic behavior of the system by addressing the response to typical test signals such as impulse, step, and sinusoid excitation of the system.

The response can be computed for relatively simple continuous or discrete dynamical systems given by low order differential or difference equations. Both the state space model and the transfer function descriptions provide helpful representations in the analysis of a dynamic system. The result are presented in the following.

5.2.1 Pulse and Step Response

A possible way to evaluate the response of a dynamic system is by means of pulse and step based test signals. For continuous time systems an input impulse signal is defined as a δ function

$$u_{\text{imp}}(t) := \delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases}$$

with the property

$$\int_{t=-\infty}^{\infty} f(t)\delta(t) = f(0)$$

where $f(t)$ is an integrable function over $(-\infty, \infty)$. Although an impulse signal is not practical from an experiment point of view, the computation or simulation of the impulse response gives insight into the transient behavior of the dynamical system.

With the properties of the impulse function $\delta(t)$ mentioned above, the Laplace transform of the impulse function is given by

$$L\{\delta(t)\} = \delta(s) = \int_{t=0}^{\infty} \delta(t)e^{-st}\,dt = e^{-s0} = 1$$

Hence the output $y(s)$ due to an impulse input is given by $y_{\text{imp}}(s) = G(s)u_{\text{imp}}(s) = G(s)\delta(s) = G(s)$. As a result, an immediate inverse Laplace transform of the continuous time transfer function $G(s)$,

$$y_{\text{imp}}(t) = L^{-1}\{G(s)\}$$

gives the dynamic response $y_{\text{imp}}(t)$ of the system to an impulse input response.

The computation of the step response is done in a similar way. In (5.9), the Laplace transform of the step signal

$$u_{\text{step}}(t) := \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$

is given as $u_{\text{step}}(s) = 1/s$. Consequently, with $y_{\text{step}}(s) = G(s)u_{\text{step}}(s) = G(s)/s$, the inverse Laplace transform of $G(s)/s$

$$y_{\text{step}}(t) = L^{-1}\left\{\frac{G(s)}{s}\right\}$$

will yield the dynamic response $y_{\text{step}}(t)$ of the system to a step input response.

From a practical point of view, the computation of an inverse Laplace transform is limited to low order models of first or second order. However, the results give insight into the dominant behavior of most dynamic systems. This is illustrated in the following examples.

- Consider a first order continuous model given by the transfer function

$$G(s) = \frac{K}{\tau s + 1}$$

where K and τ indicate, respectively, the static gain and the time constant of the system. Such a transfer function may arise from a simple RC network with $\tau = RC$. In order to compute the step response of the system, the inverse Laplace transform of $G(s)/s$ needs to be computed. This inverse Laplace transform is given by

$$y_{\text{step}}(t) = L^{-1}\left\{\frac{G(s)}{s}\right\} = \frac{K}{\tau}(1 - e^{-t/\tau})$$

and it can be seen that the step response is an exponential function. For stability the time constant τ needs to satisfy $\tau > 0$. It can also be observed that the smaller the time constant, the faster the response.

- Consider a second order continuous time model given by the transfer function

$$G(s) = \frac{\omega_n^2}{s^2 + 2\beta\omega_n s + \omega_n^2} \tag{5.16}$$

where ω_n and β, respectively, indicate the undamped resonance frequency and the damping coefficient of the system. This model can be derived from the dynamical behavior of the one cart system depicted in Figure 5.1 and given in (5.3). For $\beta < 1$ (underdamped), the inverse Laplace transform of $G(s)$ is given by

$$y_{\text{imp}}(t) = \frac{\omega_m}{\sqrt{1-\beta^2}} e^{-\beta\omega_n t} \sin\left(\omega_n\sqrt{1-\beta^2}\, t\right)$$

From this expression it can be observed that the response is a decaying sinusoid with a resonance frequency of $\omega_n\sqrt{1-\beta^2}$. For stability, both $\omega_n > 0$ and $\beta > 0$ and the larger ω_n, the faster the decay of the sinusoid and the higher is the frequency of the response $y_{\text{imp}}(t)$. Illustration of the impulse response of this second order system have been depicted in Figures 5.2 and 5.3 where variations in the undamped resonance frequency ω_n and the damping coefficient β illustrate the dynamic behavior of the system.

For discrete systems, the analysis of the pulse response is based on the discrete time pulse function

$$u_{\text{imp}}(k) := \delta(k) = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$$

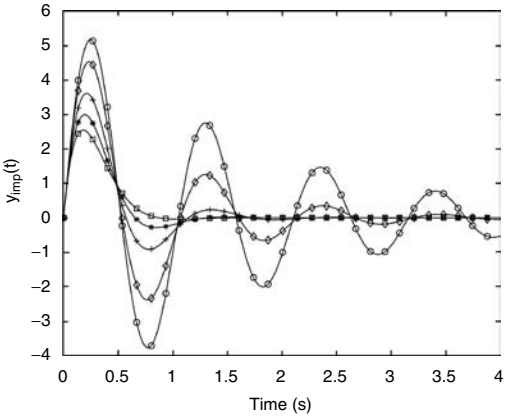


FIGURE 5.2 Variations in impulse response $y_{\text{imp}}(t)$ of second order system with $\omega_n = 6$ and $\beta = 0.1(\bigcirc)$, $0.2(\diamond)$, $0.4(+)$, $0.6\ (*)$, $0.8(\square)$.

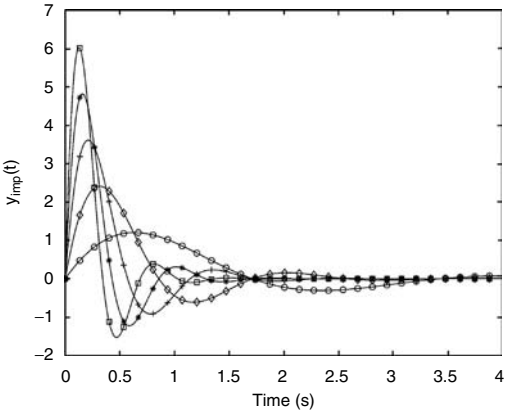


FIGURE 5.3 Variations in impulse response $y_{\text{imp}}(t)$ of second order system with $\beta = 0.4$ and $\omega_n = 2(\bigcirc)$, $4(\diamond)$, $6(+)$, $8(*)$, $10(\square)$.

which has a value of 1 at $k = 0$ and zero anywhere else. The step signal is similar to the continuous time signal and is given by

$$u_{\text{step}}(k) := \begin{cases} 0, & k < 0 \\ 1, & k \geq 1 \end{cases}$$

In order to characterize the discrete time pulse and step response a similar procedure as for the continuous time model can be followed by using the z transform. It is easy to show that the z-transform $u_{\text{imp}}(z) = 1$ and the z transform of the step signal equals $u_{\text{step}} z = z/(z-1)$. Hence, the response of the discrete time system to a pulse or step signal can be computed with

$$y_{\text{imp}}(k) = L^{-1}\{G(z)\}, \qquad y_{\text{step}}(k) = L^{-1}\left\{\frac{G(z)z}{z-1}\right\}$$

In addition to the approach using a z-transform, the ratio of the polynomials in the difference model (5.6) can be written in a series expansion:

$$G(q) = \frac{\sum_{j=0}^{n_d} d_j q^j}{\sum_{j=0}^{n_c} c_j q^j} = \sum_{j=0}^{\infty} g_k q^{-k}$$

With the discrete time pulse function $u_{\text{imp}}(k)$ as an input, it can be observed that

$$y_{\text{imp}}(k) = \sum_{j=0}^{\infty} g_k q^{-k} \delta(k) = g_k$$

and it can be concluded that the pulse response $y_{\text{imp}}(k)$ equals the coefficients in the series expansion of the difference equation. Similarly, with the discrete time step function $u_{\text{step}}(k)$ as an input, it can be observed that

$$y_{\text{imp}}(k) = \sum_{j=0}^{\infty} g_k q^{-k} u_{\text{step}}(k) = \sum_{j=0}^{k} g_k$$

and it can be concluded that the step response $y_{\text{step}}(k)$ values are computed as a finite sum of the coefficients in the series expansion of the difference equation. The computation of a discrete time pulse response for a first order discrete time model is given in the following example.

- Consider a first order discrete model given by the difference model

$$G(q) = \frac{1}{q + d}$$

where d indicates the discrete time constant of the system. The series expansion of the difference model can be computed as follows:

$$G(q) = \frac{1}{q - d} = \sum_{j=0}^{\infty} d^j$$

and it can be seen that the discrete time pulse response

$$y_{\text{imp}}(k) = d^k$$

is an exponential function. For stability the discrete constant d needs to satisfy $|d| < 1$. Similar as in the continuous time model it can be observed that the smaller the time constant, the faster the response. Additionally, the first order discrete time model may exhibit an oscillation in case $-1 < d < 0$.

5.2.2 Sinusoid and Frequency Response

So far we have considered transient effects caused by step, pulse, and impulse inputs to investigate the dynamic properties of a dynamical system. However, periodic inputs occur frequently in practical situations and the analysis of a dynamic system to periodic inputs and especially sinusoidal inputs can yield more insight into the behavior of the system.

The response of a linear system to a sinusoidal input is referred to as the frequency response of the system. An input signal, $u(t) = U\sin\omega t$, that is, a sine wave with amplitude U and frequency ω_f, has a Laplace transform

$$u(s) = \frac{U\omega}{s^2 + \omega^2}.$$

Consequently, the response of the system is given by

$$y(s) = G(s)\frac{U\omega}{s^2 + \omega^2}$$

and a partial fraction expansion of $y(s)$ will result in terms that represent the (stable) transient behavior of $y(s)$ and the term associated to the sinusoidal input $u(s)$. Elimination of the transient effects and performing an inverse Laplace transform will yield a periodic time response $y(t)$ of the same frequency ω_f given by

$$y(t) = AU\sin(\omega t + \phi)$$

where the amplitude magnification A and the phase shift ϕ are given by

$$A = G(s)\big|_{s=i\omega}, \qquad \phi = \angle G(s)\big|_{s=i\omega} \tag{5.17}$$

By evaluating the transfer function $G(s)$ along the imaginary axis $s = i\omega$, $\omega \geq 0$, the magnitude $|G(i\omega)|$ gives information on the relative amplification of the sinusoidal input, whereas the phase $\angle G(i\omega)$ gives information on the relative phase shift between input and output.

This analysis can be easily extended to discrete time systems by employing the relation between the Laplace variable s and the z-transform variable in (5.11) to obtain the discrete time sinusoidal response

$$y(k) = AU\sin(\omega k + \phi)$$

where the amplitude magnification A and the phase shift ϕ are given by

$$A = G(z)\big|_{z=e^{i\Delta T\omega}}, \qquad \phi = \angle G(z)\big|_{z=e^{i\Delta T\omega}} \tag{5.18}$$

Due to the sampling nature of the discrete time system, the transfer function $G(z)$ is now evaluated on the unit circle

$$e^{i\Delta T\omega}, \quad 0 \leq \omega < \frac{\pi}{\Delta T}$$

to attain information of the magnitude and phase shift of the sinusoidal response.

Plotting the frequency response of a dynamical system gives insight in the pole locations (resonance modes) and zero locations of the dynamical system. As an example, the frequency response of the second order system given in (5.16) has been depicted in Figure 5.4. It can be seen from the figure that, as expected, the second order system is less damped for smaller damping coefficients β and this results in a larger amplitude response of the second order system at the resonance frequency $\omega_n = 6$ rad/s. It can also be observed that the phase change at the resonance frequency becomes more abrupt for smaller damping coefficients.

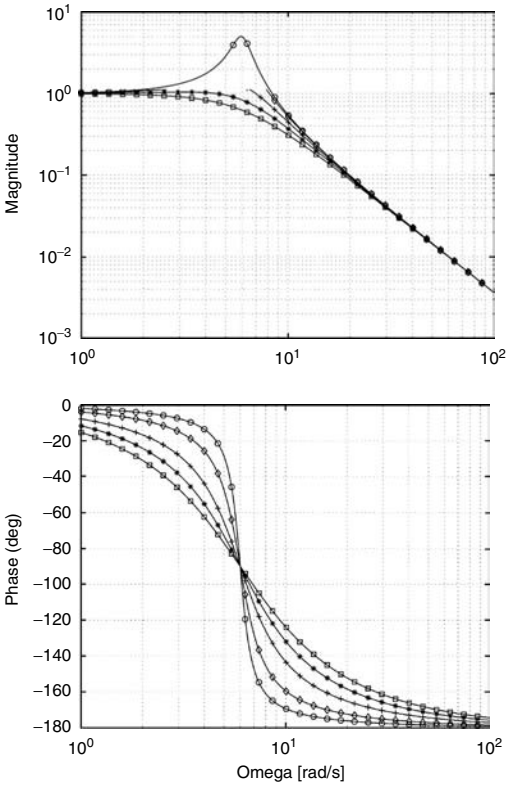


FIGURE 5.4 Variations in frequency response of second order system $G(s)$ with $\omega_n = 6$ and $\beta = 0.1(\bigcirc), 0.2(\diamond), 0.4(+), 0.6(\star), 0.8(\square)$.

5.3 Performance Indicators for Dynamic Systems

5.3.1 Step Response Parameters

Specifications for dynamic systems often involve requirements on the transient behavior of the system. Transient behavior requirements can be formulated on the basis of a step response and the most significant parameters have been summarized below and illustrated in Figure 5.5.

- Steady state or DC value y_s of step response output.
- The steady state error y_{se} is the error between steady state value y_s and desired DC value of step response output.
- The maximum overshoot A_m is the maximum deviation of the step response output above its steady state value y_s.
- The peak time t_p is the time at which the maximum overshoot occurs.
- Settling time t_s is the time at which the step response input stays within some small percentage rage of the steady state value y_s. Typically, a percentage of 2% or 5% is chosen to determine the settling time.
- The rise time t_r is usually defined as the time required for the step response output to rise from 10% to 90% of the steady state value y_s.
- The delay time t_d is defined as the time required to reach 50% of the steady state value y_s.

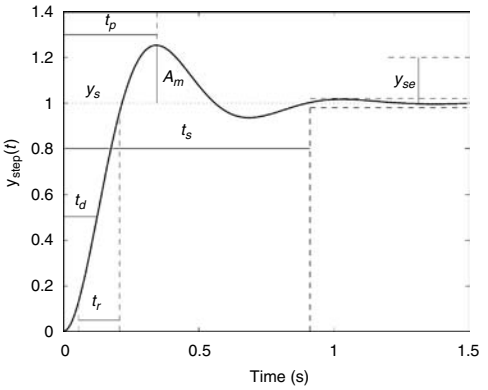


FIGURE 5.5 Parameters for step-response behavior: steady state value y_s, steady state error y_{se}, maximum overshoot A_m, peak time t_p, settling time t_s, rise time t_r, and delay time t_d.

Most of the above value can be obtained from an experimentally determined step response. In general, they cannot be obtained in an analytical form, except for low order models. For the second order model of the one mass system given in (5.3), some analytical results can be obtained. For a second order model of (5.3), the maximum overshoot A_m is determined by

$$A_m = 100e^{-\pi\beta/\sqrt{1-\xi^2}}, \quad \text{where } \xi = \frac{A}{\sqrt{\pi^2 + A^2}}, \quad A = \ln\left(\frac{100}{A_m}\right)$$

The peak time t_p can be computed by

$$t_p = \frac{\pi}{\omega_n\sqrt{1-\xi^2}}$$

whereas the delay time t_d can be approximated by

$$t_d \approx \frac{1+0.7\xi}{\omega_n}$$

As the maximum overshoot increases with a smaller damping coefficient β in the system, the maximum overshoot is often used to indicate the relative stability of the system.

5.3.2 Frequency Domain Parameters

With the frequency domain analysis of dynamic systems, specifications for the dynamic properties of a system can also be stated in the frequency domain. Frequency domain specifications in filter design often address ripple, bandwidth, roll-off, and phase lag parameters. Similar characteristics can also be specified for dynamic systems in case the model of the system is analyzed in the frequency domain. The most significant parameters have been summarized below and illustrated in Figure 5.6.

- The bandwidth ω_b is a notion for the maximum frequency at which the output will track a sinusoidal input in a satisfactory manner. By convention, the bandwidth is defined as the frequency at which the output is attenuated −3 dB (0.707).
- The resonant frequency ω_r is the first frequency at which a significant resonance mode with low damping occurs. The resonance mode can, if uncontrolled, negatively influence the settling time of the dynamic system and plays an important role in characterization of performance

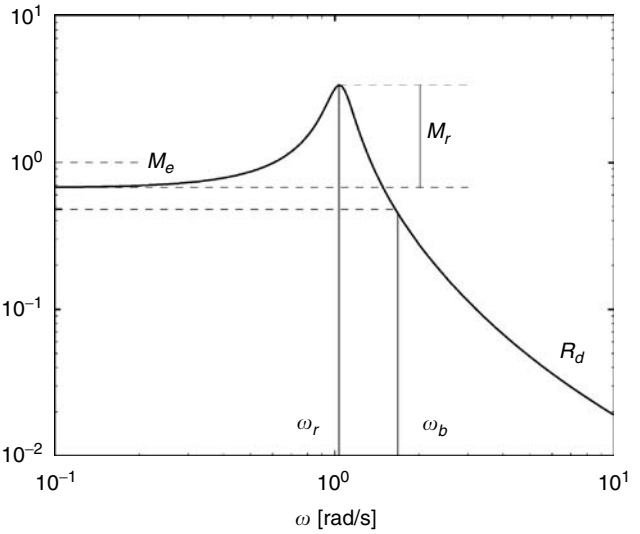


FIGURE 5.6 Parameters for frequency response behavior: bandwidth ω_b, resonance frequency ω_r, resonant peak M_r, steady state error M_e, and roll-off R_d.

- The resonant peak M_r is the height of a resonance mode. The resonant peak is a measure for the damping. As illustrated in Figure 5.2 for a second order model, the resonance mode increases at lower damping coefficients.
- Steady state errors M_e can also be analyzed in the frequency response of a system. Using the final value theorem for continuous time systems

$$\lim_{t\to\infty} y(t) = y_s = \lim_{s\to 0} sy(s)$$

the presence of steady state errors can be inspected in the frequency domain by evaluation $|G(s)|$ at $s = i\omega = 0$ or for small values of the frequency vector ω. This can be seen as follows. As the Laplace transform $u_{\text{step}}(s)$ of a step input signal $u_{\text{step}}(t)$ is $u_{\text{step}}(s) = 1/s$,

$$\lim_{t\to\infty} y_{\text{step}}(t) = \lim_{s\to 0} sy_{\text{step}}(s) = \lim_{s\to 0} sG(s)\frac{1}{s} = \lim_{s\to 0} G(s)$$

By evaluating $|G(i\omega)|$ for small frequencies ω, the steady state behavior of $G(s)$ can be studied.

A similar result exist for discrete time systems, where the final value theorem reads as follows. If $u(z)$ converges for $|z| > 1$ and all poles of $(z-1)u(z)$ are inside the unit circle, then

$$\lim_{k\to\infty} u(k) = \lim_{z\to 1} (z-1)u(z)$$

Hence, for discrete time systems the steady state behavior of a transfer function $G(z)$ can be studied by evaluating $|G(e^{i\omega\Delta T})|$ for small frequencies ω.

- Roll-off R_d at high frequencies is defined as the negative slope of the frequency response at higher frequencies. The roll-off determines the performance of the dynamic system as high frequent disturbances can be amplified if a dynamic system does not have enough high frequent roll-off.

6

The Root Locus Method

Hitay Özbay
The Ohio State University

6.1 Introduction

The root locus technique is a graphical tool used in feedback control system analysis and design. It has been formally introduced to the engineering community by W. R. Evans [3,4], who received the Richard E. Bellman Control Heritage Award from the American Automatic Control Council in 1988 for this major contribution.

In order to discuss the root locus method, we must first review the basic definition of bounded input bounded output (BIBO) stability of the standard linear time invariant feedback system shown in Figure 6.1, where the plant, and the controller, are represented by their transfer functions $P(s)$ and $C(s)$, respectively.* The plant, $P(s)$, includes the physical process to be controlled, as well as the actuator and the sensor dynamics.

The feedback system is said to be stable if none of the closed-loop transfer functions, from external inputs r and v to internal signals e and u, have any poles in the closed right half plane, $\overline{\mathbb{C}}_+ := \{s \in \mathbb{C} : \mathrm{Re}(s) \geq 0\}$. A necessary condition for feedback system stability is that the closed right half plane zeros of $P(s)$ (respectively $C(s)$) are distinct from the poles of $C(s)$ (respectively $P(s)$). When this condition holds, we say that there is no unstable pole–zero cancellation in taking the product $P(s)C(s) =: G(s)$, and then checking feedback system stability becomes equivalent to checking whether all the roots of

$$1 + G(s) = 0 \tag{6.1}$$

are in the open left half plane, $\mathbb{C}_- := \{s \in \mathbb{C} : \mathrm{Re}(s) < 0\}$. The roots of (6.1) are the closed-loop system poles. We would like to understand how the closed-loop system pole locations vary as functions of a real parameter of $G(s)$. More precisely, assume that $G(s)$ contains a parameter K, so that we use the notation

*Here we consider the continuous time case; there is essentially no difference between the continuous time case and the discrete time case, as far as the root locus construction is concerned. In the discrete time case the desired closed-loop pole locations are defined relative to the unit circle, whereas in the continuous time case desired pole locations are defined relative to the imaginary axis.

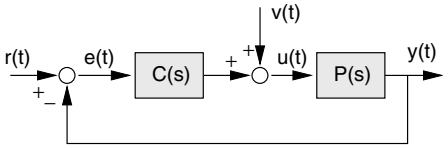


FIGURE 6.1 Standard unity feedback system.

$G(s) = G_K(s)$ to emphasize the dependence on K. The *root locus* is the plot of the roots of (6.1) on the complex plane, as the parameter K varies within a specified interval.

The most common example of the root locus problem deals with the uncertain (or adjustable) gain as the varying parameter: when $P(s)$ and $C(s)$ are fixed rational functions, except for a gain factor, $G(s)$ can be written as $G(s) = G_K(s) = KF(s)$, where K is the uncertain/adjustable gain, and

$$F(s) = \frac{N(s)}{D(s)} \quad \text{where} \quad \begin{aligned} N(s) &= \prod_{j=1}^{m}(s - z_j) \\ D(s) &= \prod_{i=1}^{n}(s - p_i), \end{aligned} \qquad n \geq m \tag{6.2}$$

with $z_1, \ldots, z_m$, and $p_1, \ldots, p_n$ being the open-loop system zeros and poles. In this case, the closed-loop system poles are the roots of the characteristic equation

$$\chi(s) := D(s) + KN(s) = 0 \tag{6.3}$$

The *usual root locus* is obtained by plotting the roots $r_1(K), \ldots, r_n(K)$ of the characteristic polynomial $\chi(s)$ on the complex plane, as K varies from 0 to $+\infty$. The same plot for the negative values of K gives the *complementary root locus*. With the help of the root locus plot the designer identifies the admissible values of the parameter K leading to a set of closed-loop system poles that are in the desired region of the complex plane. There are several factors to be considered in defining the "desired region" of the complex plane in which all the roots $r_1(K), \ldots, r_n(K)$ should lie. Those are discussed briefly in the next section. Section 6.3 contains the root locus construction procedure, and design examples are presented in Section 6.4.

The root locus can also be drawn with respect to a system parameter other than the gain. For example, the characteristic equation for the system $G(s) = G_\lambda(s)$, defined by

$$G_\lambda(s) = P(s)C(s), \quad P(s) = \frac{(1 - \lambda s)}{s(1 + \lambda s)}, \quad C(s) = K_c\left(1 + \frac{1}{T_I s}\right)$$

can also be transformed into the form given in (6.3). Here K_c and T_I are given fixed PI (Proportional plus Integral) controller parameters, and $\lambda > 0$ is an uncertain plant parameter. Note that the phase of the plant is

$$\angle P(j\omega) = -\frac{\pi}{2} - 2\tan^{-1}(\lambda\omega)$$

so the parameter λ can be seen as the uncertain phase lag factor (e.g., a small uncertain time delay in the plant can be modeled in this manner, see [9]). It is easy to see that the characteristic equation is

$$s^2(\lambda s + 1) + K_c(1 - \lambda s)\left(s + \frac{1}{T_I}\right) = 0$$

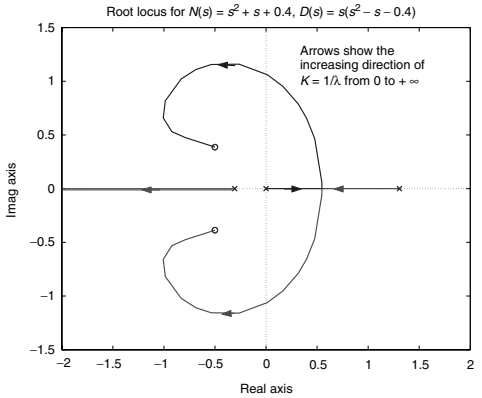


FIGURE 6.2 The root locus with respect to $K = 1/\lambda$.

and by rearranging the terms multiplying λ this equation can be transformed to

$$1 + \frac{1}{\lambda}\frac{(s^2 + K_c s + K_c/T_I)}{s(s^2 - K_c s - K_c/T_I)} = 0$$

By defining $K = \lambda^{-1}$, $N(s) = (s^2 + K_c s + K_c/T_I)$, and $D(s) = s(s^2 - K_c s - K_c/T_I)$, we see that the characteristic equation can be put in the form of (6.3). The root locus plot can now be obtained from the data $N(s)$ and $D(s)$ defined above; that shows how closed-loop system poles move as λ^{-1} varies from 0 to $+\infty$, for a given fixed set of controller parameters K_c and T_I. For the numerical example $K_c = 1$ and $T_I = 2.5$, the root locus is illustrated in Figure 6.2.

The root locus construction procedure will be given in Section 6.3. Most of the computations involved in each step of this procedure can be performed by hand calculations. Hence, an approximate graph representing the root locus can be drawn easily. There are also several software packages to generate the root locus automatically from the problem data $z_1, \ldots, z_m$, and $p_1, \ldots, p_n$.

If a numerical computation program is available for calculating the roots of a polynomial, we can also obtain the root locus with respect to a parameter which enters into the characteristic equation nonlinearly. To illustrate this point let us consider the following example: $G(s) = G_{\omega_o}(s)$ where

$$G_{\omega_o}(s) = P(s)C(s), \quad P(s) = \frac{(s-0.1)}{(s^2 + 1.2\omega_o s + \omega_o^2)(s+0.1)}, \quad C(s) = \frac{(s-0.2)}{(s+2)}$$

Here $\omega_o \geq 0$ is the uncertain plant parameter. Note that the characteristic equation

$$1 + \frac{\omega_o(1.2s + \omega_o)(s+0.1)(s+2)}{s^2(s+0.1)(s+2) + (s-0.2)(s-0.1)} = 0 \tag{6.4}$$

cannot be expressed in the form of $D(s) + KN(s) = 0$ with a single parameter K. Nevertheless, for each ω_o we can numerically calculate the roots of (6.4) and plot them on the complex plane as ω_o varies within a range of interest. Figure 6.3 illustrates all the four branches, $r_1(K), \ldots, r_4(K)$, of the root locus for this system as ω_o increases from zero to infinity. The figure is obtained by computing the roots of (6.4) for a set of values of ω_o by using MATLAB.

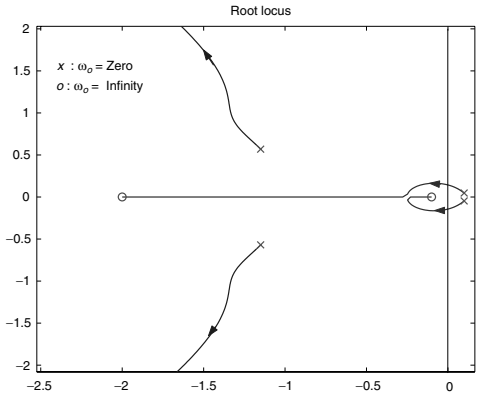


FIGURE 6.3 The root locus with respect to ω_o.

6.2 Desired Pole Locations

The performance of a feedback system depends heavily on the location of the closed-loop system poles $r_i(K) = 1,\ldots,n$. First of all, for stability we want $r_i(K) \in \mathbb{C}_-$ for all $i = 1,\ldots,n$. Clearly, having a pole "close" to the imaginary axis poses a danger, that is, "small" perturbations in the plant might lead to an unstable feedback system. So the desired pole locations must be such that stability is preserved under such perturbations (or in the presence of uncertainties) in the plant. For second-order systems, we can define certain stability robustness measures in terms of the pole locations, which can be tied to the characteristics of the step response. For higher order systems, similar guidelines can be used by considering the dominant poles only.

In the standard feedback control system shown in Figure 6.1, assume that the closed-loop transfer function from $r(t)$ to $y(t)$ is in the form

$$T(s) = \frac{\omega_o^2}{s^2 + 2\zeta\omega_o s + \omega_o^2}, \quad 0 < \zeta < 1, \quad \omega_o \in \mathrm{R}$$

and $r(t)$ is the unit step function. Then, the output is

$$y(t) = 1 - \frac{e^{-\zeta\omega_o t}}{\sqrt{1-\zeta^2}} \sin(\omega_d t + \theta), \quad t \geq 0$$

where $\omega_d := \omega_o\sqrt{1-\zeta^2}$ and $\theta := \cos^{-1}(\zeta)$. For some typical values of ζ, the step response $y(t)$ is as shown in Figure 6.4. The maximum *percent overshoot* is defined to be the quantity

$$\mathrm{PO} := \frac{y_p - y_{\mathrm{ss}}}{y_{\mathrm{ss}}} \times 100\%$$

where y_p is the peak value. By simple calculations it can be seen that the peak value of $y(t)$ occurs at the time instant $t_p = \pi/\omega_d$, and

$$\mathrm{PO} = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \times 100\%$$

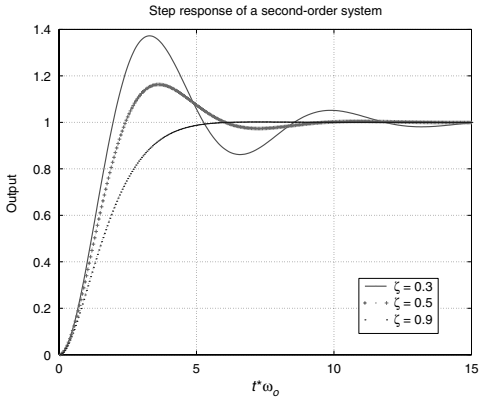


FIGURE 6.4 Step response of a second-order system.

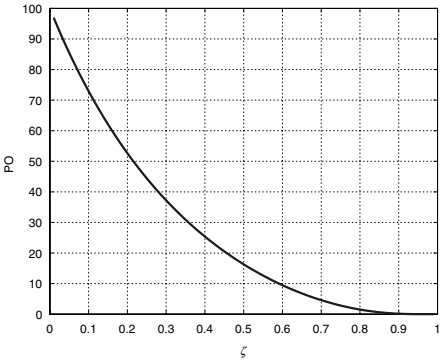


FIGURE 6.5 PO versus ζ.

Figure 6.5 shows PO versus ζ. The *settling time* is defined to be the smallest time instant t_s, after which the response $y(t)$ remains within 2% of its final value, that is,

$$t_s := \min\{\, t' \colon \left| y(t) - y_{ss} \right| \le 0.02\, y_{ss} \ \forall t \ge t' \,\}$$

Sometimes 1% or 5% is used in the definition of settling time instead of 2%; conceptually, there is no difference. For the second-order system response, we have

$$t_s \approx \frac{4}{\zeta \omega_o}$$

So, in order to have a fast settling response, the product $\zeta\omega_o$ should be large.

The closed-loop system poles are

$$r_{1,2} = -\zeta\omega_o \pm j\omega_o\sqrt{1-\zeta^2}$$

Therefore, once the maximum allowable settling time and PO are specified, we can define the region of desired pole locations by determining the minimum allowable ζ and $\zeta\omega_o$. For example, let the desired PO and t_s be bounded by

$$\text{PO} \le 10\% \quad \text{and} \quad t_s \le 8\,\text{s}$$

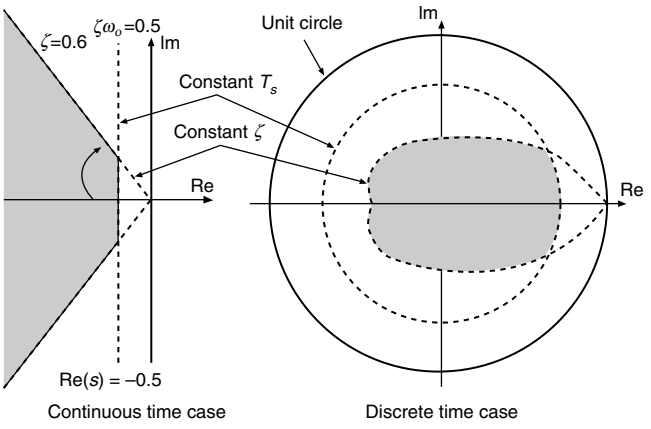


FIGURE 6.6 Region of the desired closed-loop poles.

The PO requirement implies that $\zeta \geq 0.6$, equivalently $\theta \leq 53°$ (recall that $\cos(\theta) = \zeta$). The settling time requirement is satisfied if and only if $\mathrm{Re}(r_{1,2}) \leq -0.5$. Then, the region of desired closed-loop poles is the shaded area shown in Figure 6.6. The same figure also illustrates the region of desired closed-loop poles for similar design requirements in the discrete time case.

If the order of the closed-loop transfer function $T(s)$ is higher than two, then, depending on the location of its poles and zeros, it may be possible to approximate the closed-loop step response by the response of a second-order system. For example, consider the third-order system

$$T(s) = \frac{\omega_o^2}{(s^2 + 2\zeta\omega_o s + \omega_o^2)(1 + s/r)} \quad \text{where } r \gg \zeta\omega_o$$

The transient response contains a term e^{-rt}. Compared with the envelope $e^{-\zeta\omega_o t}$ of the sinusoidal term, e^{-rt} decays very fast, and the overall response is similar to the response of a second-order system. Hence, the effect of the third pole $r_3 = -r$ is negligible.

Consider another example,

$$T(s) = \frac{\omega_o^2[1 + s/(r+e)]}{(s^2 + 2\zeta\omega_o s + \omega_o^2)(1 + s/r)} \quad \text{where } 0 < e \ll r$$

In this case, although r does not need to be much larger than $\zeta\omega_o$, the zero at $-(r + \square)$ cancels the effect of the pole at $-r$. To see this, consider the partial fraction expansion of $Y(s) = T(s)R(s)$ with $R(s) = 1/s$:

$$Y(s) = \frac{A_0}{s} + \frac{A_1}{s - r_1} + \frac{A_2}{s - r_2} + \frac{A_3}{s + r}$$

where $A_0 = 1$ and

$$A_3 = \lim_{s \to -r} (s + r)Y(s) = \frac{\omega_o^2}{2\zeta\omega_o r - (\omega_o^2 + r^2)}\left(\frac{e}{r + e}\right)$$

Since $|A_3| \to 0$ as $e \to 0$, the term $A_3 e^{-rt}$ is negligible in $y(t)$.

In summary, if there is an approximate pole–zero cancellation in the left half plane, then this pole–zero pair can be taken out of the transfer function $T(s)$ to determine PO and t_s. Also, the poles closest to the

imaginary axis dominate the transient response of $y(t)$. To generalize this observation, let $r_1, \ldots, r_n$ be the poles of $T(s)$, such that $\mathrm{Re}(r_k) \ll \mathrm{Re}(r_2) = \mathrm{Re}(r_1) < 0$, for all $k \geq 3$. Then, the pair of complex conjugate poles $r_{1,2}$ are called the *dominant poles*. We have seen that the desired transient response properties, for example, PO and t_s, can be translated into requirements on the location of the dominant poles.

6.3 Root Locus Construction

As mentioned above, the root locus primarily deals with finding the roots of a characteristic polynomial that is an affine function of a single parameter, K,

$$\chi(s) = D(s) + KN(s) \tag{6.5}$$

where $D(s)$ and $N(s)$ are fixed monic polynomials (i.e., coefficient of the highest power is normalized to 1). If N and/or D are not monic, the highest coefficient(s) can be absorbed into K.

6.3.1 Root Locus Rules

Recall that the usual root locus shows the locations of the closed-loop system poles as K varies from 0 to $+\infty$. The roots of $D(s)$, $p_1, \ldots, p_n$, are the poles, and the roots of $N(s)$, $z_1, \ldots, z_m$, are the zeros, of the open-loop system, $G(s) = KF(s)$. Since $P(s)$ and $C(s)$ are proper, $G(s)$ is proper, and hence $n \geq m$. So the degree of the polynomial $\chi(s)$ is n and it has exactly n roots.

Let the closed-loop system poles, that is, roots of $\chi(s)$, be denoted by $r_1(K), \ldots, r_n(K)$. Note that these are functions of K; whenever the dependence on K is clear, they are simply written as $r_1, \ldots, r_n$. The points in $\mathbf{C}$ that satisfy (6.5) for some $K > 0$ are on the root locus. Clearly, a point $r \in \mathbf{C}$ is on the root locus if and only if

$$K = -\frac{1}{F(r)} \tag{6.6}$$

The condition (6.6) can be separated into two parts:

$$|K| = -\frac{1}{|F(r)|} \tag{6.7}$$

$$\angle K = 0^\circ = -(2, +1) \times 180^\circ - \angle F(r), \quad , = 0, \pm 1, \pm 2, \ldots \ . \tag{6.8}$$

The phase rule (6.8) determines the points in $\mathbf{C}$ that are on the root locus. The magnitude rule (6.7) determines the gain $K > 0$ for which the root locus is at a given point r. By using the definition of $F(s)$, (6.8) can be rewritten as

$$(2, +1) \times 180^\circ = \sum_{i=1}^{n} \angle(r - p_i) - \sum_{i=1}^{m} \angle(r - z_j) \tag{6.9}$$

Similarly, (6.7) is equivalent to

$$K = \frac{\prod_{i=1}^{n} |r - p_i|}{\prod_{j=1}^{m} |r - z_j|} \tag{6.10}$$

6.3.2 Root Locus Construction

There are several software packages available for generating the root locus automatically for a given $F = N/D$. In particular, the related MATLAB commands are `rlocus` and `rlocfind`. In many cases, approximate root locus can be drawn by hand using the rules given below. These rules are determined from the basic definitions (6.5), (6.7), and (6.8).

1. The root locus has n branches: $r_1(K), \ldots, r_n(K)$.
2. Each branch starts $(K \cong 0)$ at a pole p_i and ends (as $K \to \infty$) at a zero z_j, or converges to an asymptote, $Me^{j\alpha_{,}}$, where $M \to \infty$ and

$$\alpha_{,} = \frac{2,+1}{n-m} \times 180^\circ, \quad , = 0, \ldots, (n-m-1)$$

3. There are $(n-m)$ asymptotes with angles α. The center of the asymptotes (i.e., their intersection point on the real axis) is

$$\sigma_a = \frac{\sum_{i=1}^{n} p_i - \sum_{j=1}^{m} z_j}{n-m}$$

4. A point $x \in \mathbf{R}$ is on the root locus if and only if the total number of poles p_i's and zeros z_j's to the right of x (i.e., total number of p_i's with $\mathrm{Re}(p_i) > x$ plus total number of z_j's with $\mathrm{Re}(z_j) > x$) is odd. Since $F(s)$ is a rational function with real coefficients, poles and zeros appear in complex conjugates, so when counting the number of poles and zeros to the right of a point $x \in \mathbf{R}$ we just need to consider the poles and zeros on the real axis.
5. The values of K for which the root locus crosses the imaginary axis can be determined from the Routh–Hurwitz stability test. Alternatively, we can set $s = j\omega$ in (6.5) and solve for real ω and K satisfying

$$D(j\omega) + KN(j\omega) = 0$$

   Note that there are two equations here, one for the real part and one for the imaginary part.
6. The break points (intersection of two branches on the real axis) are feasible solutions (satisfying rule 4) of

$$\frac{d}{ds}F(s) = 0 \tag{6.11}$$

7. Angles of departure $(K \cong 0)$ from a complex pole, or arrival $(K \to +\infty)$ to a complex zero, can be determined from the phase rule. See example below.
   Let us now follow the above rules step by step to construct the root locus for

$$F(s) = \frac{(s+3)}{(s-1)(s+5)(s+4+j2)(s+4-j2)}$$

First, enumerate the poles and zeros as $p_1 = -4 + j2$, $p_2 = -4 - j2$, $p_3 = -5$, $p_4 = 1$, $z_1 = -3$. So, $n = 4$ and $m = 1$.

1. The root locus has four branches.
2. Three branches converge to the asymptotes whose angles are 60°, 180°, and −60°, and one branch converges to $z_1 = -3$.

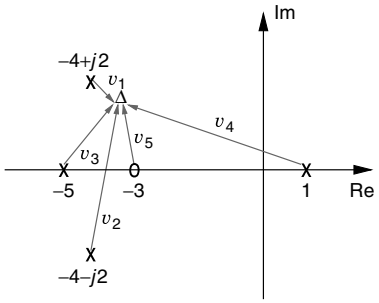


FIGURE 6.7 Angle of departure from $-4 + j2$.

3. The center of the asymptotes is $\sigma = (-12 + 3)/3 = -3$.
4. The intervals $(-\infty, -5]$ and $[-3, 1]$ are on the root locus.
5. The imaginary axis crossings are the feasible roots of

$$(\omega^4 - j12\omega^3 - 47\omega^2 + j40\omega - 100) + K(j\omega + 3) = 0 \tag{6.12}$$

for real ω and K. Real and imaginary parts of (6.12) are

$$\omega^4 - 47\omega^2 - 100 + 3K = 0$$
$$j\omega(-12\omega^2 + 40 + K) = 0$$

They lead to two feasible pairs of solutions ($K = 100/3$, $\omega = 0$) and ($K = 215.83$, $\omega = \pm 4.62$).

6. Break points are the feasible solutions of

$$3s^4 + 36s^3 + 155s^2 + 282s + 220 = 0$$

Since the roots of this equation are $-4.55 \pm j1.11$ and $-1.45 \pm j1.11$, there is no solution on the real axis, hence no break points.

7. To determine the angle of departure from the complex pole $p_1 = -4 + j2$, let Δ represent a point on the root locus near the complex pole p_1, and define v_i, $i = 1, \ldots, 5$, to be the vectors drawn from p_i, for $i = 1, \ldots, 4$, and from z_1 for $i = 5$, as shown in Figure 6.7. Let $\theta_1, \ldots, \theta_5$ be the angles of $v_1, \ldots, v_5$. The phase rule implies

$$(\theta_1 + \theta_2 + \theta_3 + \theta_4) - \theta_5 = \pm 180^\circ \tag{6.13}$$

As Δ approaches p_1, θ_1 becomes the angle of departure and the other θ_i's can be approximated by the angles of the vectors drawn from the other poles, and from the zero, to the pole p_1. Thus θ_1 can be solved from (6.13), where $\theta_2 \approx 90^\circ$, $\theta_3 \approx \tan^{-1}(2)$, $\theta_4 \approx 180^\circ - \tan^{-1}\left(\frac{2}{5}\right)$, and $\theta_5 \approx 90^\circ + \tan^{-1}\left(\frac{1}{2}\right)$. That yields $\theta_1 \approx -15^\circ$.

The exact root locus for this example is shown in Figure 6.8. From the results of item 5 above, and the shape of the root locus, it is concluded that the feedback system is stable if

$$33.33 < K < 215.83$$

that is, by simply adjusting the gain of the controller, the system can be made stable. In some situations we need to use a dynamic controller to satisfy all the design requirements.

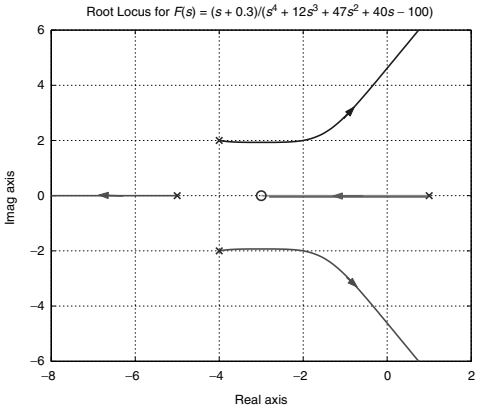


FIGURE 6.8 Root locus for $F(s) = \dfrac{(s+3)}{(s-1)(s+5)(s+4+j2)(s+4-j2)}$.

6.3.3 Design Examples

Example 1

Consider the standard feedback system with a plant

$$P(s) = \frac{1}{0.72}\frac{1}{(s+1)(s+2)}$$

and design a controller such that

- Feedback system is stable.
- PO $\leq 10\%$, $t_s \leq 4$ s, and steady state error is zero when $r(t)$ is unit step.
- Steady state error is as small as possible when $r(t)$ is unit ramp.

It is clear that the second design goal cannot be achieved by a simple proportional controller. To satisfy this condition, the controller must have a pole at $s = 0$, that is, it must have integral action. If we try an integral control of the form $C(s) = K_c/s$, with $K_c > 0$, then the root locus has three branches, the interval $[-1, 0]$ is on the root locus; three asymptotes have angles $\{60°, 180°, -60°\}$ with a center at $\sigma_a = -1$; and there is only one break point at $-1 + \frac{1}{\sqrt{3}}$, see Figure 6.9. From the location of the break point, center, and angles of the asymptotes, it can be deduced that two branches (one starting at $p_1 = -1$, and the other one starting at $p_3 = 0$) always remain to the right of p_1. On the other hand, the settling time condition implies that the real parts of the dominant closed-loop system poles must be less than or equal to -1. So, a simple integral control does not do the job. Now try a PI controller of the form

$$C(s) = K_c\left(\frac{s - z_c}{s}\right), \quad K_c > 0$$

In this case, we can select $z_c = -1$ to cancel the pole at $p_1 = -1$ and the system effectively becomes a second-order system. The root locus for $F(s) = 1/s(s + 2)$ has two branches and two asymptotes, with center $\sigma_a = -1$ and angles $\{90°, -90°\}$; the break point is also at -1. The branches leave -2 and 0, and go toward each other, meet at -1, and tend to infinity along the line $\mathrm{Re}(s) = -1$. Indeed, the closed-loop system poles are

$$r_{1,2} = -1 \pm \sqrt{1 - K}, \quad \text{where } K = K_c/0.72$$

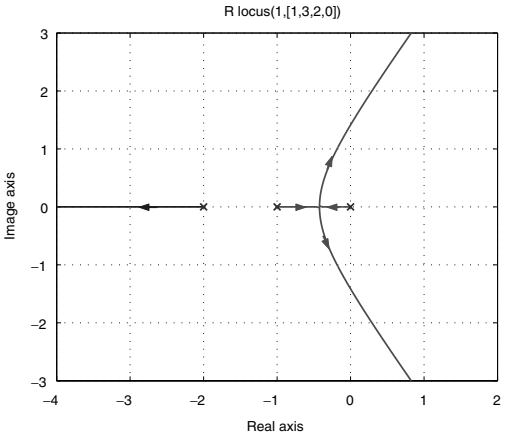


FIGURE 6.9 Root locus for Example 1.

The steady state error, when $r(t)$ is unit ramp, is $2/K$. So K needs to be as large as possible to meet the third design condition. Clearly, $\mathrm{Re}(r_{1,2}) = -1$ for all $K \geq 1$, which satisfies the settling time requirement. The percent overshoot is less than 10% if ζ of the roots $r_{1,2}$ is greater than 0.6. A simple algebra shows that $\zeta = 1/\sqrt{K}$, hence the design conditions are met if $K = 1/0.36$, that is $K_c = 2$. Thus a PI controller that solves the design problem is

$$C(s) = 2\left(\frac{s+1}{s}\right)$$

The controller cancels a stable pole (at $s = -1$) of the plant. If there is a slight uncertainty in this pole location, perfect cancellation will not occur and the system will be third-order with the third pole at $r_3 \cong -1$. Since the zero at $z_o = -1$ will approximately cancel the effect of this pole, the response of this system will be close to the response of a second-order system. However, we must be careful if the pole–zero cancellations are near the imaginary axis because in this case small perturbations in the pole location might lead to large variations in the feedback system response, as illustrated with the next example.

Example 2

A flexible structure with lightly damped poles has transfer function in the form

$$P(s) = \frac{\omega_1^2}{s^2(s^2 + 2\zeta\omega_1 s + \omega_1^2)}$$

By using the root locus, we can see that the controller

$$C(s) = K_c \frac{(s^2 + 2\zeta\omega_1 s + \omega_1^2)(s + 0.4)}{(s + r)^2(s + 4)}$$

stabilizes the feedback system for sufficiently large r and an appropriate choice of K_c. For example, let $\omega_1 = 2$, $\zeta = 0.1$, and $r = 10$. Then the root locus of $F(s) = P(s)C(s)/K$, where $K = K_c\omega_1^2$, is as shown in Figure 6.10. For $K = 600$, the closed-loop system poles are

$$\{-10.78 \pm j2.57,\ -0.94 \pm j1.61,\ -0.2 \pm j1.99,\ -0.56\}$$

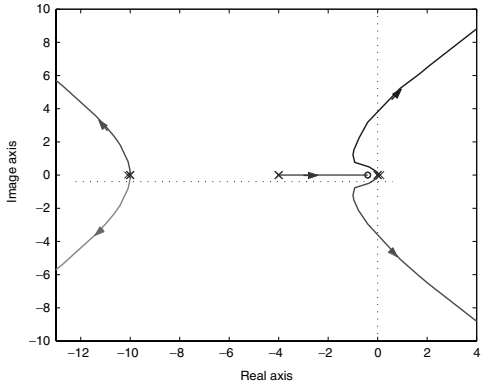


FIGURE 6.10 Root locus for Example 2(a).

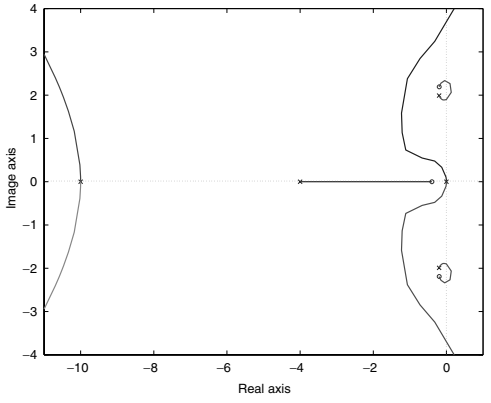


FIGURE 6.11 Root locus for Example 2(b).

Since the poles $-0.2 \pm j1.99$ are canceled by a pair of zeros at the same point in the closed-loop system transfer function $T = G(1 + G)^{-1}$, the dominant poles are at -0.56 and $-0.94 \pm j1.61$ (they have relatively large negative real parts and the damping ratio is about 0.5).

Now, suppose that this controller is fixed and the complex poles of the plant are slightly modified by taking $\zeta = 0.09$ and $\omega_1 = 2.2$. The root locus corresponding to this system is as shown in Figure 6.11. Since lightly damped complex poles are not perfectly canceled, there are two more branches near the imaginary axis. Moreover, for the same value of $K = 600$, the closed-loop system poles are

$$\{-10.78 \pm j2.57, -1.21 \pm j1.86, 0.05 \pm j1.93, -0.51\}$$

In this case, the feedback system is unstable.

Example 3

One of the most important examples of mechatronic systems is the DC motor. An approximate transfer function of a DC motor [8, pp. 141–143] is in the form

$$P_m(s) = \frac{K_m}{s(s + 1/\tau_m)}, \quad \tau_m > 0$$

Also note that if τ_m is large, then $P_m(s) \approx P_b(s)$, where $P_b(s) = K_b/s^2$ is the transfer function of a rigid beam. In this example, the general class of plants $P_m(s)$ will be considered. Assuming that $p_m = -1/\tau_m$ and K_m are given, a first-order controller

$$C(s) = K_c\left(\frac{s - z_c}{s - p_c}\right) \tag{6.14}$$

will be designed. The aim is to place the closed-loop system poles far from the Im-axis. Since the order of $F(s) = P_m(s)C(s)/K_mK_c$ is three, the root locus has three branches. Suppose the desired closed-loop poles are given as p_1, p_2, and p_3. Then, the pole placement problem amounts to finding $\{K_c, z_c, p_c\}$ such that the characteristic equation is

$$\begin{aligned}\chi(s) &= (s - p_1)(s - p_2)(s - p_3)\\ &= s^3 - (p_1 + p_2 + p_3)s^2 + (p_1p_2 + p_1p_3 + p_2p_3)s - p_1p_2p_3\end{aligned}$$

But the actual characteristic equation, in terms of the unknown controller parameters, is

$$\begin{aligned}\chi(s) &= s(s - p_m)(s - p_c) + k(s - z_c)\\ &= s^3 - (p_m + p_c)s^2 + (p_mp_c + K)s - Kz_c\end{aligned}$$

where $K := K_mK_c$. Equating the coefficients of the desired $\chi(s)$ to the coefficients of the actual $\chi(s)$, three equations in three unknowns are obtained:

$$\begin{aligned}p_m + p_c &= p_1 + p_2 + p_3\\ p_mp_c + K &= p_1p_2 + p_1p_3 + p_2p_3\\ Kz_c &= p_1p_2p_3\end{aligned}$$

From the first equation p_c is determined, then K is obtained from the second equation, and finally z_c is computed from the third equation.

For different numerical values of p_m, p_1, p_2, and p_3 the shape of the root locus is different. Below are some examples, with the corresponding root loci shown in Figures 6.12–6.14.

(a) $p_m = -0.05$, $p_1 = p_2 = p_3 = -2 \Rightarrow$

$$K = 11.70, \quad p_c = -5.95, \quad z_c = -0.68$$

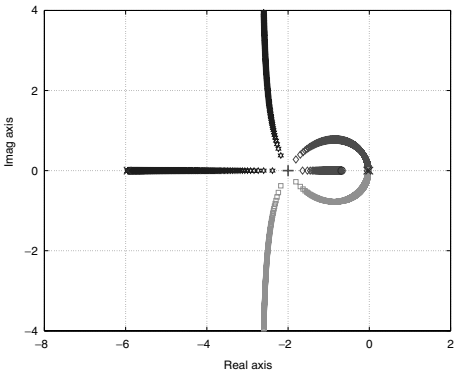


FIGURE 6.12 Root locus for Example 3(a).

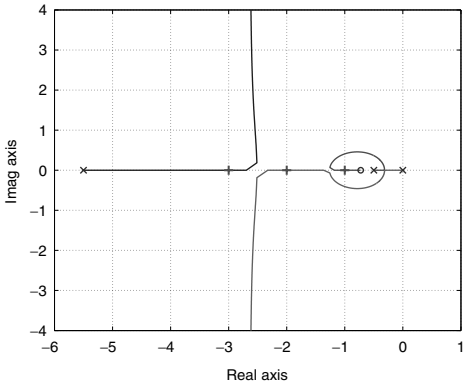


FIGURE 6.13 Root locus for Example 3(b).

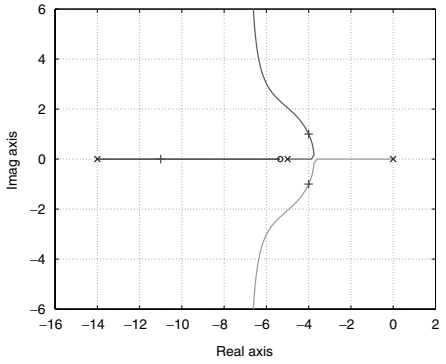


FIGURE 6.14 Root locus for Example 3(c).

(b) $p_m = -0.5,\ p_1 = -1,\ p_2 = -2,\ p_3 = -3 \Rightarrow$

$$K = 8.25, \quad p_c = -5.50, \quad z_c = -0.73$$

(c) $p_m = -5,\ p_1 = -11,\ p_2 = -4 + j1,\ p_3 = -4 - j1 \Rightarrow$

$$K = 35, \quad p_c = -14, \quad z_c = -5.343$$

Example 4

Consider the open-loop transfer function

$$P(s)C(s) = K_c \frac{(s^2 - 3s + 3)(s - z_c)}{s(s^2 + 3s + 3)(s - p_c)}$$

where K_c is the controller gain to be adjusted, and z_c and p_c are the controller zero and pole, respectively. Observe that the root locus has four branches except for the non-generic case $z_c = p_c$. Let the desired dominant closed-loop poles be $r_{1,2} = -0.4$. The steady state error for unit ramp reference input is

$$e_{ss} = \frac{p_c}{K_c z_c}$$

Accordingly, we want to make the ratio $K_c z_c / p_c$ as large as possible.

The characteristic equation is

$$\chi(s) = s(s^2 + 3s + 3)(s - p_c) + K_c(s^2 - 3s + 3)(s - z_c)$$

and it is desired to be in the form

$$\chi(s) = (s + 0.4)^2 (s - r_3)(s - r_4)$$

for some $r_{3,4}$ with $\mathrm{Re}(r_{3,4}) < 0$, which implies that

$$\chi(s)|_{s=-0.4} = 0, \quad \frac{d}{ds}\chi(s)|_{s=-0.4} = 0 \tag{6.15}$$

Conditions (6.15) give two equations:

$$0.784(0.4 + p_c) - 4.36K_c(0.4 + z_c) = 0$$
$$4.36K_c - 0.784 - 1.08(0.4 + p_c) + 3.8K_c(0.4 + z_c) = 0$$

from which z_c and p_c can be solved in terms of K_c. Then, by simple substitutions, the ratio to be maximized, $K_c z_c / p_c$, can be reduced to

$$\frac{K_c z_z}{p_c} = \frac{3.4776\,K_c - 0.784}{24.2469\,K_c - 3.4776}$$

The maximizing value of K_c is 0.1297; it leads to $p_c = -0.9508$ and $z_c = -1.1637$. For this controller, the feedback system poles are

$$\{-1.64 + j0.37,\ -1.64 - j0.37,\ -0.40,\ -0.40\}$$

The root locus is shown in Figure 6.15.

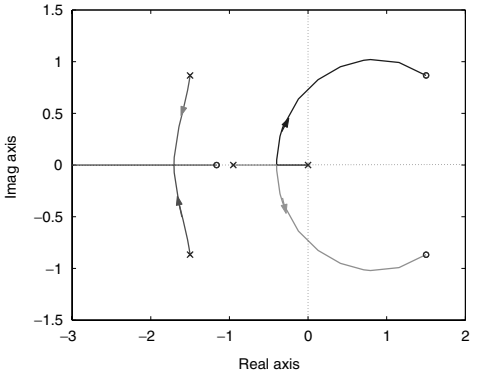


FIGURE 6.15 Root locus for Example 4.

6.4 Complementary Root Locus

In the previous section, the root locus parameter K was assumed to be positive and the phase and magnitude rules were established based on this assumption. There are some situations in which controller gain can be negative as well. Therefore, the complete picture is obtained by drawing the usual root locus (for $K > 0$) and the complementary root locus (for $K < 0$). The complementary root locus rules are

$$\times 360^\circ = \sum_{i=1}^{n} \angle(r - p_i) - \sum_{j=1}^{m} \angle(r - z_j), \quad , = 0, \pm 1, \pm 2, \ldots \tag{6.16}$$

$$|K| = \frac{\prod_{i=1}^{n} |r - p_i|}{\prod_{j=1}^{m} |r - z_j|} \tag{6.17}$$

Since the phase rule (6.16) is the 180° shifted version of (6.9), the complementary root locus is obtained by simple modifications in the root locus construction rules. In particular, the number of asymptotes and their center are the same, but their angles α's are given by

$$\alpha_{,} = \frac{2,}{(n-m)} \times 180^\circ, \quad , = 0, \ldots, (n - m - 1)$$

Also, an interval on the real axis is on the complementary root locus if and only if it is not on the usual root locus.

Example 3 (revisited)

In the Example 3 given above, if the problem data is modified to $p_m = -5$, $p_1 = -20$, and $p_{2,3} = -2 \pm j$, then the controller parameters become

$$K = -10, \quad p_c = -19, \quad z_c = 10$$

Note that the gain is negative. The roots of the characteristic equation as K varies between 0 and $-\infty$ form the complementary root locus; see Figure 6.16.

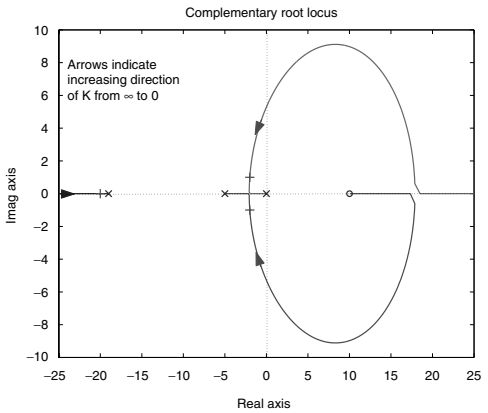


FIGURE 6.16 Complementary root locus for Example 3.

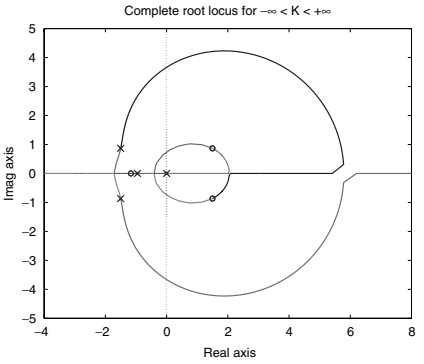


FIGURE 6.17 Complementary and usual root loci for Example 4.

Example 4 (revisited)

In this example, if K increases from $-\infty$ to $+\infty$, the closed-loop system poles move along the complementary root locus, and then the usual root locus, as illustrated in Figure 6.17.

6.5 Root Locus for Systems with Time Delays

The standard feedback control system considered in this section is shown in Figure 6.18, where the controller C and plant P are in the form

$$C(s) = \frac{N_c(s)}{D_c(s)}$$

and

$$P(s) = e^{-hs}P_0(s) \quad \text{where } P_0(s) = \frac{N_p(s)}{D_p(s)}$$

with (N_c, D_c) and (N_p, D_p) being coprime pairs of polynomials with real coefficients.* The term e^{-hs} is the transfer function of a pure delay element (in Figure 6.18 the plant input is delayed by h seconds). In general, time delays enter into the plant model when there is

- Sensor (or actuator) processing delay
- Software delay in the controller
- Transport delay in the process

In this case the open-loop transfer function is

$$G(s) = G_h(s) = e^{-hs}G_0(s)$$

where $G_0(s) = P_0(s)C(s)$ corresponds to the no delay case, $h = 0$.

Note that magnitude and phase of $G(j\omega)$ are determined from the identities

$$|G(j\omega)| = |G_0(j\omega)| \tag{6.18}$$

$$\angle G(j\omega) = -h\omega + \angle G_0(j\omega) \tag{6.19}$$

*A pair of polynomials is said to be coprime pair if they do not have common roots.

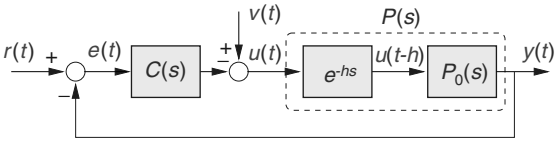


FIGURE 6.18 Feedback system a with time delay.

6.5.1 Stability of Delay Systems

Stability of the feedback system shown in Figure 6.18 is equivalent to having all the roots of

$$\chi(s) = D(s) + e^{-hs}N(s) \tag{6.20}$$

in the open left half plane, $\mathbf{C}_-$, where $D(s) = D_c(s)D_p(s)$ and $N(s) = N_c(s)N_p(s)$. We assume that there is no unstable pole–zero cancellation in taking the product $P_0(s)C(s)$, and that $deg(D) > deg(N)$ (here N and D need not be monic polynomials). Strictly speaking, $\chi(s)$ is not a polynomial because it is a transcendental function of s. The functions of the form (6.20) belong to a special class of functions called *quasi-polynomials*. The closed-loop system poles are the roots of (6.20).

Following are known facts (see [1,10]):

(i) If r_k is a root of (20), then so is r_k (i.e., roots appear in complex conjugate pairs as usual).
(ii) There are infinitely many poles $r_k \in \mathbf{C}$, $k = 1, 2, \ldots$, satisfying $\chi(r_k) = 0$.
(iii) And r_k's can be enumerated in such a way that $\mathrm{Re}(r_k + 1) \le \mathrm{Re}(r_k)$; moreover, $\mathrm{Re}(r_k) \to -\infty$ as $k \to \infty$.

Example

If $G_h(s) = e^{-hs}/s$, then the closed-loop system poles r_k, for $k = 1, 2, \ldots$, are the roots of

$$1 + \frac{e^{-h\sigma_k}e^{-jh\omega_k}}{\sigma_k + j\omega_k}e^{\pm j2k\pi} = 0 \tag{6.21}$$

where $r_k = \sigma_k + j\omega_k$ for some $\sigma_k, \omega_k \in \mathbf{R}$. Note that $e^{\pm j2k\pi} = 1$ for all $k = 1, 2, \ldots$. Equation (6.1) is equivalent to the following set of equations:

$$e^{-h\sigma_k} = |\sigma_k + j\omega_k| \tag{6.22}$$

$$\pm(2k-1)\pi = h\omega_k + \angle(\sigma_k + j\omega_k), \quad k = 1,2,\ldots \tag{6.23}$$

It is quite interesting that for $h = 0$ there is only one root $r = -1$, but even for infinitesimally small $h > 0$ there are infinitely many roots. From the magnitude condition (6.22), it can be shown that

$$\sigma_k \ge 0 \Rightarrow |\omega_k| \le 1 \tag{6.24}$$

Also, for $\sigma_k \ge 0$, the phase $\angle(\sigma_k + j\omega_k)$ is between $-\pi/2$ and $+\pi/2$, therefore (6.23) leads to

$$\sigma_k \ge 0 \Rightarrow h|\omega_k| \ge \frac{\pi}{2} \tag{6.25}$$

By combining (6.24) and (6.25), it can be proven that the feedback system has no roots in the closed right half plane when $h < \pi/2$. Furthermore, the system is unstable if $h \geq \pi/2$. In particular, for $h = \pi/2$ there are two roots on the imaginary axis, at $\pm j1$. It is also easy to show that, for any $h > 0$ as $k \to \infty$, the roots converge to

$$r_k \to \frac{1}{h}\left[-\ln\left(\frac{2k\pi}{h}\right) \pm j2k\pi\right]$$

As $h \to 0$, the magnitude of the roots converge to ∞.

As illustrated by the above example, property (iii) implies that for any given real number σ there are only finitely many r_k's in the region of the complex plane

$$\mathrm{C}_\sigma := \{s \in \mathrm{C}: \mathrm{Re}(s) \geq \sigma\}$$

In particular, with $\sigma = 0$, this means that the quasi-polynomial $\chi(s)$ can have only finitely many roots in the right half plane. Since the effect of the closed-loop system poles that have very large negative real parts is negligible (as far as closed-loop systems' input–output behavior is concerned), only finitely many "dominant" roots r_k, for $k = 1,\ldots,m$, should be computed for all practical purposes.

6.5.2 Dominant Roots of a Quasi-Polynomial

Now we discuss the following problem: given $N(s)$, $D(s)$, and $h \geq 0$, find the dominant roots of the quasi-polynomial

$$\chi(s) = D(s) + e^{-hs}N(s)$$

For each fixed $h > 0$, it can be shown that there exists $\sigma_{\max}$ such that $\chi(s)$ has no roots in the region $\mathrm{C}_{\sigma_{\max}}$, see [11] for a simple algorithm to estimate $\sigma_{\max}$, based on Nyquist criterion. Given $h > 0$ and a region of the complex plane defined by $\sigma_{\min} \leq \mathrm{Re}(s) \leq \sigma_{\max}$, the problem is to find the roots of $\chi(s)$ in this region.

Clearly, a point $r = \sigma + j\omega$ in $\mathbb{C}$ is a root of $\chi(s)$ if and only if

$$D(\sigma + j\omega) = -e^{-h\sigma}e^{-jh\omega}N(\sigma + j\omega)$$

Taking the magnitude square of both sides of the above equation, $\chi(r) = 0$ implies

$$A_\sigma(x) := D(\sigma + x)D(\sigma - x) - e^{-2h\sigma}N(\sigma + x)N(\sigma - x) = 0$$

where $x = j\omega$. The term $D(\sigma + x)$ stands for the function $D(s)$ evaluated at $\sigma + x$. The other terms of $A_\sigma(x)$ are calculated similarly. For each fixed σ, the function $A_\sigma(x)$ is a polynomial in the variable x. By symmetry, if x is a zero of $A_\sigma(\cdot)$, then $(-x)$ is also a zero.

If $A_\sigma(x)$ has a root x whose real part is zero, set $r = \sigma + x$. Next, evaluate the magnitude of $\chi(r)$; if it is zero, then n is a root of $\chi(s)$. Conversely, if $A_\sigma(x)$ has no root on the imaginary axis, then $\chi(s)$ cannot have a root whose real part is the fixed value of σ from which $A_\sigma(\cdot)$ is constructed.

6.5.2.1 Algorithm

Given $N(s)$, $D(s)$, h, $\sigma_{\min}$, and $\sigma_{\max}$:

Step 1. Pick σ values $\sigma_1,\ldots,\sigma_M$ between $\sigma_{\min}$ and $\sigma_{\max}$ such that $\sigma_{\min} = \sigma_1$, $\sigma_i < \sigma_{i+1}$, and $\sigma_M = \sigma_{\max}$. For each σ_i perform the following.

Step 2. Construct the polynomial $A_i(x)$ according to

$$A_i(x) := D(\sigma_i + x)D(\sigma_i - x) - e^{-2h\sigma_i}N(\sigma_i + x)N(\sigma_i - x)$$

Step 3. For each imaginary axis roots x of A_i, perform the following test:
Check if $|\chi(\sigma_i + x)| = 0$; if yes, then $r = \sigma_i + x$ is a root of $\chi(s)$; if not, discard x.
Step 4. If $i = M$, stop; else increase i by 1 and go to Step 2.

Example

We will find the dominant roots of

$$1 + \frac{e^{-hs}}{s} = 0 \tag{6.26}$$

for a set of critical values of h. Recall that (6.26) has a pair of roots $\pm j1$ when $h = \pi/2 = 1.57$. Moreover, dominant roots of (6.26) are in the right half plane if $h > 1.57$, and they are in the left half plane if $h < 1.57$. So, it is expected that for $h \in (1.2, 2.0)$ the dominant roots are near the imaginary axis. Take $\sigma_{\min} = -0.5$ and $\sigma_{\max} = 0.5$, with $M = 400$ linearly spaced σ_i's between them. In this case

$$A_i(x) = \sigma_i^2 - e^{-2h\sigma_i} - x^2$$

Whenever $e^{-2h\sigma_i} \geq \sigma_i^2$, $A_i(x)$ has two roots:

$$x_< = \pm j\sqrt{e^{-2h\sigma_i} - \sigma_i^2}, \quad < = 1, 2$$

For each fixed σ_i satisfying this condition, let $r = \sigma_i + x$ (note that x is a function of σ_i, so r is a function of σ_i) and evaluate

$$f(\sigma_i) := \left|1 + \frac{e^{-hr_<}}{r_<}\right|$$

If $f(\sigma_i) = 0$, then r is a root of (6.26). For 10 different values of $h \in (1.2, 2.0)$, the function $f(\sigma)$ is plotted in Figure 6.19. This figure shows the feasible values of σ_i for which r (defined from σ_i) is a root of (6.26).

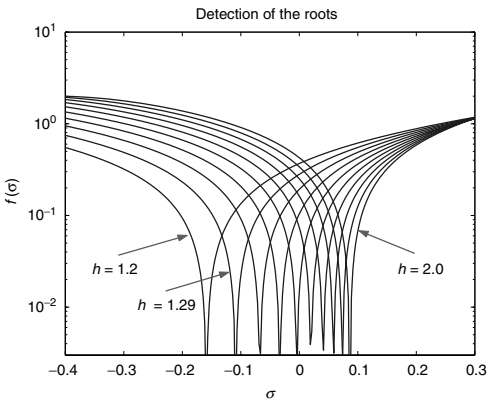


FIGURE 6.19 Detection of the dominant roots.

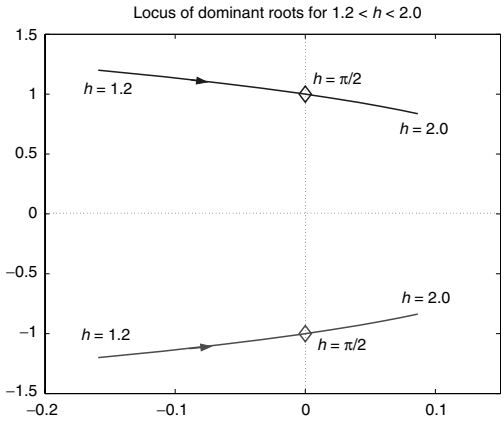


FIGURE 6.20 Dominant roots as h varies from 1.2 to 2.0.

The dominant roots of (6.26), as h varies from 1.2 to 2.0, are shown in Figure 6.20. For $h < 1.57$, all the roots are in C_-. For $h > 1.57$, the dominant roots are in C_+, and for $h = 1.57$, they are at $\pm j1$.

6.5.3 Root Locus Using Padé Approximations

In this section we assume that $h > 0$ is fixed and we try to obtain the root locus, with respect to uncertain/adjustable gain K, corresponding to the dominant poles. The problem can be solved by numerically calculating the dominant roots of the quasi-polynomial

$$\chi(s) = D(s) + KN(s)e^{-hs} \tag{6.27}$$

for varying K, by using the methods presented in the previous section. In this section an alternative method is given that uses Padé approximation of the time delay term e^{-hs}. More precisely, the idea is to find polynomials $N_h(s)$ and $D_h(s)$ satisfying

$$e^{-hs} \approx \frac{N_h(s)}{D_h(s)} \tag{6.28}$$

so that the dominant roots

$$D(s)D_h(s) + KN(s)N_h(s) = 0 \tag{6.29}$$

closely match the dominant roots of $\chi(s)$, (6.27). How should we do the approximation (6.28) for this match?

By using the stability robustness measures determined from the Nyquist stability criterion, we can show that for our purpose we may consider the following cost function in order to define a meaningful measure for the approximation error:

$$\Delta_h =: \sup_{\omega} \left| \frac{K_{\max} N(j\omega)}{D(j\omega)} \right| \left| e^{-jh\omega} - \frac{N_h(j\omega)}{D_h(j\omega)} \right|$$

where $K_{\max}$ is the maximum value of interest for the uncertain/adjustable parameter K.

The th order Padé approximation is defined as follows:

$$N_h(s) = \sum_{k=0}^{} (-1)^k c_k h^k s^k$$

$$D_h(s) = \sum_{k=0}^{} c_k h^k s^k$$

where coefficients c_k's are computed from

$$c_k = \frac{(2,-k)!,!}{2,!\,k!(,-k)!}, \quad k = 0, 1, \ldots,,$$

First- and second-order approximations are in the form

$$\frac{N_h(s)}{D_h(s)} = \begin{cases} \dfrac{1 - hs/2}{1 + hs/2}, & ,= 1 \\ \dfrac{1 - hs/2 + (hs)^2/12}{1 + hs/2 + (hs)^2/12}, & ,= 2 \end{cases}$$

Given the problem data $\{h, K_{\max}, N(s), D(s)\}$, how do we find the smallest degree, , of the Padé approximation, so that $\Delta_h \le \delta$ (or $\Delta_h / K_{\max} \le \delta'$) for a specified error δ, or a specified relative error δ'? The answer lies in the following result [7]: for a given degree of approximation we have

$$\left| e^{-jh\omega} - \frac{N_h(j\omega)}{D_h(j\omega)} \right| \le \begin{cases} 2\left(\dfrac{eh\omega}{4,}\right)^{2,+1}, & \omega \le \dfrac{4,}{eh} \\ 2, & \omega \ge \dfrac{4,}{eh} \end{cases}$$

In light of this result, we can solve the approximation order selection problem by using the following procedure:

1. Determine the frequency ω_x such that

$$\left| \frac{K_{\max} N(j\omega)}{D(j\omega)} \right| \le \frac{\delta}{2}, \quad \text{for all } \omega \ge \omega_x$$

   and initialize = 1.
2. For each ≥ 1 define

$$\omega_< = \max\left\{ \omega_x, \frac{4<}{eh} \right\}$$

   and plot the function

$$\Phi_<(\omega) := \begin{cases} 2\left| \dfrac{K_{\max} N(j\omega)}{D(j\omega)} \right| \left(\dfrac{eh\omega}{4<}\right)^{2<+1}, & \text{for } \omega \le \dfrac{4<}{eh} \\ 2\left| \dfrac{K_{\max} N(j\omega)}{D(j\omega)} \right|, & \text{for } \omega_< \ge \omega \ge \dfrac{4<}{eh} \end{cases}$$

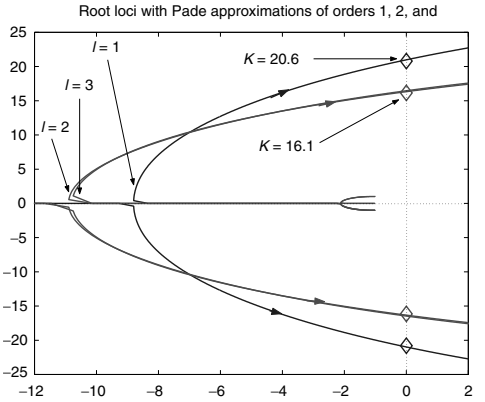


FIGURE 6.21 Dominant root for $\ell = 1$.

3. Check if

$$\max_{\omega \in [0, \omega_x]} \Phi_\ell(\omega) \leq \delta \tag{6.30}$$

If yes, stop, this value of ℓ satisfies the desired error bound: $\Delta_h \leq \delta$. Otherwise, increase ℓ by 1, and go to Step 2. Note that the left-hand side of the inequality (6.30) is an upper bound of Δ_h.

Since we assumed $deg(D) > deg(N)$, the algorithm will pass Step 3 eventually for some finite $\ell \geq 1$. At each iteration, we have to draw the error function $\Phi_\ell(\omega)$ and check whether its peak value is less than δ. Typically, as δ decreases, ω_x increases, and that forces ℓ to increase. On the other hand, for very large values of ℓ, the relative magnitude c_0/c_ℓ of the coefficients becomes very large, in which case numerical difficulties arise in analysis and simulations. Also, as time delay h increases, ℓ should be increased to keep the level of the approximation error δ fixed. This is a fundamental difficulty associated with time delay systems.

Example

Let $N(s) = s + 1$, $D(s) = s^2 + 2s + 2$ and $h = 0.1$, and $K_{\max} = 20$. Then, for $\delta = 0.05$, applying the above procedure we calculate $\ell = 2$ as the smallest approximation degree satisfying $\Delta_h/K_{\max} < \delta$. Therefore, a second-order approximation of the time delay should be sufficient for predicting the dominant poles for $K \in [0, 20]$. Figure 6.21 shows the approximate root loci obtained from Padé approximations of degrees $\ell = 1, 2, 3$. There is a significant difference between the root loci for $\ell = 1$ and $\ell = 2$. In the region $\mathrm{Re}(s) \geq -12$, the predicted dominant roots are approximately the same for $\ell = 2, 3$, for $K \in [0, 20]$. So, we can safely say that using higher order approximations will not make any significant difference as far as predicting the behavior of the dominant poles for the given range of K.

6.6 Notes

This chapter in the handbook is an edited version of related parts of the author's book [9]. More detailed discussions of the root locus method can be found in all the classical control books, such as [2, 5, 6, 8]. As mentioned earlier, extension of this method to discrete time systems is rather trivial: the method to find the roots of a polynomial as a function of a varying real parameter is independent of the variable s (in the continuous time case) or z (in the discrete time case). The only difference between these two cases is the definition of the desired region of the complex plane: for the continuous time systems, this is defined relative to the imaginary axis, whereas for the discrete time systems the region is defined with respect to the unit circle, as illustrated in Figure 6.6.

References

1. Bellman, R. E., and Cooke, K. L., *Differential Difference Equations,* Academic Press, New York, 1963.
2. Dorf, R. C., and Bishop, R. H., *Modern Control Systems,* 9th ed., Prentice-Hall, Upper Saddle River, NJ, 2001.
3. Evans, W. R., "Graphical analysis of control systems," *Transac. Amer. Inst. Electrical Engineers,* vol. 67 (1948), pp. 547–551.
4. Evans, W. R., "Control system synthesis by root locus method," *Transac. Amer. Inst. Electrical Engineers,* vol. 69 (1950), pp. 66–69.
5. Franklin, G. F., Powell, J. D., and Emami-Naeini, A., *Feedback Control of Dynamic Systems,* 3rd ed., Addison Wesley, Reading, MA, 1994.
6. Kuo, B. C., *Automatic Control Systems,* 7th ed., Prentice-Hall, Upper Saddle River, NJ, 1995.
7. Lam, J., "Convergence of a class of Padé approximations for delay systems," *Int. J. Control,* vol. 52 (1990), pp. 989–1008.
8. Ogata, K., *Modern Control Engineering,* 3rd ed., Prentice-Hall, Upper Saddle River, NJ, 1997.
9. Özbay, H., *Introduction to Feedback Control Theory,* CRC Press LLC, Boca Raton, FL, 2000.
10. Stepan, G., *Retarded Dynamical Systems: Stability and Characteristic Functions,* Longman Scientific & Technical, New York, 1989.
11. Ulus, C., "Numerical computation of inner-outer factors for a class of retarded delay systems," *Int. J. Systems Sci.,* vol. 28 (1997), pp. 897–904.

7

Frequency Response Methods

Jyh-Jong Sheen
National Taiwan Ocean University

7.1 Introduction

The analysis and design of industrial control systems are often accomplished utilizing frequency response methods. By the term frequency response, we mean the steady-state response of a linear constant coefficient system to a sinusoidal input test signal. We will see that the response of the system to a sinusoidal input signal is also a sinusoidal output signal at the same frequency as the input. However, the magnitude and phase of the output signal differ from those of the input signal, and the amount of difference is a function of the input frequency. Thus, we will be investigating the relationship between the transfer function and the frequency response of linear stable systems.

Consider a stable linear constant coefficient system shown in Figure 7.1. Using Euler's formula, $e^{j\omega t} = \cos\omega t + j\sin\omega t$, let us assume that the input sinusoidal signal is given by

$$u(t) = U_0 e^{j\omega t} = U_0\cos\omega t + jU_0\sin\omega t \tag{7.1}$$

Taking the Laplace transform of $u(t)$ gives

$$U(s) = \frac{U_0}{s - j\omega} = U_0\frac{s + j\omega}{s^2 + \omega^2} = \frac{U_0 s}{s^2 + \omega^2} + j\frac{U_0\omega}{s^2 + \omega^2} \tag{7.2}$$

The first term in Equation 7.2 is the Laplace transform of $U_0\cos\omega t$, while the second term, without the imaginary number j, is the Laplace transform of $U_0\sin\omega t$.

Suppose that the transfer function $G(s)$ can be written as

$$G(s) = \frac{n(s)}{d(s)} = \frac{n(s)}{(s + p_1)(s + p_2)\cdots(s + p_n)} \tag{7.3}$$

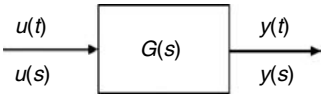


FIGURE 7.1 A stable linear constant coefficient system.

where p_i, $i = 1, 2, \ldots, n$, are assumed to be distinct poles. The Laplace transform of the output $Y(s)$ is then

$$Y(s) = G(s)U(s) = G(s)\frac{U_0}{s - j\omega} \tag{7.4}$$

Taking the partial fraction expansion of $Y(s)$ gives

$$Y(s) = \frac{k_1}{s + p_1} + \cdots + \frac{k_n}{s + p_n} + \frac{\alpha}{s - j\omega} \tag{7.5}$$

The coefficient α can be determined by

$$\alpha = [(s - j\omega)Y(s)]\big|_{s=j\omega} = [U_0 G(s)]\big|_{s=j\omega} = U_0 G(j\omega)$$

Therefore, the inverse Laplace transform of $Y(s)$ yields

$$y(t) = k_1 e^{-p_1 t} + \cdots + k_n e^{-p_n t} + U_0 G(j\omega) e^{j\omega t}, \quad t \geq 0 \tag{7.6}$$

For a stable system, all $-p_i$ have negative nonzero real parts and, therefore, all the terms $k_i e^{-p_i t}$, $i = 1, 2, \ldots, n$, approach zero as t approaches infinity. Thus, at steady state, the output $y(t)$ becomes

$$y_{ss}(t) = \lim_{t \to \infty} y(t) = U_0 G(j\omega) e^{j\omega t} = U_0 |G(j\omega)| e^{j(\omega t + \phi)} \tag{7.7}$$

The sinusoidal transfer function, $G(j\omega)$, is written in exponential form

$$G(j\omega) = |G(j\omega)| e^{j\phi}$$

where

$$|G(j\omega)| = \sqrt{\{\mathrm{Re}[G(j\omega)]\}^2 + \{\mathrm{Im}[G(j\omega)]\}^2} \tag{7.8a}$$

and

$$\phi = \angle G(j\omega) = \tan^{-1} \frac{\mathrm{Im}[G(j\omega)]}{\mathrm{Re}[G(j\omega)]} \tag{7.8b}$$

Equation 7.7 shows that for a stable system subject to a sinusoidal input, the steady-state response is a sinusoidal output of the same frequency as the input. The amplitude of the output is that of the input times $|G(j\omega)|$, and the phase angle differs from that of the input by the amount $\phi = \angle G(j\omega)$.

Example 1

A first-order low-pass filter is shown in Figure 7.2. The transfer function of this filter is

$$G(s) = \frac{V_o(s)}{V_i(s)} = \frac{1}{RCs + 1}$$

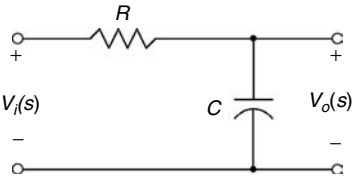


FIGURE 7.2 A first-order low-pass filter.

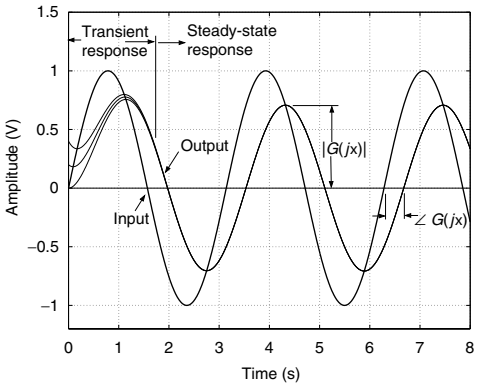


FIGURE 7.3 Frequency response of $G(s) = 1/(0.5s + 1)$ to $u(t) = \sin 2t$.

The sinusoidal transfer function is given by

$$G(j\omega) = \frac{1}{j\omega(RC) + 1} = \frac{1}{j(\omega/\omega_1) + 1}$$

where $\omega_1 = 1/RC$. The magnitude and phase angle of the frequency response are

$$|G(j\omega)| = \frac{1}{\sqrt{1 + (\omega/\omega_1)^2}} \quad \text{and} \quad \phi(\omega) = -\tan^{-1}\frac{\omega}{\omega_1}$$

Figure 7.3 shows the response of the system with $RC = 0.5$ to the input $u = \sin 2t$. It can be seen that the steady-state response is irrelevant to the initial conditions, and the steady-state amplitude of the output is $1/\sqrt{2}$ and the phase angle is $-45°$.

7.2 Bode Plots

There are three commonly used displays of frequency response of a system. They are:

1. Bode diagram or logarithmic plot
2. Polar plot
3. Log-magnitude versus phase plot or Nichols chart

We will present Bode diagrams of sinusoidal transfer functions in this section, followed by the sections on polar plots and log-magnitude versus phase plots.

The main advantages in using the logarithmic plot are the capability of plotting low and high frequency characteristics of the transfer function in one diagram, and the relative ease of adding the separate terms

of a high-order transfer function graphically. The basic types of factors that may occur in a transfer function are as follows:

1. Constant gain K
2. Poles (or zeros) at the origin $(j\omega)^{\pm n}$
3. Poles (or zeros) on the real axis $(j\omega\tau + 1)^{\pm 1}$
4. Complex conjugate poles (or zeros) $[(j\omega/\omega_n)^2 + 2\zeta(j\omega/\omega_n) + 1]^{\pm 1}$

The curves of logarithmic magnitude and phase angle for these four factors can easily be drawn and then added together graphically to obtain the curves for the complete transfer function. The process of drawing the logarithmic plot can be further simplified by using asymptotic approximations to these curves and obtaining the actual curves at specific important frequencies.

7.2.1 Constant Gain *K*

The logarithmic gain for the constant gain K is

$$20\log|K| = \text{constant in decibel}, \quad \angle K = \begin{cases} 0°, & \text{if } K > 0 \\ -180°, & \text{if } K < 0 \end{cases}$$

The gain and phase curves are simply horizontal lines on the Bode diagram.

7.2.2 Poles (or Zeros) at the Origin $(j\omega)^{\pm n}$

Since

$$20\log|j\omega|^{\pm n} = \pm 20n\log\omega, \quad \angle(j\omega)^{\pm n} = \pm n \times 90°$$

the slopes of the magnitude curves are $\pm 20n$ dB/decade for the factor $(j\omega)^{\pm n}$ and the phase angles are constants equal to $\pm n \times 90°$.

7.2.3 Poles (or Zeros) on the Real Axis $(j\omega\tau + 1)^{\pm 1}$

For a pole factor $(j\omega\tau + 1)^{-1}$,

$$\left|\frac{1}{j\omega\tau + 1}\right| = \frac{1}{\sqrt{\omega^2\tau^2 + 1}}$$

The magnitude of the pole factor is 1 when $\omega \ll 1/\tau$, and $1/(\omega\tau)$ when $\omega \gg 1/\tau$. Thus, there are two asymptotic curves for the pole factor,

$$20\log\left|\frac{1}{j\omega\tau + 1}\right| \approx \begin{cases} 0 \text{ dB}, & \text{when } \omega \ll \dfrac{1}{\tau} \\ -20\log\omega\tau = -20\left(\log\omega - \log\dfrac{1}{\tau}\right), & \text{when } \omega \gg \dfrac{1}{\tau} \end{cases}$$

The slope of the asymptotic curve when $\omega \gg 1/\tau$ is -20 dB/decade for the pole factor. The two asymptotes intersect at $\omega = 1/\tau$, the break frequency or the corner frequency. The actual logarithmic gain at $\omega = 1/\tau$ is -3 dB. The phase angle is $\phi(\omega) = -\tan^{-1}\omega\tau$.

The Bode diagram of a zero factor $(j\omega\tau + 1)$ is obtained in the same manner. However, the slope of the magnitude asymptotic curve when $\omega \gg 1/\tau$ is $+20$ dB/decade, and the phase angle is $\phi(\omega) = +\tan^{-1}\omega\tau$. The Bode diagrams of first-order factors are shown in Figure 7.4. Linear approximations to the phase angle curves are also presented.

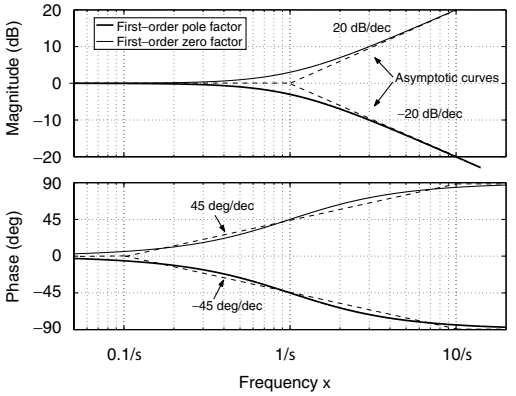


FIGURE 7.4 Bode diagrams for the first-order factors $(j\omega\tau + 1)^{\pm 1}$.

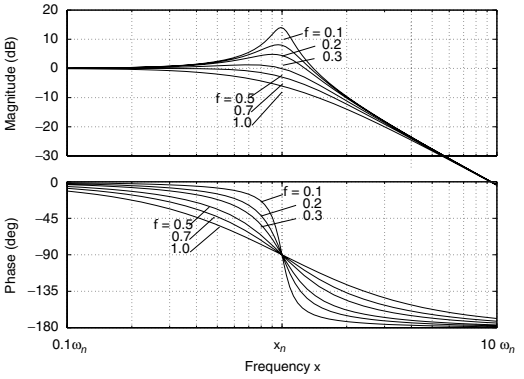


FIGURE 7.5 Bode diagram for the quadratic pole factor $[(j\omega/\omega_n)^2 + 2\zeta(j\omega/\omega_n) + 1)]^{-1}$.

7.2.4 Complex Conjugate Poles (or Zeros) $[(j\omega/\omega_n)^2 + 2\varsigma(j\omega/\omega_n) + 1]^{\pm 1}$

The magnitude and phase angle of the complex conjugate poles $[(j\omega/\omega_n)^2 + 2\zeta(j\omega/\omega_n) + 1]^{-1}$ are

$$\left|\left(j\frac{\omega}{\omega_n}\right)^2 + 2\varsigma\left(j\frac{\omega}{\omega_n}\right) + 1\right|^{-1} = \left[\left(1-\frac{\omega^2}{\omega_n^2}\right)^2 + \left(2\varsigma\frac{\omega}{\omega_n}\right)^2\right]^{-1/2}$$

$$\angle\left[\left(j\frac{\omega}{\omega_n}\right)^2 + 2\varsigma\left(j\frac{\omega}{\omega_n}\right) + 1\right]^{-1} = -\tan^{-1}\frac{2\varsigma\omega/\omega_n}{1-\omega^2/\omega_n^2}$$

The magnitude of the complex conjugate pole factor is 1 when $\omega \ll \omega_n$, and $(\omega/\omega_n)^{-2}$ when $\omega \gg \omega_n$. Therefore, the two asymptotic curves for the complex conjugate pole factor are

$$20\log\left|\left(j\frac{\omega}{\omega_n}\right)^2 + 2\varsigma\left(j\frac{\omega}{\omega_n}\right) + 1\right|^{-1} \approx \begin{cases} 0 \text{ dB}, & \text{when } \omega \ll \omega_n \\ -40(\log\omega - \log\omega_n), & \text{when } \omega \gg \omega_n \end{cases}$$

The slope of the asymptotic curve when $\omega \gg \omega_n$ is –40 dB/decade for the complex conjugate pole factor. The magnitude asymptotes intersect at $\omega = \omega_n$, the natural frequency. The actual gain at $\omega = \omega_n$ is $G(j\omega_n) = 1/2\zeta$. The Bode diagram of a complex conjugate pole factor is shown in Figure 7.5. It is seen from Figure 7.5 that the

difference between the actual magnitude curve and the asymptotic approximation is a function of damping ratio. The resonant frequency ω_r is defined as the frequency where the peak value of the frequency response M_r occurs. When the damping ratio approaches zero, ω_r approaches ω_n. The resonant frequency can be determined by taking the derivative of the magnitude with respect to the frequency, and setting it equal to zero. The resonant frequency and the peak value of the magnitude are represented by

$$\omega_r = \omega_n\sqrt{1-2\zeta^2}, \quad \zeta < 0.707 \tag{7.9a}$$

and

$$M_r = \frac{1}{2\zeta\sqrt{1-\zeta^2}}, \quad \zeta < 0.707 \tag{7.9b}$$

Example 2

Let us consider the transfer function

$$G(s) = \frac{10(s/5+1)}{s(s+1)[(s/10)^2+(s/10)+1]}$$

We first list the basic factors of $G(s)$ in Table 7.1 in the order of increasing corner or natural frequencies.

The complete asymptotic magnitude curve for $G(j\omega)$ is produced by adding together the asymptotic logarithmic magnitudes of each factor, as shown by the solid line in Figure 7.6. Since the dc gain of each factor is 1, these factors have no effect on the asymptotic magnitude until the frequency approaches their corner or natural frequencies. Thus, the asymptotic magnitude can be quickly obtained by plotting each asymptote in order as frequency increases. The asymptotic curve intersects 20 dB at $\omega = 1$ with the slope −20 dB/decade due to the pole at the origin and the constant gain $K = 10$. At $\omega = 1$ the slope further decreases to −40 dB/decade due to the pole at $\omega = 1$. Then at $\omega = 5$ the slope increases to −20 dB/decade

TABLE 7.1 The Basic Factors of $G(j\omega)$

Type of Factors	Constant Gain	Pole	Pole	Zero	Complex Poles
Corner frequency	$K = 10$	0	1	5	10
Order	0	−1	−1	+1	−2

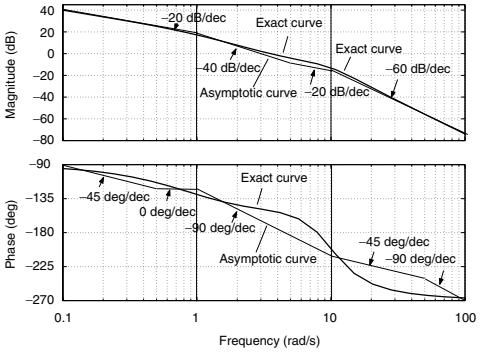


FIGURE 7.6 The Bode plot of the transfer function in Example 2.

due to the zero at $\omega = 5$. Finally at $\omega = 10$ the slope becomes −60 dB/decade due to the complex conjugate poles at $\omega_n = 10$.

The exact magnitude is obtained by calculating the actual magnitude at important frequencies such as the corner or natural frequencies of each factor. The phase curve can be obtained by adding the phase due to each factor. Although the linear approximation of the phase characteristic for a single pole or zero is suitable for initial analysis, the error between the exact phase curve and the linear approximation of complex conjugate poles can be large, as seen in Figure 7.6. Hence, if the accurate phase angle curve is required, a computer program such as Matlab or Ctrl-C can be utilized to generate the actual phase curve.

7.3 Polar Plots

The polar plot of a sinusoidal transfer function $G(j\omega)$ is a plot of both the magnitude and the phase of the frequency response in polar coordinates as the frequency ω varies from zero to infinity. Since the sinusoidal transfer function $G(j\omega)$ can be expressed as

$$G(j\omega) = \mathrm{Re}[G(j\omega)] + j\,\mathrm{Im}[G(j\omega)] = |G(j\omega)|e^{j\phi}$$

the polar plot of $G(j\omega)$ is a plot of $\mathrm{Re}[G(j\omega)]$ on the horizontal axis versus $\mathrm{Im}[G(j\omega)]$ on the vertical axis in the complex $G(s)$-plane as ω varies from zero to infinity. Hence, for each value of ω, a polar plot of $G(j\omega)$ is defined by a vector of length $|G(j\omega)|$ and a phase angle $\phi = \angle G(j\omega)$, as in Equation 7.8.

We can investigate the general shapes of polar plots according to the system types and relative degrees of transfer functions. Relative degree of a transfer function is defined as the difference between the degree of the denominator polynomial and that of the numerator. Consider a transfer function of the form

$$\begin{aligned} G(j\omega) &= \frac{K(1+j\omega\tau_a)(1+j\omega\tau_b)\cdots}{(j\omega)^N(1+j\omega\tau_1)(1+j\omega\tau_2)\cdots} \\ &= \frac{b_0(j\omega)^m + b_1(j\omega)^{m-1} + \cdots}{a_0(j\omega)^n + b_1(j\omega)^{n-1} + \cdots} \end{aligned}$$

where $K > 0$ and the relative degree $n - m \geq 0$. The magnitudes and phase angles of $G(j\omega)$ as ω approaches zero and infinity are presented in Table 7.2. The general shapes of the polar plots of various system types in the low-frequency portion are shown in Figure 7.7. The high-frequency portions of the polar plots of various relative degrees are shown in Figure 7.8. It can be seen that the $G(j\omega)$ loci are parallel to either the horizontal or the vertical axes with infinite magnitude as $\omega \to 0^+$ for system types greater than zero. If the relative degree is greater than zero, the $G(j\omega)$ loci converge to the origin clockwise and are tangent to one or the other axes. Note that the polar plot curves can be very complicated due to the numerator and denominator dynamics over the intermediate frequency range. Therefore, the polar plot of $G(j\omega)$ in the frequency range of interest must be accurately determined.

TABLE 7.2 $G(j\omega)$ versus System Type and Relative Degree as $\omega \to 0^+$ and ∞

System Type		Relative Degree	
N	$\omega \to 0^+$	$n - m$	$\omega \to \infty$
0	$K \angle 0°$	0	$b_0/a_0 \angle 0°$
1	$\infty \angle -90°$	1	$0 \angle -90°$
2	$\infty \angle -180°$	2	$0 \angle -180°$
3	$\infty \angle -270°$	3	$0 \angle -270°$

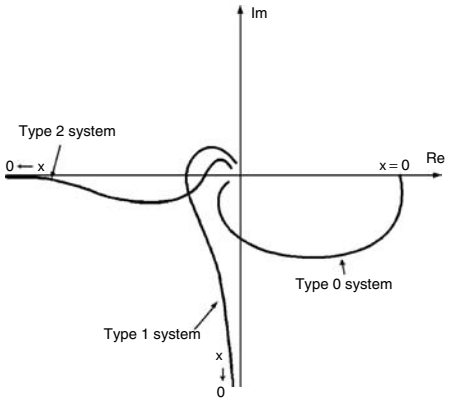


FIGURE 7.7 Polar plots of system with various system types as $\omega \to 0$.

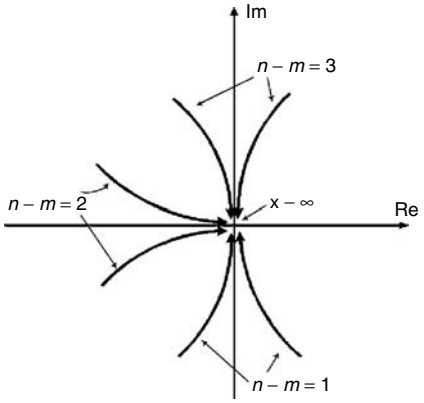


FIGURE 7.8 Polar plots of system with various relative degrees as $\omega \to \infty$.

We will see that for a closed-loop system, the polar plot of the loop transfer function is useful in determining the stability of the system. The polar plots of some simple systems are shown in Figure 7.9.

7.4 Log-Magnitude versus Phase Plots

Another approach to presenting the frequency response of a system by a single graph is to plot its logarithmic magnitude versus the phase angle over a frequency range of interest. The resulting curve is a function of the frequency ω. Such log-magnitude versus phase plots are called Nichols charts.

Advantages of the Nichols chart are that the relative stability of the closed-loop system can be determined quickly and that the process of closed-loop compensation can be carried out easily. The Nichols charts of the systems in Figure 7.9 are depicted in Figure 7.10 for comparison. Figure 7.11 displays three different frequency-response curves of the second-order system

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

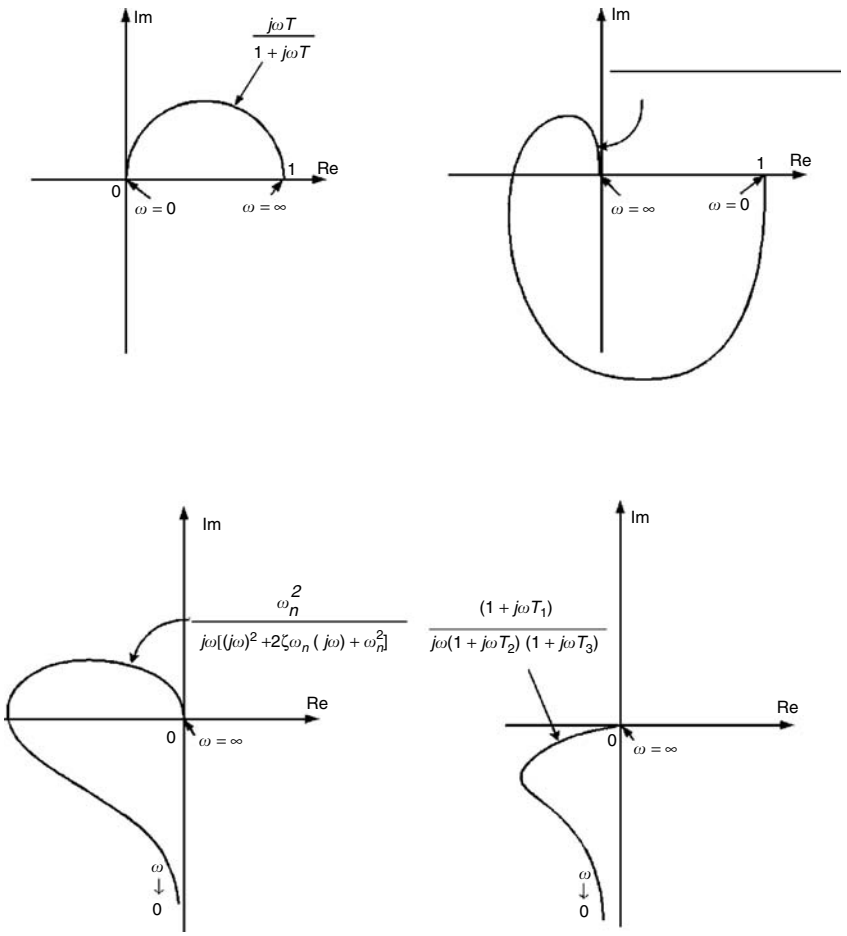


FIGURE 7.9 Polar plots of simple transfer functions.

7.5 Experimental Determination of Transfer Functions

We can obtain a transfer function model from frequency-response measurements of a stable system. First, the Bode diagram of the frequency response is plotted from the measurements. Then the open-loop transfer function can be deduced from the magnitude and phase plots based on the relationships of the basic pole and zero factors.

A wave analyzer is a device to measure the amplitudes and phases of the steady-state response as the frequency of the input sinusoidal wave is altered. A transfer function analyzer can be used to measure the open-loop and closed-loop transfer functions.

We will use a computer program combined with an analog-to-digital and digital-to-analog (AD and DA) card to generate the sinusoidal input signal and to measure the frequency response of a system. Consider the second-order Sallen-Key low-pass filter in Figure 7.12. The transfer function of the filter is given by

$$G(s) = \frac{V_o(s)}{V_i(s)} = \frac{K}{s^2/\omega_n^2 + 2\varsigma(s/\omega_n) + 1} \tag{7.10}$$

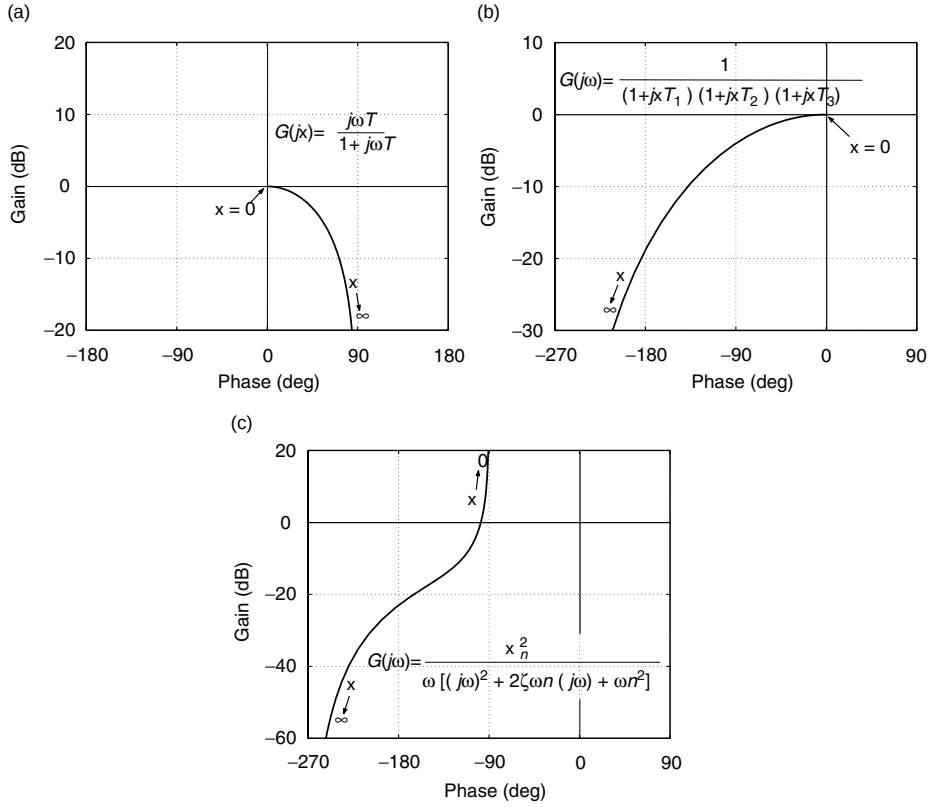


FIGURE 7.10 Nichols charts of simple transfer functions.

where

$$K = \frac{R_1 + R_2}{R_2}, \quad \omega_n = \frac{1}{\sqrt{R_A R_B C_A C_B}}$$

and

$$\varsigma = \frac{1}{2}\left[\sqrt{\frac{C_B}{C_A}}\frac{R_A + R_B}{\sqrt{R_A R_B}} + (1 - K)\sqrt{\frac{R_A C_A}{R_B C_B}}\right]$$

The Real-Time Windows Target in MATLAB is used with an Advantech PCL-818L AD and DA card. The sampling time is 0.001 s. The measured magnitudes and phase angles are shown in Figure 7.13. From the Bode plot, we can find that the dc gain is equal to 1.995 and the natural frequency $\omega_n = 17.90$ rad/s. From Equation 7.9b and $M_r = 1.993$, we have $\zeta = 0.26$.

An alternative to estimating the transfer function is to use an excitation signal that is sufficiently rich in the frequency contents of interest and to measure the corresponding output. System identification technique is then applied to find the order and parameters of the transfer function. Suitable excitation signals are the impulse signal, sweep sine signal, random sequence, and so forth. Figure 7.14 presents the sweep sine input and the corresponding output. The MATLAB System Identification Toolbox is then

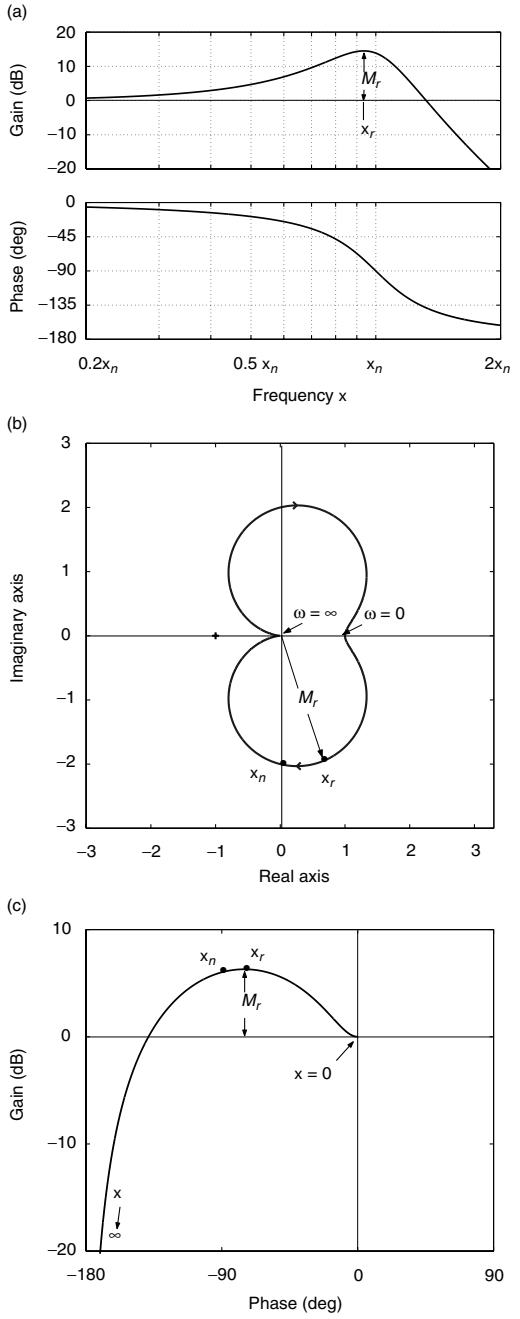


FIGURE 7.11 Three frequency response representations of $G(s) = \omega_n^2/(s^2 + 2\varsigma\omega_n s + \omega_n^2)$: (a) Bode diagram, (b) polar plot, and (c) Nichols chart.

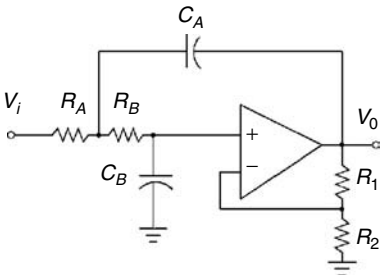


FIGURE 7.12 Sallen-Key low-pass filter.

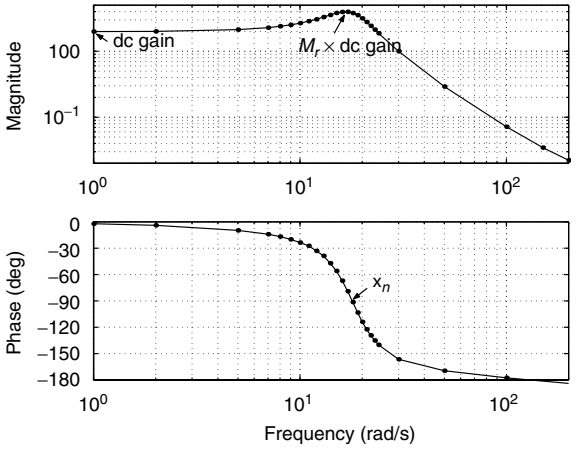


FIGURE 7.13 Frequency response of the Sallen-Key filter from experimental data.

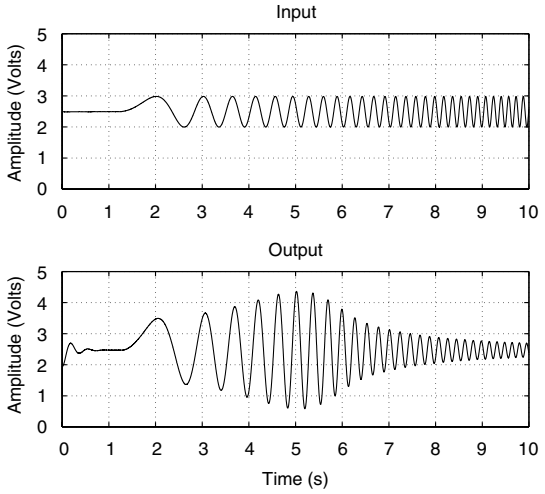


FIGURE 7.14 Sweep sine response of the Sallen-Key filter.

TABLE 7.3 Estimated Transfer Functions for the Second-Order Low-Pass Filter

1. Ideal op-amp circuit: $G(s) = \dfrac{1.997}{(s/18.09)^2 + 2 \times 0.271(s/18.09) + 1}$
   where the measured values of resistors and capacitors are substituted in Equation 7.10 with $R_1 = 98.4$ kΩ, $R_2 = 98.7$ kΩ, $R_A = 51.3$ kΩ, $R_B = 98.5$ kΩ, $C_A = 1.083$ μF, and $C_B = 0.564$ μF.
2. From the Bode plot: $G(s) = \dfrac{1.995}{(s/17.90)^2 + 2 \times 0.259(s/17.90) + 1}$
3. System identification: $G(s) = \dfrac{1.997}{(s/17.78)^2 + 2 \times 0.255(s/17.78) + 1}$

utilized to estimate the transfer function. The resulting transfer functions from the ideal op-amp circuit in Equation 7.10, the Bode plot, and system identification are shown in Table 7.3 for comparison. It is seen that the differences among the three transfer functions are very small. However, the task of determining transfer functions from Bode plots can be very difficult as various pole or zero factors of close corner frequencies can complicate the magnitude and phase plots for high-order systems. Thus, it is recommended that system identification technique be used for determination of high-order transfer functions.

7.6 The Nyquist Stability Criterion

The Nyquist stability criterion provides a graphical procedure for determining the closed-loop stability from the open-loop frequency-response curves. The criterion is based on a result from complex variables theory known as the argument principle, due to Cauchy.

Suppose $F(s)$ is a rational function of s with real coefficients that are analytic everywhere in the s-plane except at its poles. Let Γ_s be a closed, clockwise contour in the s-plane that does not pass through any zeros or poles of $F(s)$. The contour map Γ_F is defined by substituting the values of s on the contour Γ_s for s in $F(s)$. The resulting map is also a closed continuous contour in the $F(s)$-plane. The principle of the argument can be stated as follows:

> A contour map Γ_F of a complex function $F(s)$ defined on Γ_s in the s-plane will only encircle the origin of the $F(s)$-plane if the contour contains a pole or zero of the function. The net number that Γ_F encircles the origin in the clockwise direction is
>
> $$N = Z - P \tag{7.11}$$
>
> where Z and P are, respectively, the numbers of zeros and poles of $F(s)$ enclosed by a closed clockwise contour Γ_s in the s-plane.

Example 3

To illustrate the argument principle, consider a rational function

$$F(s) = \frac{(s+3)(s+4)}{(s+1)(s+2)}$$

which has zeros at $s = -3, -4$ and poles at $s = -1, -2$. The various contour maps of $F(s)$ are shown in Figure 7.15, where Γ_r denotes the contour map of a clockwise circular contour of radius r in the s-plane. We have the following observations from Figure 7.15:

1. The contour map $\Gamma_{0.5}$ does not encircle the origin of the $F(s)$-plane as the contour in the s-plane does not encircle any pole or zero.
2. $\Gamma_{1.99}$ encircles the origin once in the counterclockwise direction as the contour encircles the pole at $s = -1$ in the clockwise direction in the s-plane, and from Equation 7.11, $N = Z - P = 0 - 1 = -1$. Note that $\Gamma_{1.99}$ is a closed contour with two loops and only the one encircling the origin is shown in Figure 7.15.

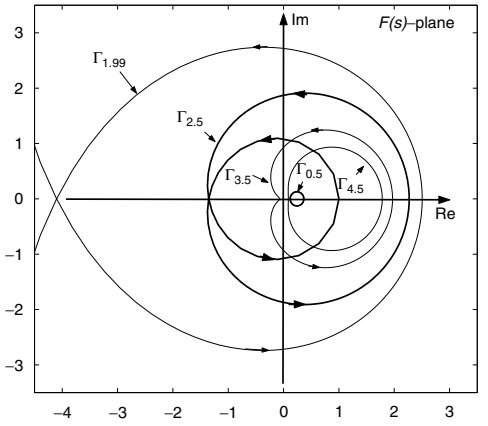


FIGURE 7.15 The contour maps of $F(s)$ in Example 3.

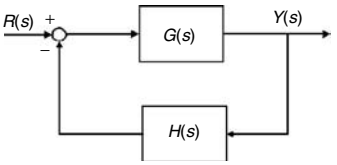


FIGURE 7.16 Closed-loop system.

3. $\Gamma_{2.5}$ encircles the origin twice in the counterclockwise direction as the contour contains two poles at $s = -1, -2$ and $N = Z - P = 0 - 2 = -2$.
4. When the radius of the contour is increased to contain the poles at $s = -1, -2$ and the zero at $s = -3$, then $N = Z - P = 1 - 2 = -1$ and a contour map like $\Gamma_{3.5}$ encircles the origin once in the counterclockwise direction.
5. When the radius of the contour is further increased to encircle the two poles and two zeros, then $N = 2 - 2 = 0$ and the contour map like $\Gamma_{4.5}$ does not encircle the origin.

We now apply Cauchy's principle of argument to develop the Nyquist stability criterion. Suppose that the characteristic equation of the closed-loop system in Figure 7.16 is

$$F(s) = 1 + G(s)H(s) = 0$$

Let $L(s) = G(s)H(s)$, the loop transfer function. Using the argument principle, let us assume that none of the poles or zeros of $F(s)$ lie on the imaginary axis in the s-plane. We now define the Nyquist path, Γ_s, that is composed of the imaginary axis and a semicircle of infinite radius. This contour completely encloses the entire complex right-half plane as depicted in Figure 7.17a. The corresponding contour map Γ_F is shown in Figure 7.17b. It follows from the argument principle that N corresponds to the net number of clockwise encirclements of the origin of the $1 + L(s)$-plane by Γ_F. P is the number of poles of $F(s)$ in the right-half s-plane and thus is the number of poles of the loop transfer function $L(s)$ in the right-half s-plane. Z is the number of zeros of the characteristic equation $F(s)$ of the closed-loop system in the right-half s-plane. Therefore, Z must be zero for the closed-loop system to be stable.

In practice, a modification is made to simplify the application of the Nyquist criterion. Instead of plotting Γ_F in the $1 + L(s)$-plane, we plot just $L(s)$ evaluated along the contour Γ_s. The resulting contour map Γ_L is in the $L(s)$-plane and has the same shape as Γ_F but is shifted 1 unit to the left, as shown in Figure 7.17c. It thus follows that N is the net number of encirclements of the -1 point in the $L(s)$-plane.

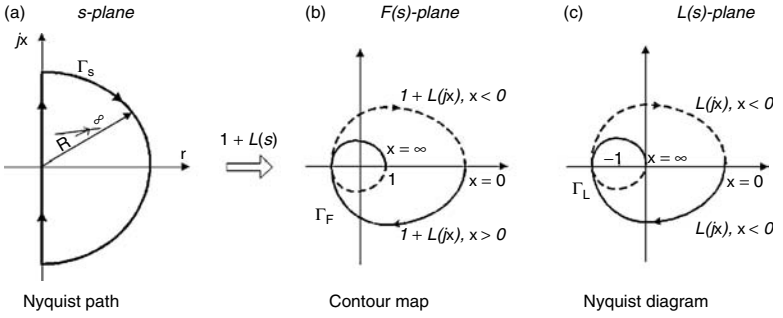


FIGURE 7.17 Nyquist diagram.

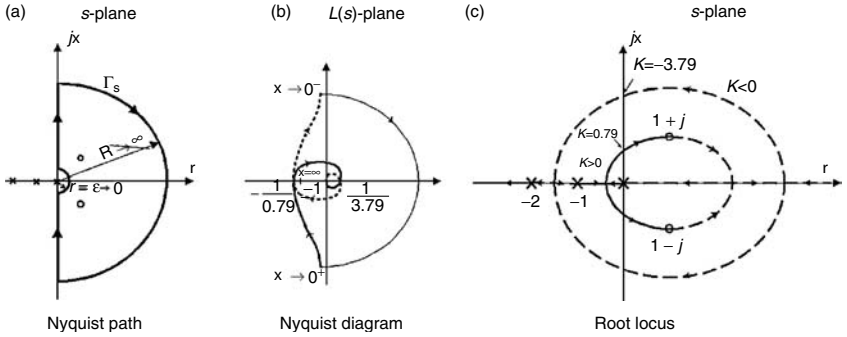


FIGURE 7.18 Nyquist diagram and root locus of Example 4.

The Nyquist stability criterion can now be stated as follows:

A necessary and sufficient condition for the closed-loop stability of a system defined by the loop transfer function $L(s)$ is that

$$Z = N + P \tag{7.12}$$

be equal to zero, where N is the net number of encirclements of the -1 point in the $L(s)$-plane, and P is the number of unstable poles of the loop transfer function $L(s)$.

Example 4

Consider the system with the loop transfer function

$$KL(s) = KG(s)H(s) = K\frac{s^2 - 2s + 2}{s(s+1)(s+2)} \tag{7.13}$$

Let us determine the range of the gain K such that the closed-loop system is stable. Since there is a pole at $s = 0$, we need to modify the Nyquist path to detour around the origin. The contour is shown in Figure 7.18a, where the detour is chosen to be a semicircle of radius approaching zero in the limit. We use the following procedure to sketch the Nyquist plot in Figure 7.18b:

1. Determine $L(j\omega)$ as $\omega \to 0^+$: $L(s)$ is of system type 1 and thus

$$L(j\omega) \approx \frac{1}{j\omega} = \infty\angle -90^\circ$$

according to Table 7.2.

2. Determine $L(j\omega)$ as $\omega \to \infty$: $L(s)$ has a relative degree of 1 and

$$L(j\omega) \approx \frac{1}{j\omega} = 0\angle -90^\circ$$

according to Table 7.2.

3. From the Bode plot, draw the polar plot of $L(j\omega)$ as ω varies from 0^+ to ∞. Although the magnitude curve of the factor $(s^2 - 2s + 2)$ is the same as the factor $(s^2 + 2s + 2)$, the phase of the factor $(s^2 - 2s + 2)$ changes from 0° to −180°. Thus, a sketch of the Bode diagram shows that the magnitude curve varies from infinity to zero and the phase changes from −90° to −450°. Since there are two points at which the phases are −180° and −360°, there will be two intersections of the $L(j\omega)$ locus with the real axis in the $L(s)$-plane.
4. Draw the polar plot of $L(j\omega)$ as ω varies from 0^- to $-\infty$ by reflecting the curve of $L(j\omega)$ in procedure 3 with respect to the real axis in the $L(s)$-plane.
5. Determine the contour map of the small detour around the origin of the s-plane to complete the plot. On the detour,

$$s = \lim_{\varepsilon \to 0} \varepsilon e^{j\theta}, \quad -90^\circ \le \theta \le 90^\circ$$

The contour map of the detour can then be determined by

$$\lim_{\varepsilon \to 0} L(\varepsilon e^{j\theta}) = \lim_{\varepsilon \to 0} \frac{(\varepsilon e^{j\theta})^2 - 2\varepsilon e^{j\theta} + 2}{\varepsilon e^{j\theta}(\varepsilon e^{j\theta} + 1)(\varepsilon e^{j\theta} + 2)} = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon e^{j\theta}} = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \angle -\theta$$

The resulting map is a large semicircle of radius approaching infinity. This semicircle starts at the point $L(j0^-)$ and swings 180° in the counterclockwise direction to connect the point $L(j0^+)$ in the $L(s)$-plane.

6. Calculate the intersections of the $L(j\omega)$ locus with the real-axis, for these points are related to the relative stability of the system. Suppose that the $L(j\omega)$ locus intersects the real axis for some critical frequency ω_{cr}. Then

$$L(j\omega_{cr}) = \begin{cases} 180^\circ + k360^\circ, & \text{for } K > 0 \\ 0^\circ + k360^\circ, & \text{for } K < 0 \end{cases}$$

where $k = 0, \pm 1, \pm 2, \pm 3, \ldots$. This phase condition at the critical frequency is directly related to the angle condition of the root locus when the root locus crosses the imaginary axis. Therefore, we can utilize the Routh–Hurwitz criterion to determine the points where the $L(j\omega)$ locus crosses the real axis. The characteristic equation of the system (7.13) can be written as

$$s^3 + (K+3)s^2 + (2-2K)s + 2K = 0$$

Thus, the Routh array is

$$\begin{array}{c|ll} s^3 & 1 & 2-2K \\ s^2 & K+3 & 2K \\ s^1 & c & \\ s^0 & 2K & \end{array}$$

where

$$c = \frac{(K+3)(2-2K) - 2K}{K+3}$$

Let $c = 0$, and solving for K, we get the critical gains

$$K_{cr} = \frac{-3 \pm \sqrt{21}}{2} = 0.79, -3.79$$

Substituting the values of K_{cr} in the auxiliary equation

$$(K_{cr} + 3)s^2 + 2K_{cr} = 0$$

we obtain the critical frequencies

$$\omega_{cr} = \sqrt{\frac{2K_{cr}}{K_{cr} + 3}} = \begin{cases} 0.65, & \text{when } K_{cr} = 0.79 \\ 3.10, & \text{when } K_{cr} = -3.79 \end{cases}$$

At the critical frequency, we have the characteristic equation

$$1 + K_{cr}L(j\omega_{cr}) = 0$$

Hence the points of the $L(j\omega)$ locus that cross the real-axis are

$$L(j\omega_{cr}) = -\frac{1}{K_{cr}} = -\frac{1}{0.79}, \frac{1}{3.79}$$

The complete Nyquist plot is shown not to scale in Figure 7.18b. The range of the gain K for which the system is stable can be determined using Nyquist criterion. For different values of K, the Nyquist diagram needs to be redrawn in order to count the number of encirclement of the -1 point. We can avoid this by counting the number of encirclement of $-1/K$ point instead. From the Nyquist criterion, $Z = N + P$, where $P = 0$. It can be seen from Figure 7.18b that there are four cases of the encirclements of the $-1/K$ point.

1. $K > 0$ and $-1/K < -1/0.79 \Rightarrow 0 < K < 0.79$, and $N = 0$. We have $Z = 0$ and the system is stable.
2. $K > 0$ and $-1/K > -1/0.79 \Rightarrow K > 0.79$, and $N = 2$. We have $Z = 2$ and the system has two unstable poles.
3. $K < 0$ and $-1/K < 1/3.79 \Rightarrow K < -3.79$, and $N = 3$. We have $Z = 3$ and the system has three unstable poles.
4. $K < 0$ and $-1/K > 1/3.79 \Rightarrow -3.79 < K < 0$ and $N = 1$. We have $Z = 1$ and the system has one unstable pole.

The root locus of system (7.13) is also shown in Figure 7.18c for comparison.

7.7 Relative Stability

In designing a control system, it is required that the system be stable. In addition to stability, there are important concerns such as acceptable transient response and capability to deal with model uncertainty. Since the model used in the design and analysis of a control system is never exact, it may suggest a stable system; but the physical system turns out to be unstable. Therefore, it is required that the system not only be stable but also have some stability margin or adequate relative stability.

Suppose that the sinusoidal loop transfer function $L(j\omega)$ locus intersects the -1 point for some critical frequency ω_{cr}. Then

$$L(j\omega_{cr}) = G(j\omega_{cr})H(j\omega_{cr}) = -1$$

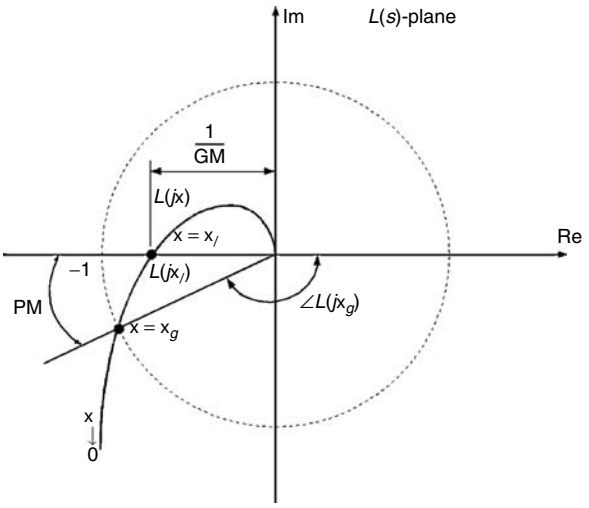


FIGURE 7.19 Gain and phase margins.

or

$$1 + L(j\omega_{\text{cr}}) = 1 + G(j\omega_{\text{cr}})H(j\omega_{\text{cr}}) = 0$$

This indicates that the closed-loop system has a pair of complex poles at $s = \pm j\omega_{cr}$. Hence, the system is marginally stable and oscillates with the frequency ω_{cr}, provided that all other closed-loop system poles are in the left half s-plane. In general, the closer the $L(j\omega)$ locus comes to the $-1 + j0$ point in the Nyquist plot, the more oscillatory is the system response. For this reason, the closeness of the $L(j\omega)$ locus to the -1 point can be used as a measure of the stability margin. Two traditional measures of the stability margin are gain margin and phase margin.

Gain margin and phase margin are usually defined for stable closed-loop systems that are characterized by a minimum phase, loop transfer function $L(s)$. The gain margin is the factor by which the open-loop gain of a stable closed-loop system can be increased before the system goes unstable. The phase margin is the amount of additional phase lag at the gain crossover frequency required to make the stable closed-loop system marginally stable. Thus we have the following definitions:

Gain margin (GM): The gain margin is the reciprocal of the magnitude $|L(j\omega)|$ at the phase crossover frequency ω_ϕ, where the phase of $L(j\omega_\phi)$ reaches $-180°$. The gain margin is given by

$$\text{GM} = \frac{1}{|L(j\omega_\phi)|} \qquad \text{or} \qquad \text{GM(dB)} = -20 \log|L(j\omega_\phi)|$$

Phase margin (PM): The phase margin is defined as the angle between the phase of the loop transfer at the gain crossover frequency ω_g where $|L(j\omega_g)| = 1$ and the angle $-180°$, or $\text{PM} = \angle L(j\omega_g) + 180°$.

The gain and phase margins are shown in Figure 7.19. Gain and phase margins are stability margins for single-input single-output systems. They cannot apply for multi-input multi-output systems. In addition, they can be a poor indication of stability margin in the face of combined gain and phase variations, as shown in Figure 7.20. This is due to the fact that gain and phase margins are measures of stability margin in terms of only pure gain and phase variations, but not a combination of both. As a consequence, a system may have good gain and phase margins, but it is close to instability, as indicated in Figure 7.20. To make up for the insufficiencies of gain and phase margins, a third stability margin, return difference, is used in modern control theory. We will only consider single-input single-output systems.

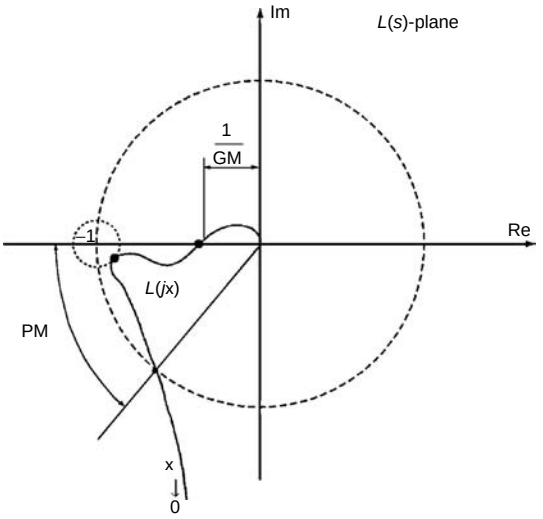


FIGURE 7.20 Insufficiency of gain and phase margins.

Minimum return difference: The minimum return difference is the minimum value of $|1 + L(j\omega)|$, for $0 < \omega < \infty$. It can be seen from Figure 7.20 that the minimum return difference is the minimum distance from the Nyquist plot to the −1 point. Therefore, the gain and phase margins are special cases of the minimum return difference. The gain margin is directly related to the case when the minimum return difference occurs at the phase crossover frequency, and the phase margin is corresponding to the case that the minimum return difference occurs at the gain crossover frequency.

Although the minimum return difference is a better measure of stability margin than the gain and phase margins, it is seldom used in the classical control theory. This is because the classical control analysis and design is usually carried out using the Bode diagram or the Nichols chart instead of the Nyquist plot. The gain and phase margins are more easily determined from the Bode diagram or the Nichols chart than the Nyquist plot. Despite the fact that the minimum return difference can be easily evaluated from the Nyquist plot, it is difficult to determine the minimum return difference from the Bode plot or the Nichols chart.

We now correlate the phase margin and the damping ratio ζ of an underdamped second-order system. Consider the standard second-order system

$$T(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{7.14}$$

We assume that the transfer function $T(s)$ comes from a unity feedback configuration and can be rewritten as

$$T(s) = \frac{G(s)}{1 + G(s)}$$

where the open-loop transfer function $G(s)$ is given by

$$G(s) = \frac{\omega_n^2}{s(s + 2\zeta\omega_n)} \tag{7.15}$$

The phase margin occurs at the gain crossover frequency ω_c when $|G(j\omega_c)| = 1$, or

$$\frac{\omega_n^2}{\omega_c(\omega_c^2 + 4\zeta^2\omega_n^2)^{1/2}} = 1$$

This equation can be rewritten as

$$(\omega_c^2)^2 + 4\zeta^2\omega_n^2(\omega_c^2) - \omega_n^4 = 0$$

Solving for positive ω_c, we obtain

$$\frac{\omega_c^2}{\omega_n^2} = (4\zeta^4 + 1)^{1/2} - 2\zeta^2$$

Substituting $s = j\omega_c$ into Equation 7.15, the phase margin for the system is

$$\begin{aligned} \text{PM} &= 180^\circ + \angle G(j\omega_c) \\ &= 180^\circ - 90^\circ - \tan^{-1}\left(\frac{\omega_c}{2\zeta\omega_n}\right) \\ &= 90^\circ - \tan^{-1}\left(\frac{1}{2\zeta}[(4\zeta^4 + 1)^{1/2} - 2\zeta^2]^{1/2}\right) \\ &= \tan^{-1}\left(2\zeta\left[\frac{1}{(4\zeta^4 + 1)^{1/2} - 2\zeta^2}\right]^{1/2}\right) \end{aligned} \tag{7.16}$$

Equation 7.16 relates the damping ratio of the standard second-order system (7.14) to the phase margin of its corresponding open-loop system (7.15) in a unity feedback configuration. This equation provides a correlation between the frequency response and the time response. A plot of ζ versus PM is shown in Figure 7.21. The curve of ζ versus PM can be approximated by a dashed line in Figure 7.21. The linear approximation can be expressed as

$$\zeta = 0.01 \text{ PM} \tag{7.17}$$

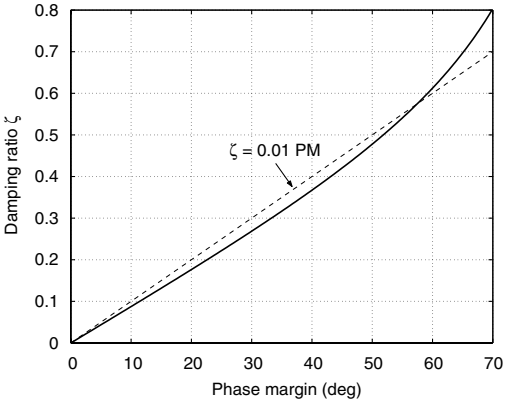


FIGURE 7.21 Damping ratio versus phase margin for a second-order system.

This approximation is reasonably accurate for $\zeta \leq 0.7$ and is useful in relating the frequency response to the transient performance of a second-order system. Equation 7.17 may also be used for higher-order systems if the system can be assumed to have a pair of dominant underdamped complex poles.

References

1. Dorf, R.C., and Bishop, R.H., *Modern Control Systems,* 9th ed., Prentice-Hall, 2001.
2. Ogata, K., *Modern Control Engineering,* 2nd ed., Prentice-Hall, 1990.
3. Kuo, B.C., *Control Systems,* 7th ed., Prentice-Hall, 1995.
4. Franklin, G.F., Powell, J.D., and Emami-Naeini, A., *Feedback Control of Dynamic Systems,* 3rd ed., Addison-Wesley, 1994.
5. Phillips, C.L., and Harbor, R.D., *Feedback Control Systems,* 4th ed., Prentice-Hall, 2000.
6. Wolovich, W.A., *Automatic Control Systems: Basic Analysis and Design,* Harcourt Brace College Publishing, 1994.

8

Kalman Filters as Dynamic System State Observers

Timothy P. Crain II
NASA Johnson Space Center

8.1 The Discrete-Time Linear Kalman Filter

Distilled to its most fundamental elements, the Kalman filter [1] is a predictor–corrector estimation algorithm that uses a dynamic system model to predict state values and a measurement model to correct this prediction. However, the Kalman filter is capable of a great deal more than just state observation in such a manner. By making certain stochastic assumptions, the Kalman filter carries along an internal metric of the statistical confidence of the estimate of individual state elements in the form of a covariance matrix. The essential properties of the Kalman filter are derived from the requirements that the state estimate be

- Linear combination of the previous state estimate and current measurement information
- Unbiased with respect to the true state
- Optimal in terms of having minimum variance with respect to the true state

Starting with these basic requirements an elegant and efficient formulation for the implementation of the Kalman filter may be derived.

The Kalman filter processes a time series of measurements to update the estimate of the system state and utilizes a dynamic model to propagate the state estimate between measurements. The observed measurement is assumed to be a function of the system state and can be represented via

$$\mathbf{Y}(t) = h(\mathbf{X}(t), \beta, t) + \mathbf{v}(t) \tag{8.1}$$

where $\mathbf{Y}(t)$ is an m dimensional observable, h is the nonlinear measurement model, $\mathbf{X}(t)$ is the n-dimensional system state, β is a vector of modeling parameters, and $\mathbf{v}(t)$ is a random process accounting for measurement noise.

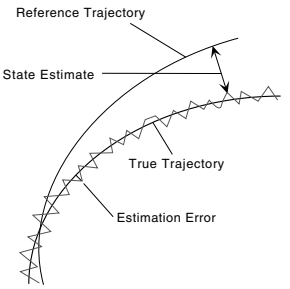


FIGURE 8.1 LKF tracking of a two-dimensional trajectory.

The true dynamic system is described by a general first-order, ordinary differential equation

$$\dot{\mathbf{X}}(t) = f(\mathbf{X}(t), \alpha, t) + \mathbf{w}(t) \tag{8.2}$$

where f is the nonlinear dynamics function that incorporates all significant deterministic effects of the environment, α is a vector of parameters used in the model, and $\mathbf{w}(t)$ is a random process that accounts for the noise present from mismodeling in f or from the quantum uncertainty of the universe, depending on the accuracy of the deterministic model in use.

With these general models available, a linear Kalman filter (LKF) may be derived in a discrete-time formulation. The dynamics and measurement functions are linearized about a known reference state, $\tilde{\mathbf{X}}(t)$, which is related to the true environment state, $\mathbf{X}(t)$, via

$$\tilde{\mathbf{X}}(t) + \mathbf{x}(t) = \mathbf{X}(t) \tag{8.3}$$

The LKF state estimate is related to the true difference by

$$\hat{\mathbf{x}}_k^{(\pm)} = \mathbf{x}_k + \delta\mathbf{x}_k^{(\pm)} \tag{8.4}$$

where the "$\hat{}$" denotes the state estimate (or filter state), $\delta\mathbf{x}_k^{(\pm)}$ is the estimation error, and "±" indicates whether the estimate and error are evaluated instantaneously before (−) or after (+) measurement update at discrete time t_k.

It is important to emphasize that the LKF filter state is the estimate of the difference between the environment and the reference state. The LKF mode of operation will therefore carry along a reference state and the filter state between measurement updates. Only the filter state is at the time of measurement update. Figure 8.1 illustrates the generalized relationship between the true, reference, and filter states in an LKF estimating a two-dimensional trajectory.

8.1.1 Linearization of Dynamic and Measurement System Models

The dynamics and measurement functions may be linearized about the known reference state, $\tilde{\mathbf{X}}(t)$, according to

$$f(\mathbf{X}, \alpha, t) \;.\; f(\tilde{\mathbf{X}}(t), \alpha, t) + \mathbf{F}(\tilde{\mathbf{X}}(t), \alpha, t)\mathbf{x}(t) + \mathbf{w}(t) \tag{8.5}$$

$$h(\mathbf{X}, \alpha, t) \;.\; h(\tilde{\mathbf{X}}(t), \beta, t) + \mathbf{H}(\tilde{\mathbf{X}}(t), \beta, t)\mathbf{x}(t) + \mathbf{v}(t) \tag{8.6}$$

where **F** is the dynamics partial derivative matrix and **H** is the measurement partial derivative matrix defined by

$$\mathbf{F}(\tilde{\mathbf{X}}(t), \alpha, t) = \left.\frac{\partial f}{\partial \mathbf{X}}\right|_{\mathbf{X}=\tilde{\mathbf{X}}} \tag{8.7}$$

$$\mathbf{H}(\tilde{\mathbf{X}}(t), \beta, t) = \left.\frac{\partial h}{\partial \mathbf{X}}\right|_{\mathbf{X}=\tilde{\mathbf{X}}} \tag{8.8}$$

and $\mathbf{x}(t)$ is the true state to be estimated representing the difference between the environment and reference states

$$\mathbf{x}(t) = \mathbf{X}(t) - \tilde{\mathbf{X}}(t) \tag{8.9}$$

After linearizing the dynamic and measurement models, the effect of neglecting the higher order terms is assumed to be included in the random processes $\mathbf{w}(t)$ and $\mathbf{v}(t)$. The linearization is an acceptable approximation if $\mathbf{x}(t)$ is sufficiently small.

The reference and filter states are propagated according to the discrete-time linear relationship

$$\tilde{\mathbf{X}}_{k+1} = \Phi(t_{k+1}, t_k)\tilde{\mathbf{X}}_k \tag{8.10}$$

$$\hat{\mathbf{x}}_{k+1}^{(-)} = \Phi(t_{k+1}, t_k)\hat{\mathbf{x}}_k^{(-)} \tag{8.11}$$

where $\Phi(t_{k+1}, t_k)$ is the state transition matrix from time t_k to time t_{k+1} and has the following properties:

$$\begin{aligned} \Phi(t_k, t_k) &= \mathbf{I} \\ \dot{\Phi}(t_{k+1}, t_k) &= \mathbf{F}(\tilde{\mathbf{X}}(t), \alpha, t)\Phi(t_{k+1}, t_k) \\ \Phi(t_{k+2}, t_k) &= \Phi(t_{k+2}, t_{k+1})\Phi(t_{k+1}, t_k) \end{aligned} \tag{8.12}$$

Note that the system dynamics are now incorporated into the propagation of the reference and filter states by the integration of the dynamics partial derivative in Equation 8.13.

Mathematically, the true difference state is propagated in a similar fashion with the addition of a process noise random value

$$\mathbf{x}_{k+1} = \Phi(t_{k+1}, t_k)\mathbf{x}_k + \mathbf{w}_k \tag{8.13}$$

In general, it is not required that the reference dynamic model be exactly the same as the truth dynamics or that the modeling parameter α be equivalent to the true modeling vector. This notation is left in place to simplify the derivation of the Kalman filter formulation. A number of innovative approaches have been developed for adapting reference model parameters to improve fidelity with the unknown real-world system model [2–6] and can be used to enhance filter operation.

The LKF also requires a linearized measurement, $\mathbf{y}_k = \mathbf{Y}_k - \tilde{\mathbf{Y}}_k$, modeled by

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{8.14}$$

For the development of the Kalman filters presented here, the random contributions $\mathbf{v}_k$ and $\mathbf{w}_k$ are assumed to be discrete realizations of the continuous zero mean Gaussian process in Equations 8.1 and 8.2 and are defined by

$$E\left[\mathbf{v}_k\mathbf{v}_i^T\right] = \mathbf{R}_k\delta_{ki} \tag{8.15}$$

$$E\left[\mathbf{w}_k\mathbf{w}_i^T\right] = \mathbf{Q}_k\delta_{ki} \tag{8.16}$$

Generally, it is assumed that the process noise and measurement noise sequences are uncorrelated so that

$$E\left[\mathbf{w}_k\mathbf{v}_i^T\right] = 0 \quad \forall k, \forall i \tag{8.17}$$

However, the Kalman filter can be configured to operate in systems where this assumption does not apply [7].

8.1.2 Linear Kalman Filter Error Covariance Propagation

The propagation of the filter and reference states in the LKF were outlined in the previous section in Equations 8.11 and 8.13. However, all Kalman filter formulations must also propagate a confidence metric of the state estimate in the form of a state error covariance matrix. The state error covariance, **P**, is defined as the expectation of the outer product of the estimation error vector

$$\mathbf{P}_k^{(\pm)} = E\left[\delta\mathbf{x}_k^{\pm}\delta\mathbf{x}_k^{\pm T}\right] \tag{8.18}$$

The state error covariance matrix is $n \times n$ and symmetric, and must remain positive definite to retain filter stability. The mechanism for propagating the covariance can be derived by taking the covariance just before measurement update at time t_{k+1}

$$\mathbf{P}_{k+1}^{(-)} = E\left[\delta\mathbf{x}_{k+1}^{(-)}\delta\mathbf{x}_{k+1}^{(-)T}\right] \tag{8.19}$$

and substituting the estimation error and propagation definitions in Equations 8.4, 8.11, and 8.13 to yield

$$\mathbf{P}_{k+1}^{(-)} = E\left[\Phi(t_{k+1}, t_k)\left(\hat{\mathbf{x}}_k^{(+)} - \mathbf{x}_k\right)\left(\hat{\mathbf{x}}_k^{(+)} - \mathbf{x}_k\right)^T\Phi(t_{k+1}, t_k)^T + \mathbf{w}_k\mathbf{w}_k^T\right] \tag{8.20}$$

Utilizing the definitions of process noise covariance in Equation 8.16 and state error covariance in Equation 8.18 the propagation equation reduces to

$$\mathbf{P}_{k+1}^{(-)} = \Phi(t_{k+1}, t_k)\mathbf{P}_k^{(+)}\Phi(t_{k+1}, t_k)^T + \mathbf{Q}_k \tag{8.21}$$

The propagation equation can be interpreted as the sum of the mapping of the previous post-update error covariance through the system dynamics and the system process noise induced uncertainty. Thus, process noise acts to increase the state error covariance between measurement updates.

8.1.3 Linear Kalman Filter Update

The LKF seeks an unbiased, minimum variance solution for the difference state, $\mathbf{x}_k$, by combining previous state information with available measurements. The state estimate after measurement update is therefore assumed to be a linear combination of the pre-update state and the linearized measurement information

$$\hat{\mathbf{x}}_k^{(+)} = \mathbf{K}_k^*\hat{\mathbf{x}}_k^{(-)} + \mathbf{K}_k\mathbf{z}_k \tag{8.22}$$

Substituting Equations 8.4 and 8.14 into Equation 8.22 and solving for the estimation error yields

$$\delta\mathbf{x}_k^{(+)} = (\mathbf{K}_k^* + \mathbf{K}_k\mathbf{H}_k - \mathbf{I})\mathbf{x}_k + \mathbf{K}_k^*\delta\mathbf{x}_k^{(-)} + \mathbf{K}_k\mathbf{v}_k \tag{8.23}$$

By definition $E[\mathbf{v}_k] = \mathbf{0}$ and $E[\,\delta\mathbf{x}_k^{(-)}\,] = \mathbf{0}$ by assumption of unbiased estimation. Therefore, the updated state estimation error is unbiased

$$E\left[\delta\mathbf{x}_k^{(+)}\right] = \mathbf{0} \tag{8.24}$$

only if

$$\mathbf{K}_k^* + \mathbf{K}_k\mathbf{H}_k - \mathbf{I} = \mathbf{0} \tag{8.25}$$

Substitution of Equation 8.25 into Equation 8.22 results in an expression for the updated state estimate

$$\hat{\mathbf{x}}_k^{(+)} = \hat{\mathbf{x}}_k^{(-)} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^{(-)}\right) \tag{8.26}$$

with estimation error

$$\delta\mathbf{x}_k^{(+)} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\delta\mathbf{x}_k^{(-)} + \mathbf{K}_k\mathbf{v}_k \tag{8.27}$$

The post-measurement error covariance in Equation 8.18 may be expanded to

$$\mathbf{P}_k^{(+)} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^{(-)}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \tag{8.28}$$

by substitution of Equation 8.27 and applying the conditions of uncorrelated process and measurement noise, zero mean measurement noise, and the definition of the pre-measurement state estimation error covariance. At this point, only the requirement that the Kalman filter be an unbiased estimator has been satisfied, so now we will select the Kalman gain $\mathbf{K}_k$ that delivers the minimum summed variance on the post-measurement state estimation error. In other words, we seek the gain that will minimize

$$J_k = trace\left[\mathbf{P}_k^{(+)}\right] \tag{8.29}$$

The necessary condition for minimality of J_k is that its partial derivative with respect to the Kalman gain is zero. By employing the following relationship:

$$\frac{\partial}{\partial\mathbf{A}}[\,trace(\mathbf{A}\mathbf{B}\mathbf{A}^T)] = 2\mathbf{A}\mathbf{B} \tag{8.30}$$

where $\mathbf{B}$ is a symmetric matrix, on the components of Equation 8.28 with respect to $\mathbf{K}_k$ results in

$$(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^{(-)}\mathbf{H}_k^T + \mathbf{K}_k\mathbf{R}_k = \mathbf{0} \tag{8.31}$$

The optimal gain (the Kalman gain) is therefore

$$\mathbf{K}_k = \mathbf{P}_k^{(-)}\mathbf{H}_k^T\left[\mathbf{H}_k\mathbf{P}_k^{(-)}\mathbf{H}_k^T + \mathbf{R}_k\right]^{-1} \tag{8.32}$$

which is sometimes written as

$$\mathbf{K}_k = \mathbf{P}_k^{(-)}\mathbf{H}_k^T\mathbf{W}_k^{-1} \tag{8.33}$$

where the term $\mathbf{W}_k$ is referred to as the innovations covariance

$$\mathbf{H}_k\mathbf{P}_k^{(-)}\mathbf{H}_k^T + \mathbf{R}_k \tag{8.34}$$

8.2 Other Kalman Filter Formulations

In addition to the LKF, there are several other formulations of the Kalman filter that may be employed to more closely follow the characteristics of specific state observation scenarios. The LKF may be varied according to the temporal nature of the dynamic and measurement systems to be continuous in dynamics and measurements or continuous in dynamics and discrete in measurements [12]. Also, there are applications when the dynamic system is energetic or the measurement quality is poor that may cause the reference state in the LKF to quickly leave the region of linearity about the environment state. In such systems, the reference state can be updated through addition of the filter state into an implementation known as the extended Kalman filter (EKF). The EKF is highly suited to real-time applications but is nonlinear in the sense that the reference state is essentially reinitialized at the time of each measurement update. Both the continuous–discrete LKF and EKF will be developed in the following sections.

8.2.1 The Continuous–Discrete Linear Kalman Filter

There may quite naturally arise an application where the reference state, filter state, and state error covariance are more suitably propagated in a continuous fashion than through the linear application of the state transition matrix. Also, it is common for the measurement system to deliver discrete-time observations even when the dynamics are best modeled continuously. In such a situation the update mechanization is unchanged from the previous LKF derivation while the propagation between updates is carried out through continuous integration. Without loss of generality, it may be stated that the reference dynamics of a continuous Kalman filter may be represented by

$$\dot{\hat{\mathbf{X}}}(t) = f(\tilde{\mathbf{X}}(t), \alpha, t) \tag{8.35}$$

Furthermore, by taking time derivatives of the filter state and covariance propagation (Equations 8.11 and 8.21) and substituting in Equation 8.13 for the derivative of the state transition matrix, the continuous-time filter state and covariance relations are found to be

$$\dot{\hat{\mathbf{x}}}^{(-)}(t) = f(\tilde{\mathbf{X}}(t), \alpha, t) + \mathbf{F}(\tilde{\mathbf{X}}(t), \alpha, t)[\hat{\mathbf{x}}^{(-)}(t) - \tilde{\mathbf{X}}(t)] \tag{8.36}$$

$$\dot{\mathbf{P}}(t) = \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(t) + \overline{\mathbf{Q}}(t) \tag{8.37}$$

where $\overline{\mathbf{Q}}(t)$ is the spectral density of the dynamic process noise at time t and the explicit functional dependency of $\mathbf{F}$ was dropped for notational convenience. In this mechanization of the LKF, the state transition matrix need not be calculated as the dynamics are included directly via the partial derivative matrix and the reference state, filter state, and error covariance are propagated continuously.

The process and measurement noise representations in this formulation are continuous and discrete for the respective models, and are again assumed to be zero mean processes governed by the continuous dynamic process noise covariance

$$E[\mathbf{w}(t)\mathbf{w}^T(\tau)] = \mathbf{Q}(t)\delta(t-\tau) \tag{8.38}$$

and the discrete measurement noise covariance

$$E\left[\mathbf{v}_k\mathbf{v}_j^T\right] = \mathbf{R}_k\delta_{kj} \tag{8.39}$$

It is also assumed here that the process and measurement noises are uncorrelated so that

$$E\left[\mathbf{w}(t)\mathbf{v}_k^T\right] = 0 \tag{8.40}$$

although the formulation can be modified to accommodate process and measurement noise correlations if necessary [7].

8.2.2 The Continuous–Discrete Extended Kalman Filter

In applications where the reference state may quickly deviate beyond the linear region of the environment state, the reference may be directly updated at the time of measurement update by adding the LKF filter state to the reference in an EKF. The EKF is similar to the LKF, in that measurements are processed to provide an estimate of the difference between the true state and reference state of the spacecraft. Also, the EKF evaluates dynamics and measurement partials with respect to the reference state in a manner similar to the LKF. However, the reference state about which these partials are evaluated is modified through the addition of measurement information

$$\tilde{\mathbf{X}}(t_k)^{(+)} = \tilde{\mathbf{X}}(t_k)^{(-)} + \hat{\mathbf{x}}(t_k) \tag{8.41}$$

The reference state dynamics model used in the EKF formulation is given by Equation 8.35, but the measurement model is the discrete form given by Equation 8.1. The filter state representing the estimated difference between the true state and the reference state is only calculated at the time of measurement update via dropping the previous estimate information term from Equation 8.26:

$$\hat{\mathbf{x}}_k = \mathbf{K}_k \mathbf{Z}_k \tag{8.42}$$

where the innovation is now the actual measurement residual

$$\mathbf{Z}_k = \mathbf{Y}_k - \mathbf{h}\left(\tilde{\mathbf{X}}_k^{(-)}, \beta, t_k\right) \tag{8.43}$$

Therefore, in the EKF there is not a separate filter state that needs to be propagated to the time of the next measurement, as the filter state has been incorporated into the updated reference state.

As before, the error covariance at each measurement is updated by

$$\mathbf{P}_k^{(+)} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^{(-)}(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T, \tag{8.44}$$

and the EKF Kalman gain and innovations covariance are analogous to their LKF counterparts

$$\mathbf{K}_k = \mathbf{P}_k^{(-)} \mathbf{H}_k \mathbf{W}_k^{-1} \tag{8.45}$$

$$\mathbf{W}_k = \mathbf{H}_k \mathbf{P}_k^{(-)} \mathbf{H}_k + \mathbf{R}_k. \tag{8.46}$$

The difference between EKF operation and LKF operation is illustrated by revisiting the two-dimensional trajectory illustration in Figure 8.2. The reference trajectory can now be seen to respond to measurement information availability and tracks the true environment trajectory.

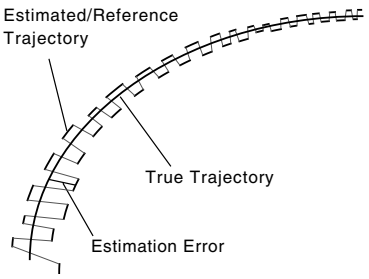


FIGURE 8.2 EKF tracking of a two-dimensional trajectory.

8.3 Formulation Summary and Review

The LKF discrete–discrete formulation was given by the following propagation equations:

$$\begin{aligned}\tilde{\mathbf{X}}_{k+1} &= \Phi(t_{k+1}, t_k)\tilde{\mathbf{X}}_k \\ \hat{\mathbf{x}}_{k+1}^{(-)} &= \Phi(t_{k+1}, t_k)\hat{\mathbf{x}}_k^{(-)} \\ \mathbf{P}_{k+1}^{(-)} &= \Phi(t_{k+1}, t_k)\mathbf{P}_k^{(-)}\Phi(t_{k+1}, t_k)^T + \mathbf{Q}_k\end{aligned}$$

and update equations

$$\begin{aligned}\hat{\mathbf{x}}_k^{(+)} &= \hat{\mathbf{x}}_k^{(-)} + \mathbf{K}_k\left[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^{(-)}\right] \\ \mathbf{P}_k^{(+)} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^{(-)}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \\ \mathbf{K}_k &= \mathbf{P}_k^{(-)}\mathbf{H}_k^T\left[\mathbf{H}_k\mathbf{P}_k^{(-)}\mathbf{H}_k^T + \mathbf{R}_k\right]^{-1}\end{aligned}$$

In the discrete time LFK mechanization, the reference state is unaffected by the incorporation of measurement information into the filter state.

In a slight variation of this approach, the dynamics of the LKF may be made continuous, and the filter state, reference state, and covariance propagated without the use of a state transition matrix.

$$\begin{aligned}\dot{\tilde{\mathbf{X}}}(t) &= f(\tilde{\mathbf{X}}(t), \alpha, t) \\ \dot{\hat{\mathbf{x}}}^{(-)}(t) &= \mathbf{F}(\tilde{\mathbf{X}}(t), \alpha, t)\hat{\mathbf{x}}^{(-)}(t) \\ \dot{\mathbf{P}}(t) &= \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(t) + \overline{\mathbf{Q}}(t)\end{aligned}$$

When the application requires that the reference state be modified to remain in the linear vicinity of the environment state, the EKF continuous–discrete formulation may be appropriate. In the continuous–discrete EKF formulation, the propagation is carried out according to

$$\begin{aligned}\dot{\tilde{\mathbf{X}}}(t) &= f(\tilde{\mathbf{X}}(t), \alpha, t) \\ \dot{\mathbf{P}}(t) &= \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}^T(t) + \overline{\mathbf{Q}}(t)\end{aligned}$$

and the measurement update according to

$$\begin{aligned}\tilde{\mathbf{X}}(t_k)^{(+)} &= \tilde{\mathbf{X}}(t_k)^{(-)} + \hat{\mathbf{x}}(t_k) \\ \hat{\mathbf{x}}_k &= \mathbf{K}_k\left[\mathbf{Y}_k - \mathbf{h}\left(\tilde{\mathbf{X}}_k^{(-)}, \beta, t_k\right)\right] \\ \mathbf{P}_k^{(+)} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^{(-)}(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \\ \mathbf{K}_k &= \mathbf{P}_k^{(-)}\mathbf{H}_k^T\left[\mathbf{H}_k\mathbf{P}_k^{(-)}\mathbf{H}_k^T + \mathbf{R}_k\right]^{-1}\end{aligned}$$

The reference state will change with the incorporation of measurement information into the EKF and the partials evaluated along this changing reference.

8.4 Implementation Considerations

It is commonly held among designers of Kalman filters that the implementation of the formulas listed above represent only a portion of the effort required to develop an accurate and robust Kalman filter application. Once the dynamics, measurements, and partial derivatives have been coded, the task remains to tune the noise magnitudes represented in the process noise covariance **Q** and the measurement noise covariance **R**. While the measurement noise can be based in realistic hardware performance specifications, the process noise is often used as a tuning parameter to ensure that the filter operates correctly. This process of tuning the filter crosses over into the area of design and is nearly an art form of such myriad approaches that it is beyond the scope of this work to outline. However, a Kalman filter checklist is provided for the newcomer to the field to reduce the time of the implementation and tuning learning curve:

- Because the linear Kalman filter does not change the reference state in the presence of measurement information, the reference state and partial derivatives for an LKF application may be computed prior to operation. This makes the LKF more amenable to computationally restricted applications or hypothesis testing where differing process noise and measurement noise parameters are being evaluated in parallel [8].
- Process noise serves to keep the filter from becoming overconfident in its estimate (i.e., a covariance with near zero diagonal values) and converging prematurely. Examining the propagation equations for the Kalman filters presented previously, it can easily be seen how the addition of process noise increases the magnitude of the state error covariance between measurements.
- The innovations covariance should ideally converge to describe the variance in the filter measurement residuals. Adaptive techniques have been implemented where the filter noise parameters are tuned according to a metric linking residual statistics with the innovations covariance [5]. In an ideal filter, the innovations covariance should approach the measurement noise covariance as the process noise magnitude approaches zero.
- When multiple measurements are available at the same time, they may be processed as a series of scalar observations as long as they are uncorrelated (i.e., **R** is a diagonal matrix). The effect of processing scalar measurements is that the innovations covariance becomes a scalar, and a numerical division rather than a matrix inversion is required to calculate the Kalman gain.
- Measurement editing may be employed to prevent spurious data from causing filter divergence in a number of ways. One of the most common is to reject measurements when the ratio of the measurement residual squared to the scalar innovations covariance

$$\frac{r_k^2}{W_k} \tag{8.47}$$

  is above a user-defined threshold. The threshold value may either be a constant or may be time varying after long propagation periods to allow for a smooth transition to a steady state innovations covariance.
- The covariance should always be positive definite. If filter divergence is a chronic problem in a particular application, the numerical integrity of the covariance may provide insight into the nature of the divergence. There are also several numerical implementations of the covariance update equation that take advantage of its symmetry and positive definiteness to enhance its stability while reducing computational load [9].
- Process noise may be enhanced by including time correlated states such as first-order Gauss–Markov processes to the filter to account for specific dynamic effects. The biases associated with these processes can be included in the filter state for estimation.

As a final note it should be stressed that the Kalman filter is not the state observer algorithm best suited for all applications. Its strengths lie in light computational requirements and real-time availability

of a state estimate in the presence of accurate measurement information. However, batch estimation techniques such as least-squares estimation may be more appropriate in applications where the dynamic process is modeled to a high degree of fidelity, measurements are not uniformly accurate, and real-time operation is not an issue. A number of quality texts [10–12] have been written on the subject of stochastic estimation in general and specifically Kalman filtering that the the reader is encouraged to pursue for more detailed information.

References

1. Kalman, R. E., "A new approach to linear filtering and prediction problems," *Transactions of the ASME, Ser. D, Journal of Basic Equations,* March 1960, pp. 35–45.
2. Burkhart, P. and Bishop, R., "Adaptive orbit determination for interplanetary spacecraft," *Journal of Guidance, Control, and Dynamics,* Vol. 19, No. 3, 1997, pp. 693–701.
3. Chaer, W., Bishop, R., and Ghosh, J., "Hierarchical adaptive Kalman filtering for interplanetary orbit determination," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 34, No. 3, 1998, pp. 1–14.
4. Crain, T. and Bishop, R., "The mixture-of-experts gating network: integration into the ARTSN extended Kalman filter," Technical Memorandum CSR-TM-99-01, Center for Space Research, March 1999.
5. Ely, T., Bishop, R., and Crain, T., "Adaptive interplanetary navigation using genetic algorithms," *The Journal of Astronautical Sciences,* 2000, Accepted for Publication.
6. Crain, T. and Bishop, R., "Unmodeled impulse detection and identification during Mars pathfinder cruise," Technical Memorandum CSR-TM-00-01, Center for Space Research, March 2000.
7. Chaer, W. and Bishop, R., "Adaptive Kalman filtering with genetic algorithms," *Advances in the Astronautical Sciences,* edited by R. Proulx, J. Liu, P. Siedelmann, and S. Alfano, Vol. 89, Univelt, San Diego, CA, 1995, pp. 141–156, Pt. 1.
8. Gholson, N. and Moose, R., "Maneuvering target tracking using adaptive state estimation," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 13, No. 3, May 1997, pp. 310–317.
9. Bierman, G., *Factorization Methods for Discrete Sequential Estimation,* Academic Press, 1977.
10. Brown, R. G. and Huang, P. Y. C., *Introduction to Random Signals and Applied Kalman Filtering,* John Wiley & Sons, 1992.
11. Lewis, F., *Applied Optimal Control and Estimation,* Prentice-Hall, Englewood Clifis, NJ, 1992.
12. Gelb, A., *Applied Optimal Estimation,* The MIT Press, Cambridge, MA, 1974.

9

Digital Signal Processing for Mechatronic Applications

Bonnie S. Heck
Georgia Institute of Technology

Thomas R. Kurfess
Clemson University

9.1 Introduction

Most engineers work in the world of mechatronics as there are relatively few systems that are purely mechanical or electronic. There are a variety of means by which electrical systems augment mechanical systems and vise versa. For example, most microprocessors found in a computer today have some sort of heat sink and perhaps a fan attached to them to keep them within their operational temperature zone. Electrical systems are widely employed to monitor and control a wide variety of mechanical systems. With the advent of inexpensive digital processing chips, digital filtering and digital control for mechanical systems is becoming commonplace. Examples of this can be seen in every automobile and most household appliances. For example, sensor signals used in monitoring and controlling of mechanical systems require some form of signal processing. This signal processing can range from simply "cleaning-up" the signal using a low pass filter to more advanced analyses such as torque and power monitoring in a DC servo motor. This chapter presents a brief overview of digital signal processing methods suitable for mechanical systems. Since this chapter is limited in space, it does not give any derivation or details of analysis. For a more detailed discussion, see references [1,2].

9.2 Signal Processing Fundamentals

A few fundamental concepts on signal processing must be introduced before a discussion of filtering or control can be undertaken.

9.2.1 Continuous-Time Signals

Laplace transforms are used for system analysis of continuous-time systems, solving for system response, and control design. The single-sided Laplace transform of a continuous-time signal, $x(t)$, is given by

$$X(s) = \int_0^\infty x(t)e^{-st}\,dt$$

A transfer function of a linear system, $H(s)$, can be found as the ratio of the Laplace transforms of the output over that of the input (with zero initial conditions).

The Fourier transform is used to determine the frequency content of a signal. The Fourier transform of $x(t)$ is given by

$$X(\omega) = \int_{-\infty}^\infty x(t)e^{-j\omega t}\,dt \tag{9.1}$$

where ω is in units of radians per second. Notice that when $x(t) = 0$ for $t \le 0$, the Laplace transform is equivalent to the Fourier transform by setting $s = j\omega$. (It should be noted that there are some additional convergence considerations for the Fourier transform.) The frequency response of a system is defined as the ratio of the Fourier transforms of the output over that of the input. Equivalently, it can be found from the transfer function as $H(\omega) \equiv H(j\omega) = H(s)|_{s=j\omega}$. For simplicity of notation, the j is usually not shown in the argument list, giving rise to the notation $H(\omega)$ to represent the frequency response. The bandwidth of a system is defined as the frequency at which $H(\omega) = 0.707H(0)$.

9.2.2 Discrete-Time Signals

The z-transform is useful for solving a difference equation and for performing system analysis. The z-transform of a discrete-time signal, $x[n]$, is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

The discrete-time Fourier transform (DTFT) is used to determine the frequency content of a signal. The DTFT and the inverse DTFT of a signal are defined by

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n} \tag{9.2}$$

and

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\Omega)e^{j\Omega n}\,d\Omega \tag{9.3}$$

Note that the DTFT can be derived from the z-transform by setting $z = e^{j\Omega}$. (Again, there are some assumptions on convergence in this derivation.) Since the DTFT is periodic with period 2π, it is typically displayed over the range $[-\pi, \pi]$ or $[0, 2\pi]$, where the frequencies of general interest are from $\Omega = 0$ (low frequency) to $\Omega = \pi$ (high frequency). The frequency response of a discrete-time system can be found as the ratio of the DTFT of the output signal over that of the input signal. Alternatively, it can be found from the transfer function as $H(\Omega) \equiv H(e^{j\Omega}) = H(z)|_{z=e^{j\Omega}}$. The notation $H(\Omega)$ is preferred over $H(e^{j\Omega})$ for its simplicity. As in the continuous-time case, the bandwidth is defined as the frequency at which $H(\Omega) = 0.707H(0)$.

While the DTFT is continuous with respect to the frequency variable Ω, the discrete Fourier transform (DFT) contains points that are discrete with respect to a parameter k. Consider a finite duration sequence $x[n]$, where $x[n] = 0$ for $n < 0$ and for $n \geq N$. The DFT of $x[n]$ and the inverse DFT are defined as

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \quad k = 0, 1, \ldots, N-1 \tag{9.4}$$

and

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi nk/N}, \quad n = 0, 1, \ldots, N-1$$

Note that the DFT is a discretized version of the DTFT where $X_k = X(\Omega)\big|_{\Omega=2\pi k/N}$ over the range $\Omega = 0$ to $\Omega = 2\pi$. Calculating a closed-form solution for the DTFT can be done only for simple signals such as a square pulse or a triangular pulse. Therefore, the DFT is generally used as a numerical method to calculate the DTFT at discrete points in frequency in the range $0 \leq \Omega \leq 2\pi$. In particular, to obtain a plot of the DTFT, plot X_k versus k where k is scaled by $2\pi/N$. For an arbitrary signal, such as obtained from measurements of a physical device, computing the DFT instead of the DTFT is the preferred method to find the frequency content of the signal. To get more resolution in plotting a DTFT from the points calculated by a DFT, zeros can be added to the end of the sequence so that the value of N is increased.

Suppose a time domain signal is not finite in duration, so that there is no value of N such that $x[n] = 0$ for n Š N. In order to perform the DFT, the signal must be truncated. There are two cases to be considered: the case where $x[n]$ is decaying to zero and the case where $x[n]$ has periodic components. The case when $x[n]$ decays to zero is handled by choosing N to be large enough so that the signal is negligible beyond that value. The resulting DFT is an approximation (not a discretized version) of the DTFT. If the signal is periodic, the DTFT cannot be computed numerically since the resulting DTFT would have impulses in it. However, the frequencies present in the signal could still be determined if the value of N used for the truncation is chosen so that the truncated signal goes through an integer number of cycles. If this not done, the resulting DFT will have leakage in the frequency plot when compared to the DTFT of the true signal. For example, consider a signal $x[n] = \cos(0.4\pi n)$. This is periodic with period $n = 5$ and has DTFT given by $X(\Omega) = \pi[\delta(\Omega + 0.4\pi) + \delta(\Omega - 0.4\pi)]$ for $-\pi \leq \Omega \leq \pi$. All the frequency content is located at $\Omega = 0.4\pi$ and $\Omega = -0.4\pi$. Since the DTFT is periodic with 2π, there is also an impulse at $\Omega = 2\pi - 0.4\pi$. The DFT is computed for two truncations of the signal, one at $N = 20$ (four complete cycles) and the other at $N = 22$. The DFT for $N = 20$ is plotted in Figure 9.1a where the independent variables k are scaled by $2\pi/N$ for the plot. This plot shows zero frequency content except at $\Omega = 0.4\pi$ (=1.2566) and $\Omega = 2\pi - 0.4\pi$ (=5.0265), giving the correct location of the impulses in the DTFT. Similarly, the DFT for $N = 22$ is plotted in Figure 9.1b, notice the resulting leakage in the frequency characteristics.

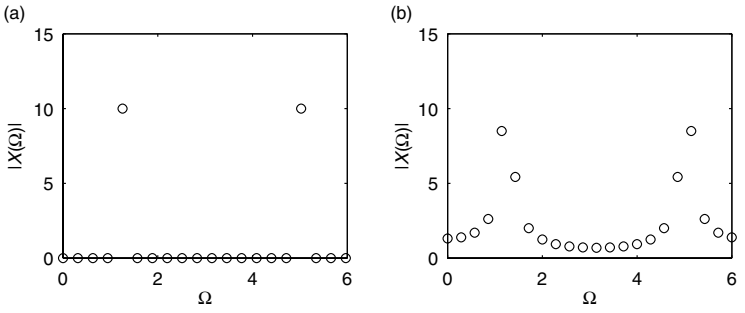


FIGURE 9.1 DFT of periodic signal (a) truncated after 4 complete cycles and (b) truncated after 4.4 cycles.

If a signal has periodic content, but is not periodic, such as $x[n] = \cos(0.5\pi n) + \cos(0.2n)$, then leakage cannot be avoided by a selection of N. An alternate means of reducing leakage is to first taper the signal to zero at the initial and end points of the sequence prior to computing the DFT. This process, known as *windowing* the data, is accomplished by multiplying $x[n]$ by a window function $w[n]$ and then performing the DFT on the product $x[n]w[n]$. Three common windows are the rectangular window, which is a sharp truncation, the Hanning window, and the Hamming window [1].

Rectangular Window:

$$w[n] = 1, \quad 0 \le n \le N-1$$

Hanning Window:

$$w[n] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right), \quad 0 \le n \le N-1$$

Hamming Window:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1$$

If the value of N in a DFT is a power of 2, there is a fast method to compute the DFT called the fast Fourier transform (FFT). If the value of N is not a power of 2, zeros can be padded to the end of the signal in order to use the FFT. This does not affect the accuracy of the result, but it does improve the resolution of the resulting plot when the DFT (or FFT) is used to compute the DTFT. In many cases, the expression used in Equation 9.4 suffices to compute the DFT since the added computational power of today's processors lessens the need for the numerical efficiency of the FFT. The details of the algorithm for the FFT are beyond the scope of this handbook. See [1] or [2] for details.

9.3 Continuous-Time to Discrete-Time Mappings

While most physical systems operate in continuous-time, computers operate in discrete-time. Therefore, in order to use computers to process measurements taken from continuous-time systems, there must be ways of mapping between the continuous-time world to the discrete-time world.

9.3.1 Discretization

Before an analog signal can be analyzed using digital techniques, it must be discretized (i.e., converted into a discrete-time signal). The ideal method for discretization is *sampling*, where the values of the signal are determined at discrete points in time. Generally, the signal is sampled at a fixed rate known as the *sampling period*. The sampling rate (in hertz) is the inverse of the sampling period. Figure 9.2 depicts a 1 Hz signal that has been sampled at two rates. The dark points are sampled at 15 ms intervals, while the lighter points are sampled at 250 ms intervals. From Figure 9.2, the waveform approximation clearly degrades as the sampling frequency is reduced and approaches the signal frequency. In fact, it can be shown that a signal must be sampled at a frequency that is higher than twice its maximum frequency content. This is known as the Nyquist Sampling Theorem. For example, if the signal in Figure 9.2 is sampled at 0.5 Hz, it is possible for every sample to have a value of 0 as at 0, 500, 1000 ms, etc… the value of the signal is 0. The erroneous interpretation of the signal due to a sample frequency that is too low is known as aliasing. There are two means by which the Nyquist Sampling Theorem can be satisfied. The first is by employing a sample frequency that is more than twice the highest frequency content of the signal being sampled. This value frequency is known as the Nyquist frequency. As one never is sure of the actual frequency content of a real signal, a low pass filter may be used to ensure that a signal does not possess frequencies above a certain cut-off level. Such a filter is commonly

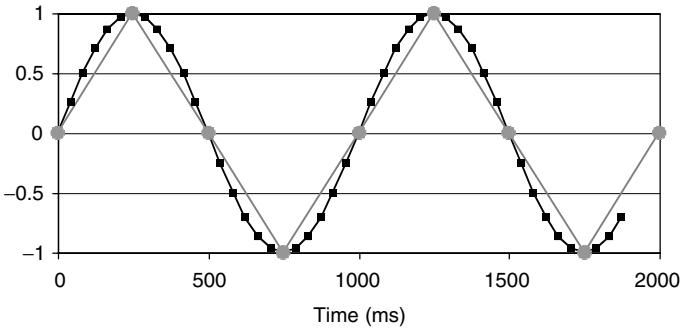


FIGURE 9.2 A 1 Hz signal.

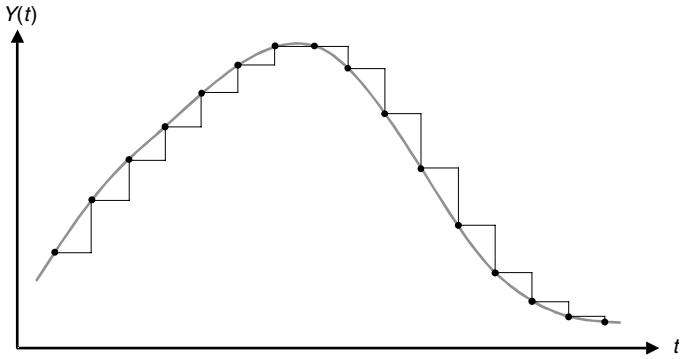


FIGURE 9.3 A signal sampled and reconstructed using a zero order hold.

called an anti-aliasing filter. This is the second and most practical method that is used to satisfy the Nyquist Sampling Theorem. Thus, a combination of a well-designed anti-aliasing filter as well as a sampling frequency that is well above the cut-off frequency of the filter will ensure that the Nyquist Sampling Theorem is satisfied.

There are two important points that should be noted when using an anti-aliasing filter. First, it is important that the anti-aliasing filter be used before the signal is sampled as sampling is what causes aliasing. Basically, this requires that the anti-aliasing filter is implemented using an analog filter prior to the signal being digitized. Once a signal has been aliased during sampling, it cannot be corrected using digital filtering. The second point is that, in practice, the cutoff frequency of the anti-aliasing filter should be a factor of 5–10 below the value of the Nyquist frequency. It should be noted that an anti-aliasing filter adds phase lag to the measurement, which might deteriorate stability and performance in a feedback loop unless the bandwidth of the anti-aliasing filter is much higher than that of the closed loop system. Commercially available devices that perform sampling are analog-to-digital converters (ADCs), and the anti-aliasing filter is used before this device.

The converse of sampling is reconstruction where a discrete-time signal is converted into a continuous-time signal. The Nyquist sampling rate ensures that if a continuous-time signal is sampled at a rate that is at least twice the highest frequency component in the signal, then the continuous-time signal can be reconstructed exactly from the samples. However, this theorem assumes that an ideal reconstruction process is available, which is not practical. The most common practical means to reconstruct a signal is a zero-order hold (ZOH). The ZOH assumes that the value of the signal is constant between samples. This approximation is quite reasonable if the sampled signal does not change substantially between individual samples. Figure 9.3 is an example of a signal and its ZOH representation. The gray, smooth line represents the original analog signal. The black points along the signal indicate sample values of the signal. Each black

point is connected to the next via a horizontal then vertical straight line. The horizontal line is representative of the ZOH assumption that the value of the signal remains constant between samples. The vertical line is the reality that the signal does not remain constant over the sample period. As the time between sample points is increased, the accuracy of the ZOH decreases. Conversely, as the sample period is decreased, the accuracy of the ZOH is improved. Commercially available devices that perform reconstruction are digital-to-analog converters (DACs), which generally use the ZOH method.

9.3.2 *s*-Plane to *z*-Plane Mappings

One method to relate the *s*-plane to the *z*-plane is to derive a continuous-time mathematical representation of the sampled signal for $x(t)$ and compute its Laplace transform. The resulting Laplace transform of the sampled signal can be related to the *z*-transform of $x[n]$ by setting $z = e^{sT}$ where T is the sampling period. The relationship $z = e^{sT}$ is commonly termed the *exact mapping* between the *z*-plane and the *s*-plane. (For details of this derivation, see [1]). For example, a digital representation, $H_d(z)$, of a continuous-time system, $H(s)$, can be obtained using this mapping $H_d(z) = H(s)\big|_{z=e^{sT}}$. However, this mapping results in a nonrational function for $H_d(z)$.

Approximate mappings between the *s*-plane and the *z*-plane are commonly used that do result in a rational function for $H_d(z)$. Three such mappings are the bilinear transformation, forward transformation, and backward transformation.

Bilinear transformation:

$$s = \frac{2(z-1)}{T(z+1)}$$

Forward transformation:

$$s = \frac{1}{T}(z-1)$$

Backward transformation:

$$s = \frac{1}{Tz}(z-1)$$

The bilinear transformation (also known as Tustin's rule or the trapezoidal rule) is the most accurate of these mappings. It maps the entire left-hand side of the *s*-plane into the unit circle of the *z*-plane, so that it preserves stability. Consider a first-order example of $H(s) = 1/(s+2)$. The discrete-time representation of this transfer function is

$$H_d(z) = H(s)\Big|_{s=\frac{2(z-1)}{T(z+1)}} = \frac{T(z+1)}{2z-2+2T}$$

Note that the resulting transfer function is rational in *z*.

An alternate method of mapping transfer functions between the continuous-time and the discrete-time domains is the response-matching mapping.

Response-matching: Suppose $x(t)$ is the input to a system $H(s)$ with the resulting output $y(t)$. Let $x[n]$ and $y[n]$ be the sampled versions of $x(t)$ and $y(t)$. Then, $H_d(z)$ is found from the ratio of *z*-transforms of $x[n]$ and $y[n]$. The most common response matching is step-response matching where $x(t)$ is a step function, $x(t) = 1$ for $t \geq 0$ and $x(t) = 0$ for $t < 0$. An expression for $H_d(z)$ is found from the following operation:

$$H_d(z) = (1 - z^{-1})Z\left[\frac{H(s)}{s}\right]$$

where $Z[H(s)/s]$ represents the z-transform of the sampled version of the step response of the continuous-time system. The form for a generic first-order system is given below:

$$H(s) = \frac{a}{s+a} \Leftrightarrow H_d(z) = \frac{(1-e^{-aT})z^{-1}}{1-e^{-aT}z^{-1}}$$

The response matching method (especially with step inputs) is commonly used to map a continuous-time plant to discrete-time when designing a digital controller in the discrete-domain. Since most digital controllers are implemented using a ZOH on the output of the digital controller, the plant sees a stepped signal, one that looks like a sum of delayed step signals. Therefore, the step-response matching method is the most accurate way to map a plant that has a ZOH on its input.

9.3.3 Frequency Domain Mappings

The continuous-time Fourier transform can be related to the DTFT through the expression:

$$X(\omega) = TX(\Omega)\big|_{\Omega=\omega T} \quad \text{for } -\pi \le \Omega \le \pi$$

where $X(\omega)$ is defined in Equation 9.1 and represents the continuous-time Fourier transform of $x(t)$, while $X(\Omega)$ is defined in Equation 9.2 and represents the DTFT of the sampled signal $x[n]$. This mapping is very useful for computing the Fourier transform of measured data. In particular, suppose a continuous-time signal is measured by sampling it through an ADC and storing it as a discrete-time sequence. If the signal $x(t)$ is finite in duration, the DTFT of $x[n]$ can be computed at discrete points in frequency by using the DFT, X_k, as given in Equation 9.4. Using the relationships $\omega = \Omega/T$, $\Omega = 2\pi k/N$, and $X_k = X(\Omega)\big|_{\Omega=2\pi k/N}$, where N is the length of the sequence for $x[n]$ and T is the sampling period, gives the relationship:

$$X(\omega)\big|_{\omega=2\pi k/NT} = TX_k \quad \text{for } 0 \le \omega \le \omega_s/2,\ 0 \le k \le (N-1)/2$$

where $\omega_s = 2\pi/T$ is the sampling frequency in radian per second. Accuracy can be improved by decreasing the sampling period T, and the resolution in the plot can be increased by increasing NT.

If the signal $x(t)$ is not finite in duration, it must be truncated in order to use this numerical method to calculate the continuous-time Fourier transform. As discussed in Section 9.2.2, if the sampled signal $x[n]$ decays to zero, choose the number of sampled points N to be large enough so that $x(t)$ is negligible beyond that value. If the sampled signal $x[n]$ is periodic, choose the sampling period T and the number of points N such that the sampled signal $x[n]$ goes through an integer number of cycles. For example, consider the signal $x(t) = \cos(\pi t)$. If the sampling period is chosen as $T = 0.4$ s, the discretized signal would be $x[n] = x(nT) = \cos(0.4\pi n)$, which is the same signal analyzed in Section 9.2.2. Choosing $N = 5$, 10, 15, etc. would yield correct results in the DFT while any other value would result in leakage. If the signal $x(t)$ has periodic content, but does not appear to be periodic, then use a windowing function as discussed in Section 9.2.2 to reduce leakage when computing the DFT.

Note that the DFT can also be used to determine the Fourier coefficients of periodic signals. Consider a Fourier series in the form

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j\omega kt}$$

Sample the signal $x(t)$ by first making sure that the sampled signal goes through an integer number of cycles. The coefficients c_k for $k = 0,\ldots,(N-1)/2$ can be found as $c_k = X_k/N$ where X_k is found from the DFT. The rest of the coefficients are obtained from $c_{-k} = c_k^*$.

Since the frequency response of a system is the ratio of the Fourier transforms of the output over the input, a mapping between a continuous-time system, $H(\omega)$, and a corresponding discrete-time system, $H_d(\Omega)$, can be derived from the previous mapping as

$$H(\omega) = H_d(\Omega)\big|_{\Omega=\omega T} \quad \text{for } -\pi \le \Omega \le \pi$$

This mapping is useful for the design of both digital filters and digital controllers.

9.4 Digital Filter Design

The frequency response function of a discrete-time system describes how the system processes input signals of different frequencies. Consider an input signal $x[n] = A\cos(\Omega_0 n)$ to a system with frequency response $H(\Omega)$ where $0 \eth \Omega_0 \eth 2\pi$. The corresponding output is given by

$$y[n] = |H(\Omega_0)|\cos(\Omega_0 n + \angle H(\Omega_0))$$

For aperiodic signals, the filtering property of Fourier transforms gives the relationship:

$$Y(\Omega) = H(\Omega)X(\Omega)$$

Thus, if $|H(\Omega)|$ is small over a certain range of frequencies, then input signals with frequency content in that range are attenuated as they pass through the system.

It is often convenient to filter continuous-time signals through a digital filter as shown in Figure 9.4. The analog-to-digital converter (ADC) samples the continuous-time signal creating a sequence of discrete-time signals for processing by the computer or digital signal processing board. The filtered signal can be stored digitally for further study or it can be sent through a digital-to-analog converter (DAC). The digital filter can be implemented in software by a recursive equation obtained from the difference equation. Consider a digital filter with transfer function:

$$H(z) = \frac{b_1 z^N + b_2 z^{N-1} + \cdots + b_{N+1}}{a_1 z^N + a_2 z^{N-1} + \cdots + a_{N+1}} = \frac{b_1 + b_2 z^{-1} + \cdots + b_{N+1} z^{-N}}{a_1 + a_2 z^{-1} + \cdots + a_{N+1} z^{-N}}$$

The recursion used to calculate the current value of the output $y[n]$ is given by the difference equation:

$$y[n] = \frac{1}{a_1}(b_1 x[n] + b_2 x[n-1] + \cdots + b_{N+1} x[n-N] - a_2 y[n-1] - \cdots - a_{N+1} y[n-N]) \qquad (9.5)$$

Notice that the past values of y and x must be stored for use in the recursion.

Now consider the impulse response of a digital filter, where $y[n]$ is calculated for an input $x[n]$ equal to an impulse (that is, $\delta[n] = 1$ when $n = 0$ and $\delta[n] = 0$ otherwise). The recursion shown above results in a response for $y[n]$ that has infinite duration (that is, there is no value of M so that $y[n] = 0$ for all $n > M$). This type of filter is called an *infinite impulse response* (*IIR*) *filter*.

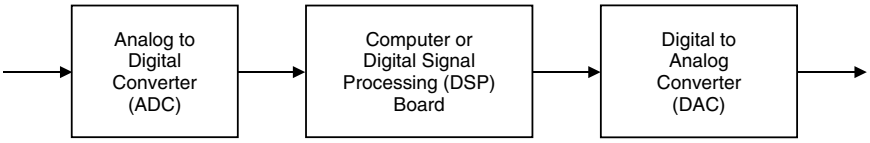


FIGURE 9.4 Configuration for standard digital signal processing hardware.

Now consider the case where the coefficients of the filter $a_m = 0$ for $m > 1$. The resulting expression for $y[n]$ from Equation 9.5 would no longer be recursive since it would depend only on present and past values of x, and not on past values of y. As a result, the impulse response would have duration N. This type of filter is called *a finite impulse response* (*FIR*) *filter*. FIR filters are sometimes preferred over IIR filters since they have linear phase in the frequency response. Linear phase means that the angle of the frequency response is given by $-\theta\Omega$, where θ is a constant. This corresponds to a delay in the time domain. Design methods for both types of filters are described in the next two sections.

9.4.1 IIR Filter Design

The two methods for designing IIR filters are termed *analog emulation* (or indirect design) and *direct design*. Analog emulation involves designing an analog filter first and then using one of the mapping techniques described in Section 9.3.2 to convert it to a digital filter. This method has advantages in that there is a wealth of design techniques for analog filters that can be used in digital filter design. Direct design methods generally involve numerical techniques, and they are often preferred over analog emulation when the sampling period is not very small. Direct design is beyond the scope of this handbook; consult reference [2] for more information on the topic.

Analog filter design begins by selecting a bandwidth, a prototype of filter, and an order of the filter. Additional specifications may be set on the amount of ripple that is allowed in the passband or stopband. Two common analog prototypes are the Butterworth filters and the Chebyshev filters.

Butterworth filter: The Butterworth filter is characterized by having no zeros and having poles that are situated on a semicircle in the left-half of the s-plane. The distance of the poles to the origin is the bandwidth frequency and is denoted as ω_b. The angle of the poles can be determined by equally spacing out twice the number of poles around a full circle with radius ω_b and then keeping only the poles in the left-half plane, as shown in Figure 9.5. An Nth order Butterworth filter is given by

$$H(s) = \frac{\omega_b^N}{\prod_k (s - \omega_b p_k)}$$

where

$$p_k = \begin{cases} e^{jk\pi/N}, & k = \dfrac{N+1}{2} \text{ to } \dfrac{3N-1}{2} \quad \text{for } N \text{ odd} \\ e^{j(k+0.5)\pi/N}, & k = \dfrac{N}{2} \text{ to } \dfrac{3N-2}{2} \quad \text{for } N \text{ even} \end{cases}$$

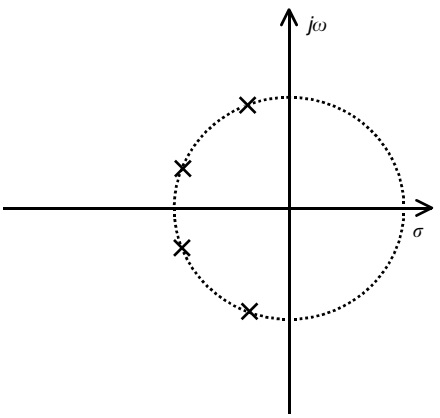


FIGURE 9.5 Pole distribution of a fourth-order Butterworth filter.

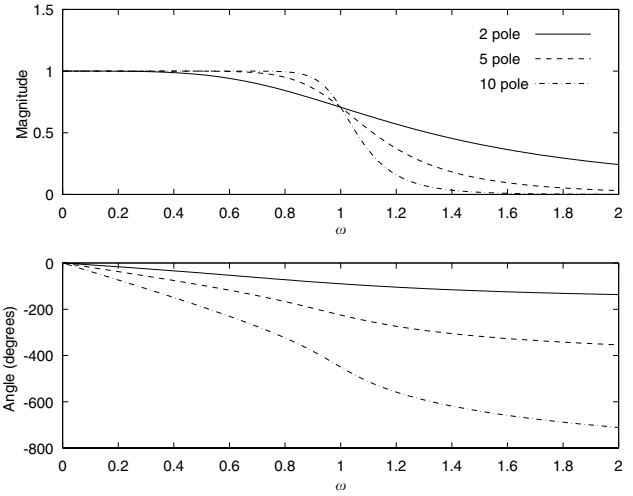


FIGURE 9.6 Comparison of analog Butterworth filters.

This filter is lowpass in that the magnitude of the frequency response is reasonably flat and close to a value of 1 for $\omega < \omega_b$ and drops off sharply beyond the bandwidth frequency. The larger the order of the filter, the sharper the drop off. Three filters are compared in Figure 9.6. Notice that the sharp transitions offered by the larger order filters do come with a price: the phase is also decreased dramatically. The phasing becomes important in real-time measurement systems such as that required by feedback controllers.

Chebyshev filter: Unlike the monotonic behavior of the Butterworth filter, the Chebyshev filter allows some ripple in the magnitude plot for either the passband or the stopband. The Type 1 Chebyshev filter allows for ripple in the passband while the Type 2 Chebyshev filter allows for ripple in the stopband. Allowing for a ripple results in the Chebyshev filters having sharper transitions near the bandwidth than are achievable by a Butterworth filter of the same order. In Chebyshev design, the cutoff frequency ω_c is usually specified as opposed to the bandwidth. The cutoff frequency is the frequency at which the magnitude of the filter decays to a preset ratio of the DC value. When this ratio is 0.707, the cutoff frequency is the bandwidth. Often, in Chebyshev design, this ratio is chosen to correspond to the amount of ripple allowed in the passband. A Type 1 Chebyshev lowpass filter is defined by the relationships:

$$|H(\omega)| = \frac{1}{\sqrt{1 + \varepsilon^2 C_N^2(\omega/\omega_c)}}$$

and

$$C_N(x) = 2xC_{N-1}(x) - C_{N-2}(x)$$

The $C_N(x)$ expression is called the Nth order Chebyshev polynomial, and it is calculated recursively starting with $C_0(x) = 1$ and $C_1(x) = x$. The value of $\varepsilon > 0$ determines the amount of ripple allowed in the passband; in particular, the ripple exists between the values of 1 and $1/\sqrt{1+\varepsilon^2}$. Consider, for example, the Type 1 Chebyshev filters shown in Figure 9.7; these filters were designed to have 1 dB of ripple in the passband ($\varepsilon = 0.51$). Note that $\varepsilon = 1$ for 3 dB of ripple.

As mentioned above, these prototype filters are lowpass. To design another type of filter, first the lowpass filter is designed, $H(s)$, with a cutoff frequency ω_c (typically chosen to be 1). Then a frequency transformation is used to convert the filter to the desired type. The standard frequency transformations are given below.

Lowpass to lowpass: To obtain a lowpass filter with cutoff frequency ω_1, replace s in the original $H(s)$ by $s\omega_c/\omega_1$.

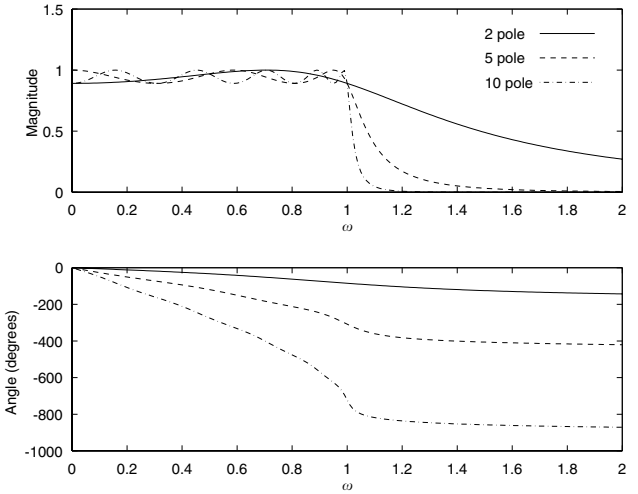


FIGURE 9.7 Comparison of analog Type I Chebyshev filters.

Lowpass to highpass: To obtain a highpass filter with a passband running from ω_1 to ∞, replace s in the original $H(s)$ by $\omega_1\omega_c/s$.

Lowpass to bandpass: To obtain a bandpass filter with a passband running from ω_1 to ω_2, replace s in the original $H(s)$ by

$$\frac{s^2 + \omega_2\omega_1}{s(\omega_2 - \omega_1)}$$

Lowpass to bandstop: To obtain a bandstop filter with stopband running from ω_1 to ω_2, replace s in the original $H(s)$ by

$$\frac{s(\omega_2 - \omega_1)}{s^2 + \omega_2\omega_1}$$

9.4.2 FIR Filter Design

One way to obtain an FIR filter is to truncate the impulse response of an ideal IIR filter. For example, an ideal IIR lowpass filter has the frequency response:

$$H(\Omega) = \begin{cases} A, & -\Omega_c \le \Omega \le \Omega_c \\ 0, & \text{otherwise} \end{cases}$$

where A is a constant and Ω_c is the cutoff frequency. The impulse response of this filter is found from taking the inverse DTFT using Equation 9.3:

$$h[n] = \frac{A\Omega_c}{\pi} e^{jn\Omega_c/2} \operatorname{sinc}\left(\frac{\Omega_c n}{\pi}\right)$$

Notice that this has infinite duration for both $n < 0$ and $n > 0$. Creating an FIR filter would entail truncating the impulse response for $n < -N$ and for $n > N$. However, the original IIR filter and the resulting truncated FIR filter are both noncausal; that is, the impulse response is nonzero for $n < 0$.

Noncausal filters need future values of the input in order to calculate the present value of the output; hence, they cannot be implemented in real-time. For this reason, typical IIR design uses nonideal filters (often based on analog prototypes) that approximate the ideal frequency response. When filtering a signal off-line that has been stored, causality is no longer required since all of the values of the signal are available (including "future" values).

In order to perform real-time implementation of an FIR filter that was generated by truncating an ideal IIR filter, the filter must be delayed so that all of the significant information of the impulse response occurs for $n \geq 0$. This delay in the time domain is equivalent to a linear phase lag in the frequency domain.

Thus, an FIR filter can be designed by first selecting an ideal IIR filter (lowpass, highpass, etc.), then taking the inverse DTFT to find the impulse response, and then truncating the impulse response, and finally, delaying it in time. An equivalent and more preferred method is to rearrange the steps described above. First, add a phase lag in the frequency response of the ideal IIR filter. This is done by multiplying the frequency response by $e^{j(N-1)/2}$. Then, take the inverse DTFT and truncate it for $n < 0$ and $n > N-1$. The result is a causal FIR filter with order N.

The following are generic FIR filters of order N that have been generated using the method described above. Let $m = (N-1)/2$.

Lowpass FIR filter with cutoff frequency Ω_c:

$$h[n] = \begin{cases} \dfrac{\Omega_c}{\pi}, & n = 0 \\ \dfrac{\Omega_c}{\pi}\operatorname{sinc}\left[\dfrac{\Omega_c(n-m)}{\pi}\right], & \text{for } 0 < n \leq N-1 \\ 0, & \text{otherwise} \end{cases}$$

where $\operatorname{sinc}(x) = \sin(\pi x)/\pi x$.

Highpass FIR filter with passband from Ω_1:

$$h[n] = \begin{cases} 1 - \dfrac{\Omega_1}{\pi}, & \text{for } n = 0 \\ -\dfrac{\Omega_1}{\pi}\operatorname{sinc}\left[\dfrac{\Omega_1(n-m)}{\pi}\right]0, & \text{for } 0 < n \leq N-1 \\ 0, & \text{otherwise} \end{cases}$$

Bandpass FIR filter with passband from Ω_1 *to* Ω_2:

$$h[n] = \begin{cases} \dfrac{\Omega_2 - \Omega_1}{\pi}, & \text{for } N = 0 \\ \dfrac{\Omega_2}{\pi}\operatorname{sinc}[\Omega_2(n-m)/\pi] - \dfrac{\Omega_1}{\pi}\sin[\Omega_1(n-m)/\pi], & \text{for } 0 < n \leq N-1 \\ 0, & \text{otherwise} \end{cases}$$

To implement these filters, the coefficients in Equation 9.5 are set to $b_m = h[m-1]$, $a_1 = 1$, and $a_m = 0$ for $m > 1$.

The FIR filters designed using this method have frequency responses that have rather sharp transitions between the passband and the stopband (the larger the order, the sharper the transition), but they tend to give rise to a ripple in the passband and stopband. This ripple results from the sharp truncation of the IIR filter's impulse response. A more gradual truncation using a window can be performed that smooths the ripple in the frequency response. The windows discussed in Section 9.3.2 that are employed in data

collection are also used in FIR filter design, where the modified filter is given as $h[n]w[n]$. FIR design using different windows is discussed in further detail in [1,2].

9.4.3 Computer-Aided Design of Digital Filters

MATLAB™ is a common software package for signal processing analysis and design. The signal processing toolbox contains several commands for designing and simulating digital filters. For example, the commands butter and cheby1 automatically design a prototype analog filter for an IIR and then use the bilinear transformation to map the filter to the discrete-time domain. Lowpass, highpass, bandstop, and bandpass filters can be designed using these commands as long as the digital cutoff frequencies, normalized by π, are specified. To design a digital lowpass filter based on the analog Butterworth filter with cutoff frequency $w1$, use the command [b, a] = butter(N, $w1 * T$/pi) where N is the number of poles, T is the sampling period, and $w1 * T$ is digital cutoff frequency. This command puts the coefficients of the filter, defined in Equation 9.5, in vectors b and a in ascending order. To design a digital highpass filter with analog cutoff frequency $w1$, use the commands [b, a] = butter(N, $w1 * T$/pi, 'high'). To design a digital bandpass filter with analog passband from $w1$ to $w2$, define w = [$w1$, $w2$] and use the command [b, a] = butter (N, $w * T$/pi). To design a digital bandstop filter with stopband from $w1$ to $w2$, define w = [$w1$, $w2$] and use the command [b, a] = butter(N, $w * T$/pi, 'stop'). The design for an Nth order Type I Chebyshev filter is accomplished using the same methods as for butter except that "butter" is replaced by "cheby1."

The signal processing toolbox also provides commands for designing FIR filters. To obtain a lowpass FIR filter with length N and analog cutoff frequency $w1$, use the command h = fir1($N - 1$, $w1 * T$/pi). The resulting vector h contains the impulse response of the FIR where $h(1)$ is the value of $h[0]$. The values in the vector h also equal the coefficients of b in Equation 9.5 in ascending order. (Recall, that $a_1 = 1$ and $a_m = 0$ for $m > 1$.) A length N highpass FIR filter with analog cutoff frequency $w1$ is designed by using the command h = fir1($N - 1$, $w1 * T$/pi, 'high'). A bandpass FIR filter with passband from $w1$ to $w2$ is obtained by typing h = fir1($N - 1$, $w * T$/pi) where w = [$w1$, $w2$]. A bandstop FIR filter with stopband from $w1$ to $w2$ is obtained by typing h = fir1($N - 1$, $w * T$/pi, 'stop') where w = [$w1$, $w2$]. The fir1 command uses the Hamming window by default. Other windows are obtained by adding an option of "hanning" or "boxcar" (which is the rectangular window) to the arguments; for example, h = fir1($N - 1$, $w1 * T$/ pi, 'high,' boxcar(N)) creates a highpass FIR filter with analog cutoff frequency $w1$ using a rectangular window.

The filter command in MATLAB is used to compute an output of a digital filter given its input sequence. An example of its use is y = filter(b, a, x) where b and a are the coefficients of the filter and x is the input sequence.

9.4.4 Filtering Examples

Quite often, 60 Hz noise is encountered in measurements of electromechanical systems due to standard line voltage. (Note, in Europe noise at a 50-Hz frequency is typically encountered.) For demonstration purposes, a 60-Hz signal is superimposed on a lower frequency signal shown in Figure 9.8. To alleviate the detrimental effects of the 60-Hz noise, a bandstop filter may be employed. Typically, most systems

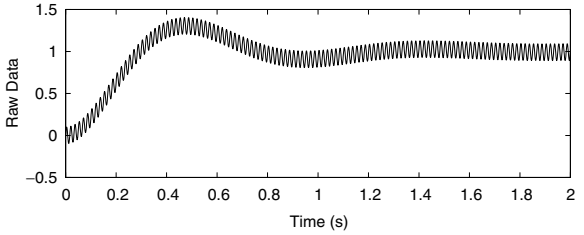


FIGURE 9.8 Measurement corrupted with 60-Hz noise.

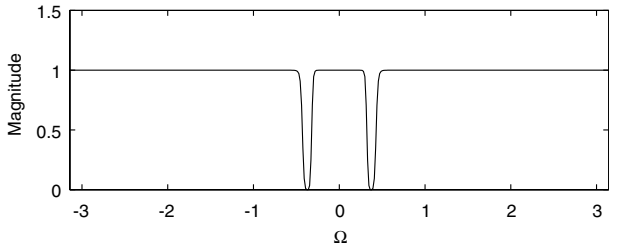


FIGURE 9.9 Bandstop filter.

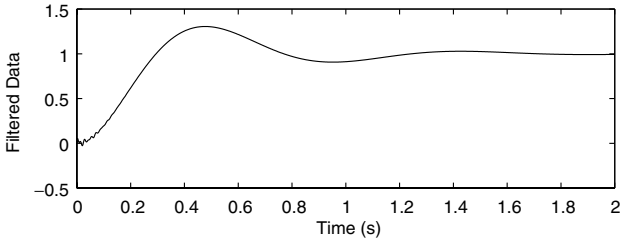


FIGURE 9.10 Filtered measurement.

have a bandstop filter designed around 60 Hz to avoid the type of response seen in Figure 9.8. The following MATLAB commands can be employed to design an eighth-order digital Butterworth bandstop filter whose break frequencies are 50 and 70 Hz. Thus, the filter should reject the 60-Hz noise.

```
T = 0.001; %Sample period
n = 4; % half the order of filter
low_freq = 50 * (2*pi); %Stop signals between 50 and 70 Hz
high_freq = 70 * (2*pi);

w1 = low_freq*(T/pi); % normalized digital break frequencies
w2 = high_freq*(T/pi);
w = [w1 w2];

[b,a] = butter(n,w,'stop'); % filter coefficients

W = -pi:pi/200:pi; % define a digital frequency vector
H = freqz(b,a,W); % computes the frequency response for plotting
```

Figure 9.9 shows the magnitude of the frequency response for the resulting IIR filter. Note that the frequency variable is plotted for the range $[-\pi, \pi]$ where DC frequency corresponds to $\Omega = 0$ and the highest frequency allowable is $\Omega = \pi$. In this example, the digital break frequencies correspond to $\Omega_1 = 50(2\pi)T = 0.314$ and $\Omega_2 = 70(2\pi)T = 0.44$. Figure 9.10 shows the result of applying this filter to the noisy signal. For all practical purposes, the 60-Hz noise is completely attenuated. As can be seen in Figure 9.10, there are some initial system transients during the first 100 ms of the step response. This is a combination of the fourth-order Butterworth filter and the initial system transients to the 60-Hz signal. It should be noted that the sample frequency of 1000 kHz is fast enough to accurately capture the 60-Hz signal. If a sample frequency of less than 120 Hz is used, the 60-Hz signal will be aliased, and no amount of digital filtering would be able to eliminate the effects of the 60-Hz disturbance.

Another application of digital filtering in mechatronics is used when estimating displacement from an acceleration measurement. A simplistic approach to calculating the displacement is to integrate the acceleration twice. In the *s*-domain, this double integration is equivalent to multiplying by $1/s^2$. Using the

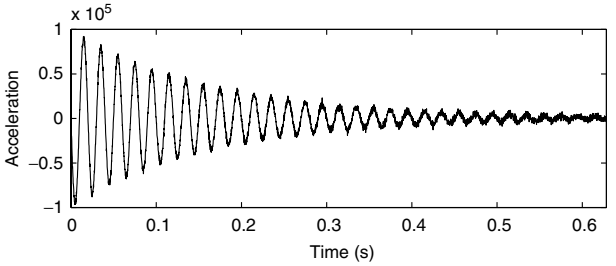


FIGURE 9.11 Acceleration measurement.

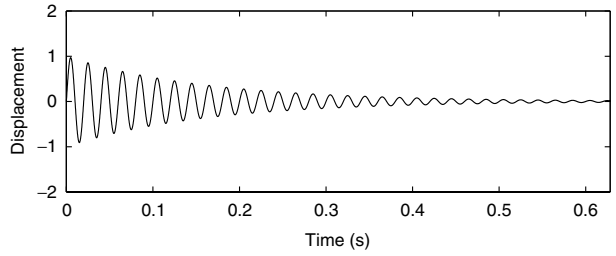


FIGURE 9.12 Actual displacement.

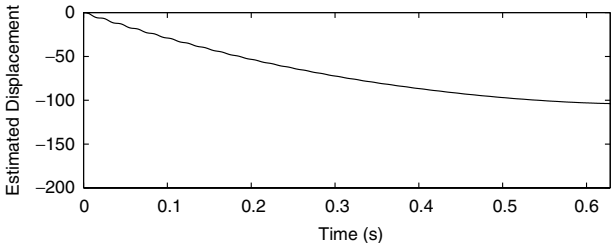


FIGURE 9.13 Estimated displacement without use of a prefilter.

bilinear transformation to convert $1/s^2$ to the z-domain yields the following transfer function,

$$H(z) = H(s)\Big|_{s=\frac{2(z-1)}{T(z+1)}} = \frac{1}{s^2}\Bigg|_{s=\frac{2(z-1)}{T(z+1)}} = \left(\frac{T^2}{4}\right)\left(\frac{z^2+2z+1}{z^2-2z+1}\right) = \left(\frac{T^2}{4}\right)\left(\frac{1+2z^{-1}+z^{-2}}{1-2z^{-1}+z^{-2}}\right)$$

The corresponding difference equation used to calculate the displacement $y[\bullet]$ from the acceleration $y_{dd}[\bullet]$ is $4(y[n] - 2y[n-1] + y[n-2]) = T^2(y_{dd}[n] + 2y_{dd}[n-1] + y_{dd}[n-2])$. However, accelerometers generally do not have good response at low frequencies; in fact, they often insert a bias in the data yielding a drift in the calculated displacement. They also are very sensitive to random vibrations. An alternate approach is to process the acceleration data through a bandpass filter before using the difference equation to integrate it numerically. The bandpass range must contain the natural frequencies in the system.

Consider, for example, the acceleration data shown in Figure 9.11, which has some random noise. This signal is sampled at a rate of 6400 Hz, where the natural frequency of the system is 50 Hz. Figure 9.12 shows the actual displacement, while Figure 9.13 shows the estimated displacement calculated by numerically integrating the acceleration data, using the difference equation given above. This estimation is very poor. Alternatively, an analog eighth-order Chebyshev Type I bandpass filter with passband 25–500 Hz

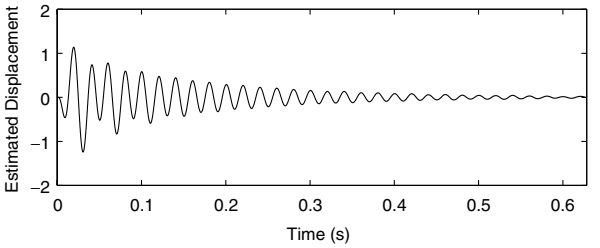


FIGURE 9.14 Estimated displacement with IIR prefilter.

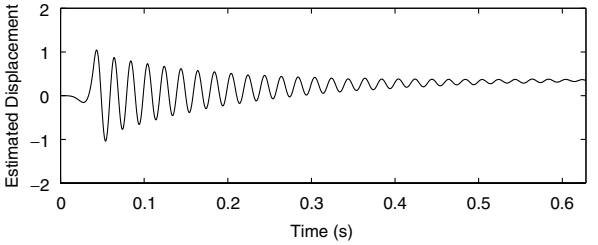


FIGURE 9.15 Estimated displacement with FIR prefilter.

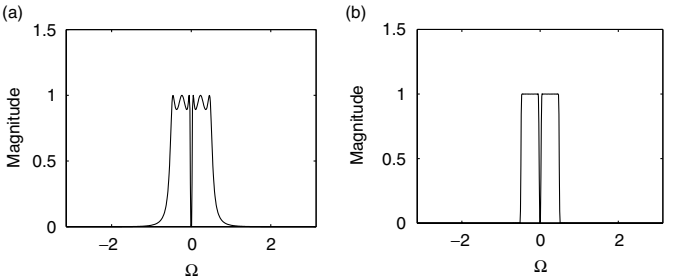


FIGURE 9.16 Digital bandpass filters: (a) Chebyshev IIR and (b) FIR filter.

is designed and then discretized using the bilinear transformation. The acceleration data is processed through this filter first, and then the filtered data are numerically integrated with the result shown in Figure 9.14. Notice that the estimate is much better than that obtained without the bandpass filter. A 500th order FIR bandpass is also designed for this example with passband 25–500 Hz. After passing the data through the FIR filter, it is then numerically integrated resulting in the estimated displacement shown in Figure 9.15. Due to the linear phase characteristic of the FIR filter, it has less transient distortion than the IIR filter, but it adds a larger lag. The larger the order, the more accurate the result, since less significant information is lost in truncating the impulse response of the ideal IIR bandpass filter, but the lag is larger. The magnitudes of the IIR filter and the FIR bandpass filters are shown in Figure 9.16.

Two observations on this example should be mentioned:

1. The unfiltered calculation was extremely sensitive to bias in the data (as expected from the double integration). Therefore, the bias was removed from the acceleration before processing. Both filters effectively removed bias, so the results were virtually unchanged if the bias was present.
2. The FIR filter shows some drift. Presumably, the cause of the drift is that the filter seems to have some difficulty with the small stopband region near the origin. Increasing the stopband region

does reduce the drift. This can be done by decreasing the sample frequency or by increasing the passband frequency. Both of these remedies decrease the drift but increase other errors in the signal. Increasing the length of the filter decreases the drift error without introducing other errors.

Some of the MATLAB commands used to design the filters and generate the results are:

```
[num,den] = c2dm(1,[1 0 0],T,'tustin'); %digitize 1/s^2
y1 = filter(num,den,ydd); % double integration of ydd

Wbreak = [2*pi*25*T, 2*pi*500*T}; % digital break frequencies
[b,a] = cheby1(4,1,Wbreak); % design IIR filter with 1dB ripple

W = -pi:pi/200:pi; % define digital frequency range for plot
H = freqz(b,a,W); % get frequency response
plot(W,abs(H)); % plot magnitude of frequency response

yddfilt = filter(b,a,ydd); % calculate output of IIR filter
y2 = filter(num,den,yddfilt); % double integration of yddfilt

hfir = fir1(500,Wbreak); % design FIR filter of order 500
yddfilt = filter(hfir,1,ydd); % calculate output of FIR filter
y3 = filter(num,den,yddfilt); % double integration of yddfilt
```

9.5 Digital Control Design

As in the digital filter design case, there are two general methods for designing a digital controller: an *indirect method* that is based on discretizing an analog design, and a *direct method* that is based on discretizing a plant (usually using the step-response matching method) and then designing the controller directly in the discrete domain. Most engineers learn classical continuous-time controls, and it is common for them to have more training in continuous-time control design than in discrete-time or digital control design. Fortunately, continuous-time control tools can often be used when designing digital control systems. To make use of controllers designed in the continuous-time domain, an *s*-plane to *z*-plane mapping is used. Any of the mappings discussed in this chapter can be used for a variety of controllers. It is always best to determine the mapping that is most efficient for a particular control or filter. Even though the bilinear approximation is more complex than the forward or backward approximations, it is used for most mechatronic systems. This is due to the fact that most modern controllers have enough computational power to manage the increased complexity at the required bandwidth of the mechatronic system.

As an example of the indirect design method, consider a PD (proportional derivative) controller that may be used to enhance the performance of a system. The derivative and proportional gains for the controller are K_d and K_p, respectively. The PD controller, $K(s)$, is given by

$$K(s) = K_d s + K_p. \tag{9.6}$$

Equation 9.6 can be implemented digitally using any of the *s*-plane to *z*-plane mappings discussed earlier in this chapter. As an example, the bilinear transformation is used generating the digital controller, $K(z)$.

$$K(z) = K(s)\Big|_{s=\frac{2(z-1)}{T(z+1)}} = \frac{(2K_d + TK_p)z + (TK_p - 2K_d)}{Tz + T} \tag{9.7}$$

Besides the control gains, the only factor that is needed for Equation 9.7 is T, the sample time. As previously stated, the sample time should be at least a factor of 5–10 times the fastest system time constant.

However, sampling times are often chosen to be several hundred times faster than the fastest system time constant. An alternative strategy for a feedback system is to choose the sample rate to be at least 20 times the desired closed loop bandwidth. Having sampling times that are substantially faster than the actual system mitigates any differences between the controller as it is designed in the continuous domain and the implementation in the discrete domain. It should be noted that as the sampling frequency becomes higher, the control gains become smaller. For example, in Equation 9.7, as the sampling time becomes smaller, T becomes smaller requiring better numerical resolution for the controller gains. If T becomes smaller than the controller's numerical gain resolution, it may be erroneously implemented at a value of 0 (zero) yielding an incorrect control law.

9.5.1 Digital Control Example

Consider a high speed position motor with motor dynamics governed by the first-order equation

$$G(s) = \frac{\omega(s)}{V_{\text{in}}(s)} = \frac{K_m}{T_m s + 1}$$

where K_m is the motor gain constant, T_m is the motor time constant, $\omega(s)$ is the Laplace transform of the motor velocity, and $V_{\text{in}}(s)$ is the Laplace transform of the motor input voltage. To determine the values of T_m and K_m, the velocity step response of the motor is used. Figure 9.17 is the response of the motor to a 1-V step input. The motor gain, K_m, is the steady-state value of the final motor speed and is 5. This result can also be determined using the Final Value Theorem as

$$\begin{aligned}\lim_{t\to\infty}\omega(t) &= \lim_{s\to 0} s\omega(s) = \lim_{s\to 0} sG(s)V_{\text{in}}(s)\\ &= \lim_{s\to 0} sG(s)\frac{1}{s} = \lim_{s\to 0} s\frac{K_m}{T_m s+1}\frac{1}{s} = K_m\end{aligned}$$

The motor time constant, T_m, can be computed by determining the motor velocity for the step response at time $t = T_m$ as follows:

$$\omega(t = T_m) = K_m(1 - e^{-t/T_m}) = K_m(1 - e^{-1}) = 0.632K_m$$

So the time required for the motor to reach 63.2% of its steady-state step response is its time constant. From Figure 9.18, the time constant of this motor is 0.05 s. Thus, the transfer function for the motor is given by

$$G(s) = \frac{\omega(s)}{V_{\text{in}}(s)} = \frac{K_m}{T_m s + 1} = \frac{5}{0.05s + 1} \tag{9.8}$$

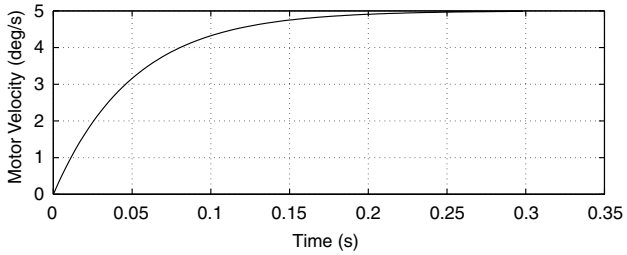


FIGURE 9.17 Motor velocity step response.

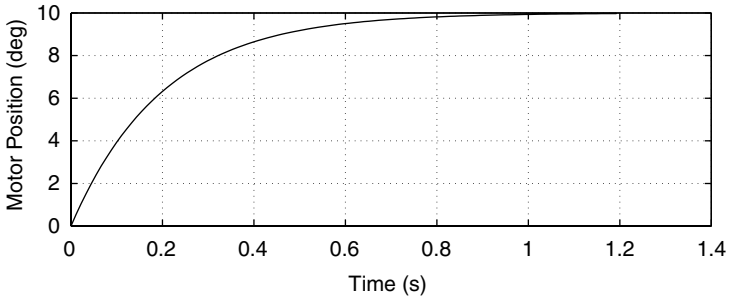


FIGURE 9.18 Closed-loop position response.

For this example, the motor is used in position control mode. Since the motor position is the integral of its velocity, Equation 9.8 can be augmented with an integrator to generate the transfer function of the motor relating the input voltage to the output position, $\theta(s)$:

$$G_p(s) = \frac{\theta(s)}{V_{\text{in}}(s)} = \frac{K_m}{s(T_m s + 1)} \tag{9.9}$$

A PD controller is chosen for use in this example in order to enhance the system performance. To achieve a fast response with no overshoot, the derivative gain, K_d, and the proportional gain, K_p, are chosen to be 0.05 and 1, respectively, yielding the following control law:

$$K(s) = K_d s + K_p = 0.05 s + 1 \tag{9.10}$$

Nominally, this design cancels the high frequency pole of the motor dynamics given in Equation 9.9.

A sample period of 1 ms is chosen for this example as it is significantly faster than the system's time constants, and it is not an unreasonable value given modern digital controllers. As previously discussed, using a 1-kHz (1 ms) sample frequency mitigates any differences between the controller as it is designed in the continuous domain and its implementation is in the discrete domain. Using the bilinear transformation given in Section 9.3.2 results in a digital controller of the form:

$$K_D(z) = \frac{101 z - 99}{z + 1}$$

In fact, the closed-loop response of the system using the digital controller cannot be easily distinguished from that of the system using the analog controller given by Equation 9.10. The closed-loop position response of the motor for a 10° command input is shown in Figure 9.18.

As mentioned in Section 9.4.4, 60 Hz noise is often present in measurements of electro-mechanical systems, so a bandstop filter is often used to attenuate the noise. In the closed-loop operation, the digital bandpass filter is cascaded with the digital PD controller.

References

1. Kamen, E.W., and Heck, B.S., *Signals and Systems Using the Web and Matlab,* 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 2000.
2. Britton Rorabaugh, C., *Digital Filter Designer's Handbook: with C++ Algorithms,* 2nd ed., McGraw-Hill, New York, 1997.

10

Control System Design Via $\mathcal{H}^2$ Optimization

Armando A. Rodriguez
Arizona State University

10.1 Introduction

This chapter addresses control system design via $\mathcal{H}^2$ (quadratic) optimization. A unifying framework based on the concept of a generalized plant and weighted optimization permits designers to address state feedback, state estimation, dynamic output feedback, and more general structures in a similar fashion. The framework permits one to easily incorporate design parameters and/or weighting functions that may be used to influence the outcome of the optimization, satisfy desired design specifications, and systematize the design process. Optimal solutions are obtained via well-known Riccati equations; for example, Control Algebraic Riccati Equation (CARE) and Filter Algebraic Riccati Equation (FARE). While dynamic weighting functions increase the dimension of the Riccati equations being solved, solutions are readily obtained using today's computer-aided design software (e.g., MATLAB, robust control toolbox, μ-synthesis toolbox, etc.).

In short, $\mathcal{H}^2$ optimization generalizes all of the well-known quadratic control and filter design methodologies:

- Linear Quadratic Regulator (LQR) design methodology [7,11]
- Kalman–Bucy Filter (KBF) design methodology [5,6]
- Linear Quadratic Gaussian (LQG) design methodology [4,10,11]

$\mathcal{H}^2$ optimization may be used to systematically design constant gain state feedback control laws, state estimators, dynamic output controllers, and much more.

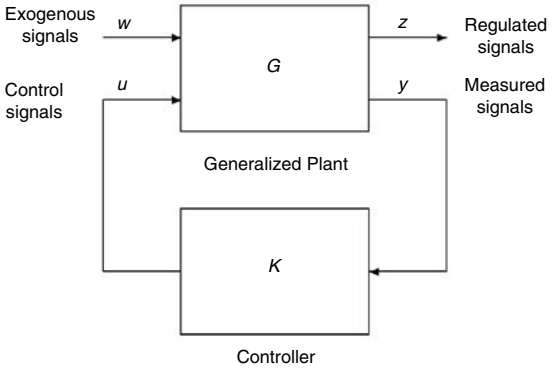


FIGURE 10.1 Generalized feedback system.

10.2 General Control System Design Framework

In this section, we present a general framework for control system (and estimator) design. Toward this end, we consider the generalized feedback system in Figure 10.1. In this figure, G represents a *generalized plant*. G contains a model for the actual plant P (physical system) to be controlled. It may also contain additional (frequency dependent) weighting functions that are used to address closed loop design objectives. K represents a controller or compensator to be designed.

10.2.1 Central Idea: Design Via Optimization

The central idea here is that many important problems that arise in controls, estimation, filtering, and other areas of engineering may be cast in terms of a generalized plant G and a controller K to be designed by minimizing some norm (e.g., $\mathcal{H}^2$) on the closed loop transfer function matrix T_{wz} from the signals w to the signals z.

10.2.2 The Signals

To appreciate the flexibility of our generalized feedback system structure, it suffices to consider the nature of the signals z, u, w, and y in the figure. These signals may be described as follows:

- *Regulated Signals*. The signals $z \in \mathrm{R}^{n_z}$ represent *regulated signals* or signals that we would like to keep "small" in some sense, which depends on the application and desired performance objectives. Such signals might include tracking errors, actuator or control inputs, signal estimation errors, etc.
- *Control Signals*. The signals $u \in \mathrm{R}^{n_u}$ represent *control signals* or *manipulated variables* that are generated by the controller K. Control signals might include fuel flow to an engine, voltage applied to a dc motor, etc. They might also include state estimates provided by K. The idea is for K to manipulate and coordinate control signals u in a manner which keeps the regulated signals z "small." In practice, we typically have more signals that require "regulation" than controls (i.e., $n_z \geq n_u$). It should be noted, however, that generally if we want to independently control m quantities, then we need at least m independent controls. This basic tenet must be adhered to in practice. The more independent controls u that are available, the easier (in principle) it is to influence the signals z to be regulated.
- *Exogenous Signals*. The signals $w \in \mathrm{R}^{n_w}$ represent *exogenous* (*or external*) *signals* that act upon the system. Exogenous signals may include reference commands issued to the control system, disturbances acting on the system, sensor noise, etc.

- *Measurement Signals.* The signals $y \in \mathbf{R}^{n_y}$ represent measurements or signals that are directly available to the controller K. Measurements may include a portion of or all of the plant state variables, measurable plant "outputs," measurable control signals, measurable exogenous signals, etc. In practice, we typically have more exogenous signals than measurements (i.e., $n_w \geq n_y$). Generally, the more independent measurements we have the better—since, in theory, more useful information can be extracted.

Comment 10.1 (Toward a Separation Principle)

It is natural to associate the controls u with the regulated signals z. One might argue that the pair implicitly defines a regulation or control problem. This is analogous to the situation addressed in classical LQR problems. In such problems, one trades off control action (size) versus speed of regulation.

Similarly, it is natural to associate the exogenous signals w with the measurements y. One might argue that the pair implicitly defines an information extraction or estimation problem. This is analogous to the situation addressed in classical KBF problems. In such problems, one trades off sensor cost (or immunity to noise) versus speed of estimate construction.

Such associations suggest that just as in classical LQG problems, our surprisingly general structure may give rise to a natural separation principle. Indeed, this will be the case for the so-called $\mathcal{H}^2$ output feedback problem that we consider. ■

10.2.3 General $\mathcal{H}^2$ Optimization Problem

The so-called general $\mathcal{H}^2$ optimization problem may be stated as follows:

- 'Find a proper real-rational (finite dimensional) controller K that internally stabilizes G such that the $\mathcal{H}^2$ norm of the closed loop system transfer function matrix $T_{wz}(K)$ is minimized:

$$\min_K \| T_{wz}(K) \|_{\mathrm{H}^2} \tag{10.1}$$

where

$$\| F \|_{\mathrm{H}^2} \stackrel{\text{def}}{=} \sqrt{\frac{1}{2\pi} \int_{-\infty}^{\infty} trace\{ F^H(j\omega) F(j\omega) \}\, d\omega} \tag{10.2}$$

$$= \sqrt{\int_0^{\infty} trace\{ f^H(t) f(t) \}\, dt} \tag{10.3}$$

$$= \| f \|_{\mathrm{L}^2(\mathrm{R}_+)} \tag{10.4}$$

and f is the impulse response matrix associated with the transfer function matrix F.

Comment 10.2 (Use of Two Norm: Wide Band Exogenous Signals)

Noting that the two norm measures the energy of the response to an impulse and noting that the transform of unit Dirac delta function δ is unity, it follows that the two norm is appropriate when the exogenous signals w are wide band in nature. This can always be justified by introducing appropriate (low pass) filters within G. It should be noted that these ideas have stochastic interpretations as well. Instead of unit delta functions, one instead deals with white noise with unit intensity. ■

Comment 10.3 (Control and Estimation Problems)

Although we are seeking an $\mathcal{H}^2$ optimal controller, it must be noted that the generalized plant framework will enable the design of state estimators as well as dynamic and constant gain control laws. ■

Given the above problem statement, it is appropriate to recall the following elementary result:

Lemma 10.1 (Two Norm of a Stable System)

Consider a causal stable LTI strictly proper system $F = [A, B, C]$. *It follows that*

$$\|F\|_{H^2} = \|f\|_{L^2(R^+)} = \sqrt{CL_cC^H} = \sqrt{B^HL_oB} \tag{10.5}$$

where L_c *is the system controllability gramian and* L_o *is the system observability gramian. The controllability gramian*

$$L_c \stackrel{\text{def}}{=} \int_0^\infty e^{At}BB^He^{A^Ht}dt \tag{10.6}$$

is the unique symmetric (at least) positive semi-definite solution of the algebraic Lyapunov equation

$$AL_c + L_cA^H + BB^H = 0 \tag{10.7}$$

L_c *is positive definite if and only if* (A, B) *is controllable. The observability gramian*

$$L_o \stackrel{\text{def}}{=} \int_0^\infty e^{A^Ht}C^HCe^{At}dt \tag{10.8}$$

is the unique symmetric (at least) positive semi-definite solution of the algebraic Lyapunov equation

$$A^HL_o + L_oA + C^HC = 0 \tag{10.9}$$

L_o *is positive definite if and only if* (A, C) *is observable.* ■

Comment 10.4 ($\mathcal{H}^2$ Norm May Mislead—L^∞ Norm Is Important)

It is important to note that the $\mathcal{H}^2/\mathcal{L}^2$ norm (or energy) of a function may be very small, while the function itself may be very large in amplitude. Consider a tall thin pulse, for example. This observation is critical because there are many important cases in which we are very concerned with the height of a function—more so than its energy. A good example of this comes from classical Nyquist stability theory [2,8]. Nyquist taught us that the peak magnitude of the sensitivity function $S = 1/(1 + L)$ associated with a standard negative feedback loop is very important in terms of the feedback loop's stability robustness. A large sensitivity means that the Nyquist plot comes close to the critical −1 point—implying that a small perturbation (or unanticipated modeling error) may cause the closed loop system to go unstable. To assist us with this fundamental issue we may use frequency dependent weighting functions, but what we really need is a norm that directly addresses such concerns. This motivates the so-called $\mathcal{H}^\infty$ and $\mathcal{L}^\infty$ norms as well as $\mathcal{H}^\infty/\mathcal{L}^\infty$ control theory [4,11]. ■

Comment 10.5 (Computation of $\mathcal{H}^2$ Norm in MATLAB)

The $\mathcal{H}^2$ norm of a system $F = [A, B, C, D]$ may be computed using the following MATLAB command sequence:

```
lc = lyap (a, b*b')
twonorm = sqrt (trace(c*lc*c'))
```

or

```
lo = lyap (a', c*c')
twonorm = sqrt (trace(b'*lo*b)).
```

■

10.2.4 Generalized Plant

The generalized plant G is assumed to possess the following two-port state space structure:

$$G = \left[\begin{array}{c|cc} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right] = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & 0_{n_z \times n_w} & D_{12} \\ C_2 & D_{21} & 0_{n_y \times n_u} \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{10.10}$$

where $G_{ij}(s) = C_i(sI - A)^{-1}B_j$, $A \in \mathbf{R}^{n \times n}$, $B_1 \in \mathbf{R}^{n \times n_w}$, $B_2 \in \mathbf{R}^{n \times n_u}$, $C_1 \in \mathbf{R}^{n_z \times n}$, $C_2 \in \mathbf{R}^{n_y \times n}$, $D_{12} \in \mathbf{R}^{n_z \times n_u}$, $D_{21} \in \mathbf{R}^{n_u \times n_z}$.

Comment 10.6 (Weighting Functions: Satisfying Closed Loop Design Specifications)

As stated earlier, the generalized plant G may contain frequency dependent weighting functions as well as a model for the physical system P (plant) being controlled. Typically $P = G_{22} = [A, B_2, C_2]$. Weighting functions within G may be viewed as design parameters (mathematical knobs) that may be manipulated by a designer to influence the $\mathcal{H}^2$ problem in a manner which results in a controller that is not just optimal—a notion that is often irrelevant in practical applications—but which satisfies desired closed loop design specifications. Weighting functions may be used to weight (penalize) tracking errors, actuator and other signal levels, state estimation errors, etc. By making the weight on a signal large in a specific frequency range, we are indirectly telling the optimization problem to find a controller that makes the signal small in that frequency range. By making the weight on a signal small in a specific frequency range, we are indirectly conveying our willingness to tolerate a signal which is large in that frequency range. This idea can be illustrated via example. ■

Comment 10.7 ($D_{11} = 0$ Necessary, $D_{22} = 0$ Not Necessary)

$D_{11} = 0$ *Necessary.* Note that we have assumed that $D_{11} = 0$; that is, there is no direct path from the exogenous signals w to the regulated signals z. This assumption is essential for the $\mathcal{H}^2$ norm of the closed loop transfer function T_{wz} to be finite. If $D_{11} \neq 0$, then $\|T_{wz}\|_{\mathrm{H}^2}$ will be infinite and the $\mathcal{H}^2$ problem will be ill-posed; that is, make no sense. If we have a nonzero D_{11}, adding strictly proper filters on either w or z (e.g., $[1000/(s + 1000)]I$) will result in $D_{11} = 0$. In this sense, the assumption is not restrictive.

$D_{22} = 0$ *Not Necessary.* It has also been assumed that $D_{22} = 0$; that is, the transfer function matrix D_{22} from controls u to measurements y is strictly proper. This assumption is very realistic since G_{22} (our plant P) is typically strictly proper in practice. If not, high frequency dynamics (e.g., actuator dynamics, flexible modes, parasitics, etc.) may be included to make it strictly proper. One might even include a simple high bandwidth low pass filter (e.g., $1000/(s + 1000)$) to make G_{22} strictly proper. If this is not desirable because of the increased dimension, there is an alternative that does not increase the dimension of G.

- One can always remove D_{22} from G_{22} to obtain a new generalized plant $\hat{G}$ with $\hat{D}_{22} = 0$. The term D_{22} is then absorbed into an augmented controller K by noting that u is related to y as follows:

$$u = K[y + D_{22}u] = [I - KD_{22}]^{-1}Ky \tag{10.11}$$

  Noting this, it follows that the augmented controller, denoted $\hat{K}$, is given by

$$\hat{K} = [I - KD_{22}]^{-1}K \tag{10.12}$$

  The $\mathcal{H}^2$ problem can then be carried out for $\hat{G}$ and $\hat{K}$ (without regard to D_{22}). When the optimal controller $\hat{K}$ for $\hat{G}$ is obtained, one can compute the optimal controller K for G using the relationship

$$K = \hat{K}[I + D_{22}\hat{K}]^{-1} \tag{10.13}$$

  With this stated, the assumption $D_{22} = 0$ is made without any loss of generality. ■

10.2.5 Closed Loop Transfer Function Matrices

Given the structure for the generalized plant G, we have the following closed loop relationships:

$$u = Ky \tag{10.14}$$

$$= K(G_{21}w + G_{22}u) \tag{10.15}$$

$$= [I - KG_{22}]^{-1}KG_{21}w \tag{10.16}$$

$$= K[I - G_{22}K]^{-1}G_{21}w \tag{10.17}$$

$$y = [I - G_{22}K]^{-1}G_{21}w \tag{10.18}$$

$$z = G_{11}w + G_{12}u \tag{10.19}$$

$$= G_{11}w + G_{12}Ky \tag{10.20}$$

$$= [G_{11} + G_{12}K[I - G_{22}K]^{-1}G_{21}]w \tag{10.21}$$

From this, we have the following closed loop transfer function matrices:

$$T_{wu} = K[I - G_{22}K]^{-1}G_{21} \tag{10.22}$$

$$T_{wy} = [I - G_{22}K]^{-1}G_{21} \tag{10.23}$$

$$T_{wz} = G_{11} + G_{12}K[I - G_{22}K]^{-1}G_{21} \tag{10.24}$$

We say that each of these is a *linear fractional transformation* (*LFT*) involving K.

Comment 10.8 (Well Posedness of Closed Loop System)

In the above manipulation, it has been assumed that the inverse $[I - G_{22}K]^{-1}$ is well defined. This well posedness condition is guaranteed by our assumption that $D_{22} = 0$. This assumption implies that $G_{22}(j\infty) = D_{22} = 0$ and hence that the inverse is well defined. ■

The following example shows how to formulate a so-called Weighted $\mathcal{H}^3$ Mixed Sensitivity Problem to address feedback control system design issues.

Example 10.1 (Weighted $\mathcal{H}^2$ Mixed Sensitivity Problem: Design Philosophy)

This example considers the design of a controller K for a plant $P = [A_p, B_p, C_p, D_p]$ as shown in Figure 10.2. To obtain K, we will formulate an $\mathcal{H}^2$ optimization that considers (directly or indirectly) various issues that are of importance in the design of a good feedback loop.

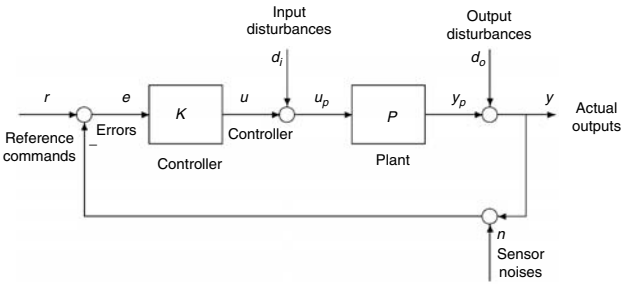


FIGURE 10.2 Standard negative feedback loop.

Feedback System Performance Issues. Generally, in designing a feedback controller K as shown in Figure 10.2, a designer must consider each of the following closed loop performance issues:

1. *Closed Loop Stability.* The closed loop system should be stable. This involves all closed loop transfer function matrices since we generally want all of them to be stable. A stricly proper closed loop transfer function matrix whose $\mathcal{H}^2$ norm is infinite, for example, implies that the transfer function matrix is unstable (or marginally stable). Stable strictly proper transfer function matrices necessarily have a finite $\mathcal{H}^3$ norm.
2. *Command Following.* The closed loop system should exhibit good low frequency reference command following; i.e., the output y (not to be confused with generalized plant measurements) should track low frequency reference commands r that are issued to the feedback system. This typically requires that the sensitivity transfer function matrix

$$S \stackrel{\text{def}}{=} [I + PK]^{-1} \tag{10.25}$$

   be small at low frequencies.
3. *Disturbance Attenuation.* The closed loop system should exhibit good low frequency disturbance attenuation. For disturbances d_o modeled at the plant output, this requires that the sensitivity transfer function matrix be small at low frequencies. For disturbances d_i modeled at the plant input, this requires that

$$T_{d_i y} \stackrel{\text{def}}{=} SP \tag{10.26}$$

   be small at low frequencies.
4. *Sensor Noise Attenuation.* The closed loop system should exhibit good high frequency noise n attenuation. This typically requires that the complementary sensitivity transfer function matrix

$$T \stackrel{\text{def}}{=} I - S \tag{10.27}$$

   be small at high frequencies.
5. *Stability Robustness.* The closed loop system should exhibit robustness with respect to high frequency unmodeled dynamics (e.g., flexible modes, parasitic dynamics, time delays, etc.); This typically requires that the "peak" of some closed loop transfer function matrix be small at high frequencies.
   - *Multiplicative Modeling Error.* For a plant modeled as

$$P_{\text{act}} = [I + \Delta]P \tag{10.28}$$

     where P_{act} represents the actual plant, P represents a nominal model, and Δ represents a stable multiplicative perturbation at the plant output, the relevant closed loop transfer function matrix (that seen by Δ) is T.
   - *Additive Modeling Error.* For a plant modeled as

$$P_{\text{act}} = P + \Delta \tag{10.29}$$

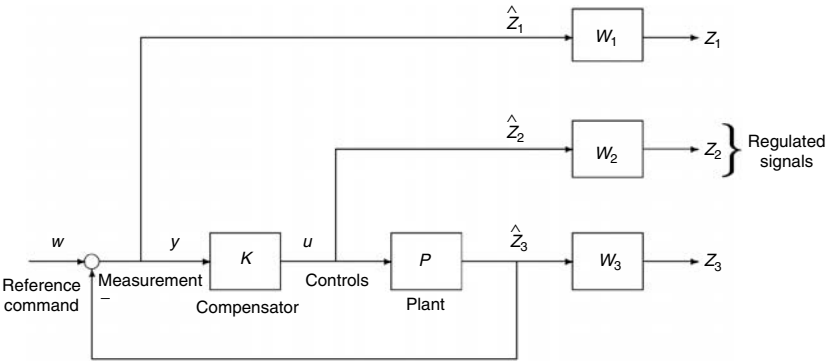


FIGURE 10.3 Negative feedback system for weighted mixed sensitivity problem.

where P_{act} represents the actual plant, P represents a nominal model, and Δ represents a stable additive perturbation, the relevant closed loop transfer function matrix (that seen by Δ) is KS.

- *Reasonable Control Action.* The closed loop system should exhibit reasonably sized control action for typical reference commands and sensor noise. This typically requires that the "size" of KS be controlled. Too much lead (i.e., derivative action) in K may help in terms of stabilization, achieving a high bandwidth and phase margin, but it may result in controls that are unnecessarily large in the presence of typical reference commands r and sensor noise n.

The above list suggests that there are many important issues that impact the control system design process. Part of a designer's job, however, is to prioritize and select issues that are most important. Toward this end, we turn our attention away from Figure 10.2 and consider instead the "fictitious" (mathematical) system depicted in Figure 10.3.

Weighting Functions and Closed Loop Transfer Function Matrix

Figure 10.3 includes specific weighting functions that will help us formulate an $\mathcal{H}^2$ optimization that (directly or indirectly) addresses some of the issues mentioned above. The figure shows a weighting W_1 on the signal y (the tracking error), a weighting W_2 on the controls u, and a weighting W_3 on the plant outputs $\hat{z}_3$. From the figure, it follows that regulated signals

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

are related to the exogenous signals w as follows:

$$z_1 = W_1\hat{z}_1 = W_1 S w \tag{10.30}$$

$$z_2 = W_2\hat{z}_2 = W_2 K S w \tag{10.31}$$

$$z_3 = W_3\hat{z}_3 = W_3 T w \tag{10.32}$$

From this, it follows that the closed loop transfer function matrix from w to z is given by

$$T_{wz} = \begin{bmatrix} W_1 S \\ W_2 K S \\ W_3 T \end{bmatrix} \tag{10.33}$$

Since T_{wz} involves various "sensitivity" transfer function matrices, we say that we have a weighted mixed sensitivity problem.

Selection of Weighting Functions

Typically, the weighting functions W_1, W_2, W_3 are selected to be stable transfer function matrices that are (at least initially) diagonally structured.

- *Sensitivity Weighting*. One might select the sensitivity weighting W_1 on the sensitivity S as follows:

$$W_1 = \left[\frac{k_1}{s+\varepsilon}\right] I_{n_y \times n_y} \tag{10.34}$$

  where k_1, $\varepsilon > 0$. The parameter k_1 is typically selected to be large. The parameter ε is typically selected to be small. Such selections are made so that S is heavily penalized at low frequencies—precisely where we want K to make S small.
- *Control Weighting*. One might select the control weighting W_2 on KS as follows:

$$W_2 = k_2 I_{n_u \times n_u} \tag{10.35}$$

  where $k_2 > 0$ provides a nonsingular penalty on the controls u (i.e., on KS).
- *Output Weighting*. One might select the output weighting W_3 on T as follows:

$$W_3 = \left[\frac{k_3(s+z_3)}{s+p_3}\right] I_{n_y \times n_y} \tag{10.36}$$

  with $k_3 > 0$ and $z_3 < p_3$. Such a weighting would penalize T more heavily at higher frequencies.

In general, care must be taken in selecting the structure of weighting functions. Inappropriate selections may result in an ill-posed problem and a very arduous design process. For example, W_1 must be strictly proper for $\mathcal{H}^2$ problems—otherwise the $\mathcal{H}^2$ norm of $W_1 S$ makes no sense (since S approaches the identity at high frequencies). While there exists no precise systematic method for the selection of weighting functions, the above structures seem to work well (as starting points) in many applications.

Input–Output Representation for Generalized Plant G

To obtain an input–output (transfer function matrix) description for our generalized plant, we must express the regulated signals z_1, z_2, z_3 and the measurements y in terms of the exogenous signals w and the controls u. Doing so yields

$$z_1 = W_1 \hat{z}_1 = W_1(w - Pu) = W_1 w - W_1 P u \tag{10.37}$$

$$z_2 = W_2 \hat{z}_2 = W_2 u \tag{10.38}$$

$$z_3 = W_3 \hat{z}_3 = W_3 P u \tag{10.39}$$

$$y = w - \hat{z}_3 = w - Pu \tag{10.40}$$

From this, we obtain the following input–output (transfer function matrix) description for our generalized plant G:

$$\left[\frac{z}{y}\right] = \left[\begin{array}{c|c} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right] \left[\frac{w}{u}\right] \tag{10.41}$$

or

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \hline y \end{bmatrix} = \left[\begin{array}{c|c} W_1 & -W_1P \\ 0 & W_2 \\ 0 & W_3P \\ \hline I & -P \end{array}\right] \begin{bmatrix} w \\ u \end{bmatrix} \tag{10.42}$$

State Space Representation for Generalized Plant G

Next we obtain a two-port state space representation for G. To do so, we assume the following state space representations:

$$P = [A_p, B_p, C_p, D_p] \quad \text{with state } x_p \tag{10.43}$$

$$W_1 = [A_1, B_1, C_1, D_1] \quad \text{with state } x_1 \tag{10.44}$$

$$W_2 = [A_2, B_2, C_2, D_2] \quad \text{with state } x_2 \tag{10.45}$$

$$W_3 = [A_3, B_3, C_3, D_3] \quad \text{with state } x_3 \tag{10.46}$$

To obtain the desired state space representation for G, we need to express the signals $(\{x_i\}_{i=1}^3, x_p, \{z_i\}_{i=1}^3, y)$ in terms of the signals $(\{x_i\}_{i=1}^3, x_p, w, u)$. This is just a matter of simple bookkeeping. Doing so yields the following:

$$x_1 = A_1x_1 + B_1y = A_1x_1 + B_1(w - C_px_p - D_pu) = A_1x_1 - B_1C_px_p - B_1D_pu \tag{10.47}$$

$$x_2 = A_2x_2 + B_2u \tag{10.48}$$

$$x_3 = A_3x_3 + B_3\hat{z}_3 = A_3x_3 + B_3(C_px_p + D_pu) = A_3x_3 + B_3C_px_p + B_3D_pu \tag{10.49}$$

$$x_p = A_px_p + B_pu \tag{10.50}$$

$$z_1 = C_1x_1 + D_1y = C_1x_1 + D_1(w - C_px_p - D_pu) = C_1x_1 - D_1C_px_p + D_1w - D_1D_pu \tag{10.51}$$

$$z_2 = C_2x_2 + D_2u \tag{10.52}$$

$$z_3 = C_3x_3 + D_3\hat{z}_3 = C_3x_3 + D_3(C_px_p + D_pu) = C_3x_1 + D_3C_px_p + D_3D_pu \tag{10.53}$$

$$y = w - C_px_p - D_pu = -C_px_p + w - D_pu \tag{10.54}$$

The above equations may be written in standard two-port form:

$$\begin{bmatrix} \dot{x} \\ \hline z \\ \hline y \end{bmatrix} = \left[\begin{array}{c|c|c} A & B_{11} & B_{12} \\ \hline C_{11} & D_{11} & D_{12} \\ \hline C_{21} & D_{21} & D_{22} \end{array}\right] \begin{bmatrix} x \\ \hline w \\ \hline u \end{bmatrix} \tag{10.55}$$

as follows:

$$\left[\begin{array}{c} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_p \\ \hline z_1 \\ z_2 \\ z_3 \\ \hline y \end{array}\right] = \left[\begin{array}{cccc|c|c} A_1 & & & -B_1C_p & & -B_1D_p \\ & A_2 & & & & B_2 \\ & & A_3 & B_3C_p & & B_3D_p \\ & & & A_p & & B_p \\ \hline C_1 & & & -D_1C_p & D_1 & -D_1D_p \\ & C_2 & & & & D_2 \\ & & C_3 & D_3C_p & & D_3D_p \\ \hline & & & -C_p & I & -D_p \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_p \\ \hline w \\ \hline u \end{array}\right] \tag{10.56}$$

Checking Assumptions

In selecting the weights, W_1, W_2, W_3, one must make sure that none of the "standard" $\mathcal{H}^2$ problem assumptions are violated. Thus far, we require that $D_{11} = 0$ and $D_{22} = 0$. To ensure that $D_{11} = 0$, we need

$$D_1 = 0 \tag{10.57}$$

To ensure that $D_{22} = 0$, we need

$$D_p = 0. \tag{10.58}$$

This results in

$$\left[\begin{array}{c} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_p \\ \hline z_1 \\ z_2 \\ z_3 \\ \hline y \end{array}\right] = \left[\begin{array}{cccc|c|c} A_1 & & & -B_1C_p & & B_2 \\ & A_2 & & & & \\ & & A_3 & B_3C_p & & \\ & & & A_p & & B_p \\ \hline C_1 & & & & & \\ & C_2 & & & & D_2 \\ & & C_3 & \Delta_3C_p & & \\ \hline & & & -C_p & I & \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_p \\ \hline w \\ \hline u \end{array}\right] \tag{10.59}$$

In subsequent sections, additional assumptions will be imposed on the two-port state space representation for the generalized plant G. The additional assumptions imposed will depend upon the specific $\mathcal{H}^2$ problem being considered.

Weighted $\mathcal{H}^2$ Optimal Mixed Sensitivity Problem

Given the above, the weighted $\mathcal{H}^2$ optimal mixed sensitivity problem is then to find a real-rational (finite-dimensional) proper internally stabilizing controller K that minimizes $\|T_{wz}\|_{\mathrm{H}^2}$; that is,

$$\min_K \|T_{wz}\|_{\mathrm{H}^2} = \min_K \left\| \begin{bmatrix} W_1S \\ W_2KS \\ W_3T \end{bmatrix} \right\|_{\mathrm{H}^2} \tag{10.60}$$

We will show how this optimal control problem—and problems like it—can be readily solved using computer-aided design soflware (e.g., MATLAB, robust control toolbox, μ-tools). ■

Comment 10.9 (Construction of Generalized Plant G)

Generalized plants G are very easy to construct within SIMULINK. Input port blocks may be used to specify exogenous signals w and controls u. Output port blocks may be used to specify regulated signals z and measurements y. The "linmod" command may be applied to the constructed block diagram (SIMULINK file) to obtain a two-port state space (A, $B = [B_1\ B_2]$, $C = [C_1; C_2]$, $D = [D_{11}\ D_{12}; D_{21}\ D_{22}]$) representation for G. The syntax for the command is as follows:

```
[ a, b, c, d ] = linmod ('filename')
```

This method enables one to create generalized plant models quickly. ■

10.2.6 Overview of $\mathcal{H}^2$ Optimization Problems to Be Considered

Three fundamental problems are considered in this chapter:

1. ***$\mathcal{H}^2$ Output Feedback Problem.*** The solution to this problem is an optimal model-based dynamic compensator possessing the structure

$$K_{\text{opt}} = \left[\begin{array}{c|c} A - B_2 G_c - H_f C_2 & H_f \\ \hline -G_c & O_{n_u \times n} \end{array}\right] \tag{10.61}$$

   where G_c, is a control gain (state feedback) matrix and H_f is a filter gain (observer) matrix. G_c is found by using the solution of a Control Algebraic Riccati Equation (CARE)—similar to that found in Linear Quadratic Regulator (LQR) problems. H_f is found by using the solution of a Filter Algebraic Riccati Equation (FARE)—similar to that found in Kalman–Bucy Filtering (KBF) problems. The structure of K_{opt}, G_c, and H_f can be thought of as the solution to a classical Linear Quadratic Gaussian (LQG) control problem which gives rise to the well known separation principle: closed loop poles are the eigenvalues of $A - B_2 G_c$, and the eigenvalues of $A - H_f C_2$.
2. ***$\mathcal{H}^2$ State Feedback Problem.*** The solution to this problem is an optimal constant gain (state feedback) compensator possessing the structure

$$K_{\text{opt}} = -G_c \tag{10.62}$$

   where G_c is a control gain (state feedback) matrix found by using the solution to a CARE—similar to that found in LQR problems. The poles of the resulting closed loop system are the eigenvalues of $A - B_2 G_c$. In short, this problem should be viewed as a mechanism for computing control gain matrices G_c that may be used in a state feedback application or in a model-based compensator application.
3. ***$\mathcal{H}^2$ Output Injection Problem.*** The solution to this problem is an optimal constant gain (static) compensator possessing the structure

$$K_{\text{opt}} = -H_f \tag{10.63}$$

   where H_f is a filter gain (observer) matrix found by using the solution to a FARE—similar to that found in KBF problems. The poles of the resulting closed loop system are the eigenvalues of $A - H_f C_2$. In short, this problem should be viewed as a mechanism for computing filter gain matrices H_f that may be used in a state estimation application or in a model-based compensator application.

10.3 $\mathcal{H}^2$ Output Feedback Problem

In this section, we consider the $\mathcal{H}^2$ output feedback problem. This problem results in model-based (dynamic) compensators involving a control gain (state feedback) matrix G_c and a filter gain (observer) matrix H_f. As such, the problem generalizes the ideas presented in classical LQG theory.

The following "standard" $\mathcal{H}^2$ output feedback problem assumption is now made.

Assumption 10.1 ($\mathcal{H}^2$ Output Feedback Problem)

Throughout this section, it will be assumed that

1. *Plant G_{22} Assumption.* (A, B_2, C_2) stabilizable and detectable.
   - This assumption is necessary and suficient for the existance of a proper internally stabilizing controller K. With this assumption, the following model based (observer based) controller stabilizes the feedback loop in Figure 10.1:

$$K = \left[\begin{array}{c|c} A - B_2 G_c - H_f (C_2 - D_{22} G_c) & H_f \\ \hline -G_c & O_{n_u \times n} \end{array}\right] \tag{10.64}$$

   provided that $(A - B_2 G_c)$ and $(A - H_f C_2)$ are stable, as suggested by the classical separation principle from the theory of linear systems.

   This assumption mandates that *all of the "bad" open loop poles (right half plane and imaginary) must be controllable through the controls u and observable through the measurements y.*

   Suppose that G satisfies the assumption. Consider the augmentation of an integrator $1/s$ (weighting function). Such an augmentation can result in the assumption being violated. Absorbing an integrator $1/s$ (weighting function) on the exogenous signals w into G, for example, would violate the stabilizability assumption since it would introduce an open loop pole on the imaginary axis that is not controllable through the controls u. Absorbing an integrator on the regulated signals z into G would violate the detectability assumption since it would introduce an open loop pole on the imaginary axis that is not observable through the measurements y. Using $1/(s + \delta)(\delta > 0)$ instead of $1/s$—in either case—would result in a G that does not violate the assumption.
2. *Nonsingular Control Weighting Assumption.* $R = D_{12}^T D_{12} > 0$.
   - This assumption implies that $D_{12} \in \mathbf{R}^{n_z \times n_u}$ has full column rank (i.e., rank $D_{12} = n_u$) and hence that every control (direction) u influences the regulated signals z through D_{12} (i.e., D_{12} has no right null space). The matrix D_{12} must therefore be "tall" and "thin;" that is,

$$\text{(number of regulated signals) } n_z \geq n_u \text{ (number of control signals)} \tag{10.65}$$

   The matrix $R = D_{21}^T D_{12} \in \mathbf{R}^{n_u \times n_u}$ may be interpreted as a weighting on the controls u—just like the control weighting matrix "R" in LQR problems. As in the LQR problem, we might say that the control weighting R on u is nonsingular. The larger R, the smaller we want our controls to be—sacrificing speed of regulation. A large R results in a low "regulation" bandwidth. The smaller R, the larger we will permit our controls u to be—in order to speed up regulation. A small R results in a high "regulation" bandwidth.
3. *Regulator Assumption.* $\begin{bmatrix} j\omega I - A & -B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank $(n + n_u)$ for all ω.
   - This assumption implies that transfer function matrix from control signals u to regulated signals z has no (right) zero on the imaginary axis. Together with (1) and (2), it will guarantee that the Hamiltonian $\mathcal{H}_{\text{con}}$ involving (A, B_2, C_1, D_{12}, R)—that is, associated with the controls u and regulated signals z—will belong to $dom(Ric)$. This, in turn, guarantees that the solution of the associated CARE results in a control gain matrix $G_c \in \mathbf{R}^{n_u \times n}$ such that $A - B_2 G_c$ is stable.

The assumption implies that G has no imaginary modes that are unobservable through the regulated signals z; that is, all open loop poles on the imaginary axis must be observable through the regulated signals z. (A,C_1), therefore, cannot possess unobservable imaginary modes. This is a necessary condition. It is not suficient. An integrator hanging on the measurements y, for example, would violate this.

Since D_{12} has full column rank, the assumption is equivalent to the pair

$$(A - B_2R^{-1}D_{12}^TC_1,\ \ (I - D_{12}R^{-1}D_{12}^T)C_1) \tag{10.66}$$

having no unobservable imaginary modes:

- If D_{12} is square, then it is invertible and the assumption is equivalent to $A - B_2R^{-1}D_{12}^TC_1$ having no imaginary modes.
- If $D_{12}^TC_1 = 0$ (no cross penalty between controls and states), then the assumption is equivalent to (A, C_1) having no unobservable imaginary modes.

4. *Nonsingular Measurement Weighting Assumption.* $\Theta = D_{21}D_{12}^T > 0$.
   - This assumption implies that $D_{21} \in \mathbf{R}^{n_y \times n_w}$ has full row rank (i.e., rank $D_{2l} = n_y$) and hence that the measurements y are linearly independent through D_{21} (i.e., D_{21} has no left null space). The matrix D_{21} must therefore be "short" and "fat;" that is,

$$\text{(number of measurements) } n_y \leq n \text{ (number of exogenous signals)} \tag{10.67}$$

   The matrix $\Theta = D_{21}D_{21}^T \in \mathbf{R}^{n_y \times n_y}$ may be interpreted as the intensity of sensor noise impacting the measurements y—just like the sensor noise intensity matrix "Θ" found in KBF problems. As in the KBF problem, we say that the intensity matrix Θ associated with the measurements y is nonsingular. The larger Θ, the more we want to low pass filter the measurements y—sacrificing speed of estimation. A large Θ results in a low bandwidth for the associated estimator (observer). The smaller Θ, the less we want to low pass filter the measurements y—trading off our immunity to noise for speed of estimation. A small Θ results in a high bandwidth for the associated estimator (observer).
5. *Filter Assumption.* $\begin{bmatrix} j\omega I - A & -B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank $(n + n_y)$ for all ω.
   - This assumption implies that transfer function matrix from exogenous signals w to measurements y has no (left) zero on the imaginary axis. Together with (1) and (3), it will guarantee that the Hamiltonian $\mathcal{H}_{\text{fil}}$ involving $(A, B_1, C_2, D_{21}, \Theta)$—that is, involving exogenous signals z and measurements y—will belong to $dom(Ric)$. This, in turn, guarantees that the solution of the associated FARE results in a filter gain matrix $H_f \in \mathbf{R}^{n \times n_y}$ such that $A - H_fC_2$ is stable.

   The assumption implies that G has no imaginary modes that are uncontrollable through the exogenous signals w; that is, all open loop poles on the imaginary axis must be controllable through the exogenous signals w. (A, B_1), therefore, cannot possess uncontrollable imaginary modes. This is a necessary condition. It is not suficient. An integrator hanging on the controls u, for example, would violate this.

   Since D_{21} has full row rank, the assumption is equivalent to the pair

$$(A - B_1D_{21}^T\Theta^{-1}C_2,\ \ \mathrm{B}_1(I - D_{21}^T\Theta^{-1}D_{21})) \tag{10.68}$$

having no uncontrollable imaginary modes.

- If D_{21} is square, then it is invertible and the assumption is equivalent to $A - B_1D_{21}^T\Theta^{-1}C_2$ having no imaginary modes.
- If $B_1D_{21}^T = 0$ (uncorrelated process and sensor noise), then the assumption is equivalent to (A, B_1) having no uncontrollable imaginary modes. ■

Comment 10.10 (Duality Relationships)

In the above discussion, we note the following dual relationships:

$$A \longleftrightarrow A^T \tag{10.69}$$

$$B_2 \longleftrightarrow C_2^T \tag{10.70}$$

$$C_1 \longleftrightarrow B_1^T \tag{10.71}$$

$$D_{12} \longleftrightarrow D_{21}^T \tag{10.72}$$

$$R = D_{12}^T D_{12} \longleftrightarrow \Theta = D_{21} D_{21}^T \tag{10.73}$$

These imply that

- Controls u are dual to measurements y.
- Regulated signals z are dual to exogenous signals w. ■

10.3.1 Hamiltonian Matrices

Associated with our $\mathcal{H}^2$ optimal control problem are the following two Hamiltonian matrices:

$$H_{\text{con}} = \begin{bmatrix} A & 0 \\ -C_1^T C_1 & -A^T \end{bmatrix} - \begin{bmatrix} B_2 \\ -C_1^T D_{12}^T \end{bmatrix} R^{-1} \begin{bmatrix} D_{12}^T C_1 & B_2^T \end{bmatrix} \tag{10.74}$$

$$= \begin{bmatrix} A - B_2 R^{-1} D_{12}^T C_1 & -B_2 R^{-1} B_2^T \\ -C_1^T (I - D_{12}^T R^{-1} D_{12}^T) C_1 & -(A - B_2 R^{-1} D_{12}^T C_1)^T \end{bmatrix} \tag{10.75}$$

$$H_{\text{fil}} = \begin{bmatrix} A^T & 0 \\ -B_1 B_1^T & -A \end{bmatrix} - \begin{bmatrix} C_2^T \\ -B_1 D_{21}^T \end{bmatrix} \Theta^{-1} \begin{bmatrix} D_{21} B_1^T & C_2 \end{bmatrix} \tag{10.76}$$

$$= \begin{bmatrix} (A - B_1 D_{21}^T \Theta^{-1} C_2)^T & -C_2^T \Theta^{-1} C_2 \\ -B_1 (I - D_{21}^T \Theta^{-1} D_{21}) B_1^T & -(A - B_1 D_{21}^T \Theta^{-1} C_2) \end{bmatrix} \tag{10.77}$$

The first Hamiltonian is associated with an optimal state feedback control or regulator problem. The second is associated with an optimal filtering or estimation problem.

The solution to the $\mathcal{H}^2$ output feedback problem is now given [11, pp. 261–262].

Theorem 10.1 (Solution to $\mathcal{H}^2$ Output Feedback Problem Subject to Standard Assumptions)

Suppose that G satisfies the assumptions given in Assumption 10.1—the so-called standard $\mathcal{H}^2$ output feedback problem assumptions. Given this, we have the following.

The unique minimizing $\mathcal{H}^2$ optimal controller is n dimensional (like generalized plant G) and is given by

$$K_{\text{opt}} = \left[\begin{array}{c|c} A - B_2 G_c - H_f C_2 & H_f \\ \hline -G_c & O_{n_u \times n_y} \end{array}\right] \tag{10.78}$$

where the control gain matrix $G_c \in \mathrm{R}^{n_u \times n}$ *is given by*

$$G_c = R^{-1}[B_2^T X + D_{12}^T C_1] \tag{10.79}$$

$X = Ric(\mathcal{H}_{con}) \geq 0$ *is the unique (at least) positive semi-definite solution of the CARE:*

$$(A - B_2 R^{-1} D_{12}^T C_1)^T X + X(A - B_2 R^{-1} D_{12}^T C_1) + C_1^T (1 - D_{12}^T R^{-1} D_{12}^T) C_1 - X B_2 R^{-1} B_2^T X = 0 \tag{10.80}$$

and the filter gain matrix $H_f \in \mathrm{R}^{n \times n_y}$ *is given by*

$$H_f = [Y C_2^T + B_1 D_{21}^T] \Theta^{-1} \tag{10.81}$$

$Y = Ric(\mathcal{H}_{fil}) \geq 0$ *is the unique (at least) positive semi-definite solution of the FARE:*

$$(A - B_1 D_{12}^T \Theta^{-1} C_2) Y + Y(A - B_1 D_{21}^T \Theta^{-1} C_2)^T + B_1 (I - D_{21}^T \Theta^{-1} D_{21}) B_1^T - Y C_2^T \Theta^{-1} C_2 Y = 0 \tag{10.82}$$

Moreover, the minimum norm is given by

$$\left\| T_{wz}(K_{\text{opt}}) \right\|_{L^2} = \sqrt{\left\| M_c B_1 \right\|_{L^2}^2 + \left\| R^{1/2} G_c M_f \right\|_{L^2}^2} \tag{10.83}$$

$$= \sqrt{trace(B_1^T X B_1) + trace(R G_c Y G_c^T)} \tag{10.84}$$

where

$$M_c = [A - B_2 G_c, I_{n \times n}, C_1 - D_{12} G_c] \tag{10.85}$$

$$M_f = [A - H_f C_2, I_{n \times n}, B_1 - H_f D_{21}] \tag{10.86}$$

Finally, the closed loop poles are the eigenvalues of $A - B_2 G_c$ *and* $A - H_f C_2$. ■

Comment 10.11 (Computing Optimal $\mathcal{H}^2$ Controller in MATLAB)

The following MATLAB command sequence may be used to compute the optimal $\mathcal{H}^2$ controller K_{opt} and the resulting closed loop transfer function matrix T_{wz}:

```
tss_g = mksys(a, [b1 b2], [c1; c2], [0*ones(nz, nw) d12; d21
0*ones(ny, nu),'tss')
[ss_k ss_twz] = h21qg(tss_g, 'schur')
[a_k, b_k, d_k] = branch(ss_k, 'a,b,c,d')
```

The "mksys" command packs the two-port state space data for the generalized plant G into a column vector data structure (called a tree) possessing the "tss" (two-port state space) variable designation. All dimension information is encoded into the column vector. The "h21qg" command computes the optimal $\mathcal{H}^2$ controller K_{opt} and the associated closed loop system from the exogenous signals w to the regulated signals z. An eigenvalue-eigenvector method is the default method used to solve the two relevant algebraic Riccati equations. A Schur method—based on Schur's unitary transformation of a matrix to upper triangular form—may be used by including the "schur" option. The results are stored in the tree vectors `ss_k` and `ss_twz`, respectively. The 'branch' command is then used to retrieve the state space representation for K_{opt} from the tree vector `ss_k`. ■

Comment 10.12 (Relationship to LQG, Stability Robustness Margins)

Theorem 10.1 shows that the optimal $\mathcal{H}^2$ output feedback controller is identical in structure to that found in classical LQG problems. While certain LQR, KBF, and LQG/LTR problem formulations do result in feedback loops possessing stability robustness margins, LQG controllers need not possess margins [3].

The same is true for $\mathcal{H}^2$ output feedback designs. We will show how the $\mathcal{H}^2$ framework presented can be manipulated to solve LQG/LTR problems which yield model-based controllers with desirable stability robustness margins—comparable to those found in feedback designs resulting from suitably formulated LQR and KBF problems (e.g., infinite upward gain margin, at least 6 dB downward gain margin, at least ±60° phase margin). ■

The following example shows how weighted $\mathcal{H}^2$ mixed sensitivity optimization may be used to design a controller for an unstable system with a time delay.

Example 10.2 (Weighted H^2 Mixed Sensitivity Design for Unstable System with Time Delay)

In this example, we consider an unstable system with a time delay $\Delta = 0.05$ s (50 ms). The system is modeled (approximately) as follows:

$$P \approx \frac{1}{s-1}\left[\frac{2/\Delta - s}{2/\Delta + s}\right] = \frac{1}{s-1}\left[\frac{40 - s}{40 + s}\right] \tag{10.87}$$

Design Specifications. The objective is to design a controller K that satisfies the following closed loop specifications: (1) closed loop stability, (2) sensitivity below −60 dB for all frequencies below 0.1 rad/s, (3) sensitivity gain crossover between 2 and 3 rad/s, (4) peak sensitivity below 5 dB, (5) peak complementary sensitivity below 10 dB.

Weighted $\mathcal{H}^2$ Mixed Sensitivity Problem. To achieve the above specifications, we formulated a weighted $\mathcal{H}^2$ mixed sensitivity problem—with a weighting W_1 on the sensitivity S and a weighting W_2 on KS; i.e.,

$$\min_K \|T_{wz}\|_{H^2} = \min_K \left\| \begin{bmatrix} W_1 S \\ W_2 K S \end{bmatrix} \right\|_{H^2} \tag{10.88}$$

The weighting functions used were as follows:

$$W_1 = \frac{k_1}{s + p_1} = \frac{10}{s + 0.01} \tag{10.89}$$

$$W_2 = \frac{k_2(s + z_2)}{s + p_2} = \frac{0.1(s + 40)}{s + 2} \tag{10.90}$$

W_1 penalizes the sensitivity S heavily at low frequencies (e.g., below 0.001 rad/s). Above 0.1 rad/s, W_1 is small and W_2 penalizes KS (with magnitude greater than unity) until about 4 rad/s. Since the solution of our $\mathcal{H}^2$ optimization depends in a very complex manner on the parameters that define W_1 and W_2, it should be no surprise that it took a while to determine suitable parameters.

Construction of Generalized Plant. The generalized plant G was assembled using SIMULINK and the "linmod" command. The resulting two-port state space representation is as follows:

$$G = \left[\begin{array}{c|c} 0 & -W_1 P \\ 0 & W_2 \\ \hline 1 & -P \end{array}\right] = \left[\begin{array}{c|c|c} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ \hline C_2 & D_{21} & D_{22} \end{array}\right] = \left[\begin{array}{cccccc} -0.01 & 0 & -40 & 1 & 1 & 0 \\ 0 & -2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 40 & -39 & 0 & 1 \\ \hline 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0.1 \\ \hline 0 & 0 & -40 & 1 & 1 & 0 \end{array}\right] \tag{10.91}$$

Computation of $\mathcal{H}^2$ Optimal Controller. The "mksys" command was used to pack the above two-port state space into a tree vector data structure. The "h2lqg" command was then used to obtain the optimal controller. Note that the generalized plant is 4th order (two for plant P, one for sensitivity weighting W_1, one for control weighting W_2). The optimal controller:

$$K_{\text{opt}} = \frac{191.0813(s+40)(s+2)(s+0.526)}{(s+1.915)(s+0.01)(s^2+84.15s+2133)} \tag{10.92}$$

is also 4th order—the order of the generalized plant G. The pole at $s=-0.01$ is an approximate integrator—a consequence of the heavy weighting that W_1 places on the sensitivity at low frequencies.

Closed Loop Analysis. The resulting closed loop poles (two plant $P = G_{22}$, four from controller K_{opt}) are as follows:

$$s = -1, -2.0786 \pm j0.8302, -40, -40, -39.9216 \tag{10.93}$$

The resulting sensitivity, KS, and complementary sensitivity frequency responses are shown in Figures 10.4–10.6, respectively. The figures show that all of the design specifications are met (or nearly met). The peak sensitivity is about 4.855 dB. The peak complementary sensitivity is about 8.71 dB. The KS response shows the impact of the compensators' lead between 0.1 and 10 rad/s.

Computation of Minimum $\mathcal{H}^2$ Norm. The minimum two-norm was computed using the following MATLAB command sequence:

```
lc = lyap(acl, bcl*bcl)
minnorm = sqrt( trace( cc1*c*c'))
```

The minimum two-norm was found to be 9.0648. ■

The following simple example illustrates how the $\mathcal{H}^2$ output feedback problem solution can be used to solve classical LQG problems.

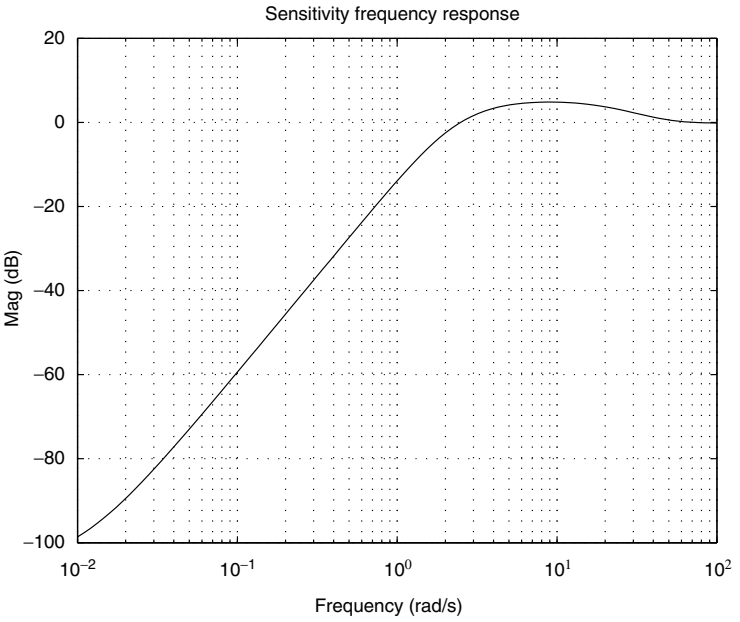


FIGURE 10.4 $\mathcal{H}^2$ Design sensitivity frequency response.

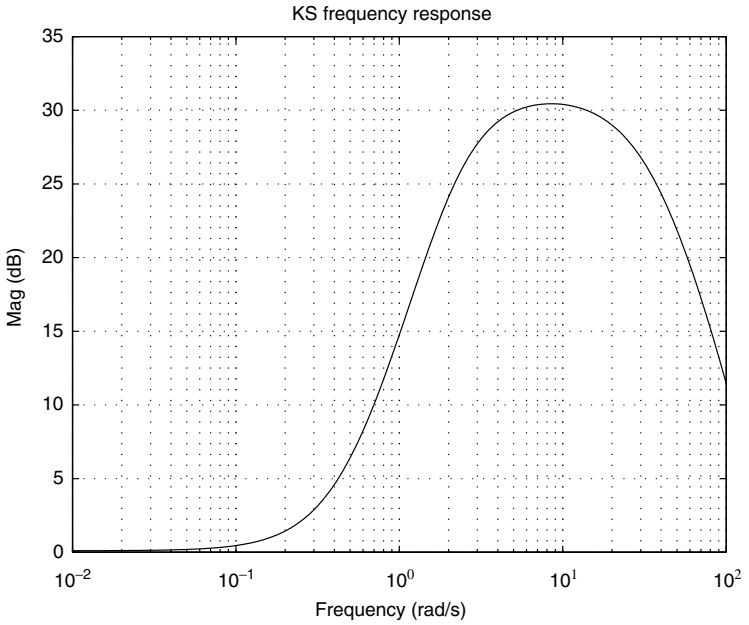


FIGURE 10.5 $\mathcal{H}^2$ Design *KS* frequency response.

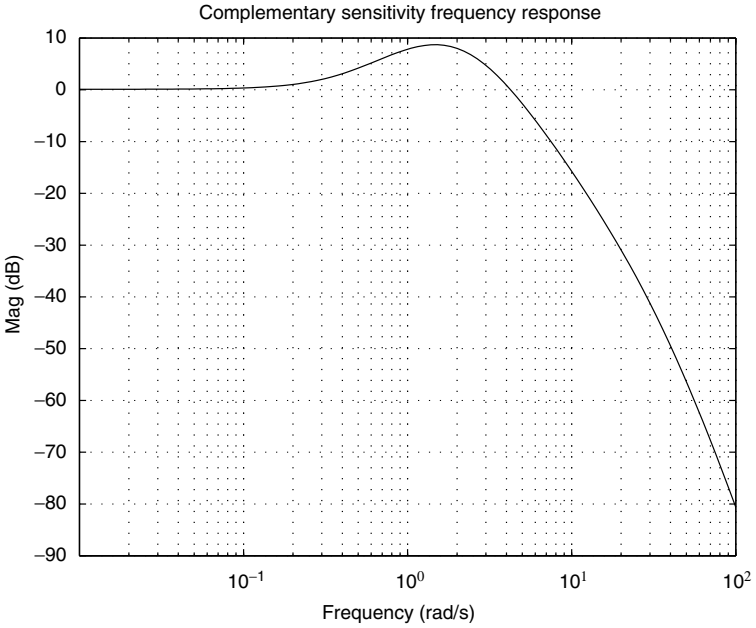


FIGURE 10.6 $\mathcal{H}^2$ Design complementary sensitivity frequency response.

Example 10.3 (LQG/LTR Design for First Order Unstable Missile Model)

We consider an unstable missile described by a simple first order model with state x (pitch attitude), control input u (fin elevator deflection), process noise $w_1 = \xi$ (angular wind gust), and sensor noise $w_2 = \theta$.

It is assumed that the missile's center of gravity (c.g.) is aft of its center of pressure (c.p.)—where lift is concentrated. This assumption results in a missile pitch instability. It is also assumed that the missile's

moment of inertia about its c.g. is very small. This assumption leads to a simple first order model. The missile's angular velocity $\dot{x}$ is assumed to be proportional to its attitude x and the process noise $w_1 = \xi$. Regulated signals $z = [z_1 \; z_2]^T$ include the vehicle's pitch attitude $z_1 = x$ and a weighted control input $z_2 = \sqrt{\rho}u$. Here, $\rho > 0$ is a design parameter to be selected below. The vehicle's pitch attitude is measured. The pitch attitude measurement y includes additive sensor noise $w_2 = \theta$.

Missile Model. The (generalized) missile model is given as follows:

$$\dot{x} = x + \xi + u \tag{10.94}$$

$$z = \begin{bmatrix} x \\ \sqrt{\rho}u \end{bmatrix} \tag{10.95}$$

$$y = x + \sqrt{\mu}\theta \tag{10.96}$$

where $\mu > 0$ is design parameter to be selected below.

Design Specifications. The goal is to design a real-rational proper model-based $\mathcal{H}^2$ optimal compensator (i.e., minimizes $\|T_{wz}\|_{H^2}$) which results in a stable closed loop system with a dominant closed loop pole at $s = -5$ (settling time $t_s \approx 1$ s).

Construction of Generalized Plant. The above model may be rewritten as follows:

$$\dot{x} = x + \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \theta \end{bmatrix} + u \tag{10.97}$$

$$z = \begin{bmatrix} 1 \\ 0 \end{bmatrix} x + 0_{2\times 2} \begin{bmatrix} \xi \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ \sqrt{\sigma} \end{bmatrix} u \tag{10.98}$$

$$y = x + \begin{bmatrix} 0 & \sqrt{\mu} \end{bmatrix} \begin{bmatrix} \xi \\ \theta \end{bmatrix} + 0_{1\times 1} u \tag{10.99}$$

From this, it follows that

$$A = 1, \qquad B_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad B_2 = 1 \tag{10.100}$$

$$C_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad D_{11} = 0_{2\times 2}, \qquad D_{12} = \begin{bmatrix} 0 \\ \sqrt{\rho} \end{bmatrix} \tag{10.101}$$

$$C_2 = 1, \qquad D_{21} = \begin{bmatrix} 0 & \sqrt{\mu} \end{bmatrix}, \qquad D_{22} = 0_{1\times 1} \tag{10.102}$$

$\mathcal{H}^2$ Problem Assumptions. We now check each of the $\mathcal{H}^2$ output feedback problem assumptions, as stated in Assumption 10.2. From the above data, it follows that $D_{11} = 0_{2\times 2}$, $D_{22} = 0_{1\times 1}$, and (A, B_2, C_2) is stabilizable and detectable, and

$$R = D_{12}^T D_{12} = \rho > 0 \tag{10.103}$$

$$\Theta = D_{21}^T D_{21} = \mu > 0 \tag{10.104}$$

Since

$$D_{12}^T C_1 = 0 \tag{10.105}$$

$$B_1 D_{21}^T = 0 \tag{10.106}$$

the imaginary axis rank conditions involving (A, B_2, C_1, D_{12}) and (A, B_1, C_2, D_{21}) in Assumption 10.2 become equivalent to (A, C_1) having no imaginary unobservable modes and (A, B_1) having no imaginary uncontrollable modes. These are clearly satisfied since $A = 1$ has no imaginary modes. Given this, it follows that all of the $\mathcal{H}^2$ output feedback problem assumptions in Assumption 10.2 are satisfied.

Plant. Finally, we note that the so-called plant (or missile) transfer function $P = G_{22}$ is given by

$$P = G_{22} = C_2(sI - A)^{-1}B_2 \tag{10.107}$$

$$= \frac{1}{s-1} \tag{10.108}$$

G_{22} is unstable with a right half plane pole at s = 1. G_{22} is also minimum phase (i.e., no zeros in Res > 0).

Filter Gain Matrix H_f. Since $B_1 D_{21}^T = 0$, the associated FARE is given by

$$AY + YA^T + B_1 B_1^T - YC_2^T \Theta^{-1} C_2 Y = Y + Y + 1 - \frac{1}{\mu} Y^2 = 0 \tag{10.109}$$

or

$$Y^2 - 2\mu Y - \mu = 0 \tag{10.110}$$

Application of the quadratic formula and selecting the positive (stabilizing) root yields:

$$Y = \mu + \sqrt{\mu^2 + \mu} \tag{10.111}$$

This yields the following filter gain matrix:

$$H_f = YC_2^T \Theta^{-1} = 1 + \sqrt{1 + \frac{1}{\mu}} \tag{10.112}$$

We now select μ to achieve the given dominant pole specification:

$$A - H_f C_2 = 1 - 1 - \sqrt{1 + \frac{1}{\mu}} = -5 \tag{10.113}$$

This yields

$$\mu = \frac{1}{24} \tag{10.114}$$

The associated KBF open loop transfer function is given by

$$G_{\mathrm{KF}} = -C_2(sI - A)^{-1}H_f \tag{10.115}$$

$$= \frac{-6}{s-1} \tag{10.116}$$

We will see below that this will be the approximate open loop transfer function PK_{opt} for the final design. In this sense, G_{KF} represents our target open loop transfer function.

Control Gain Matrix G_c. Since $D_{12}^T C_1 = 0$, the associated CARE is given by

$$A^T X + XA + C_1^T C_1 - XB_2 R^{-1} B_2 X = X + X + 1 - \frac{1}{\rho} X^2 = 0 \tag{10.117}$$

or

$$X^2 - 2\rho X - \rho = 0 \tag{10.118}$$

Application of the quadratic formula and selecting the positive (stabilizing) root yields:

$$X = \rho + \sqrt{\rho^2 + \rho} \tag{10.119}$$

This yields the following control gain matrix:

$$G_c = R^{-1} B_2^T X = 1 + \sqrt{1 + \frac{1}{\rho}} \tag{10.120}$$

This results in a closed loop (regulator) pole at

$$A - B_2 G_c = 1 - 1 - \sqrt{1 + \frac{1}{\rho}} = -\sqrt{1 + \frac{1}{\rho}} \tag{10.121}$$

Note that for large ρ (referred to as expensive control in LQR problems) we have a closed loop pole at s = −1, at the left half plane reflection of the plant pole at $s = 1$. We will select the design parameter ρ to be small (referred to as cheap control in LQR problems) so that this closed loop (regulator) pole $s \approx -1/\sqrt{\rho}$ is fast and the closed loop filter pole at $s = -5$ at is the dominant closed loop pole.

$\mathcal{H}^2$ Optimal Output Feedback Model-Based Compensator. The resulting $\mathcal{H}^2$ optimal output feedback model-based compensator is given by

$$K_{\text{opt}} = \left[\begin{array}{c|c} A - B_2 G_c - H_f C_2 & H_f \\ \hline -G_c & 0_{n_u \times n_y} \end{array}\right] \tag{10.122}$$

where

$$A - B_2 G_c - H_f C_2 = 1 - 1 - \sqrt{1 + \frac{1}{\rho}} - 1 - \sqrt{1 + \frac{1}{\mu}} \tag{10.123}$$

$$= 1 - 1 - \sqrt{1 + \frac{1}{\rho}} - 1 - \sqrt{1 + 24} \tag{10.124}$$

$$= -6 - \sqrt{1 + \frac{1}{\rho}} \tag{10.125}$$

$$H_f = 1 + \sqrt{1 + \frac{1}{\mu}} \tag{10.126}$$

$$= 1 + \sqrt{1 + 24} \tag{10.127}$$

$$= 6 \tag{10.128}$$

$$G_c = 1 + \sqrt{1 + \frac{1}{\rho}} \tag{10.129}$$

Given this, the compensator transfer function is given by

$$K_{\text{opt}} = -G_c(sI - A + B_2G_c + H_fC_2)^{-1}H_f \tag{10.130}$$

$$= \frac{-6(1+\sqrt{1+1/\rho})}{s+6+\sqrt{1+1/\rho}} \tag{10.131}$$

For small ρ (cheap control), this yields

$$K_{\text{opt}} \approx \frac{-6(1/\sqrt{\rho})}{s+1/\sqrt{\rho}} \tag{10.132}$$

Open Loop Transfer Function. The associated open loop transfer function is given by

$$PK_{\text{opt}} = -C_2(sI - A)^{-1}B_2 \quad G_c(sI - A + B_2G_c + H_fC_2)^{-1}H_f \tag{10.133}$$

$$= \frac{1}{s-1}\left[\frac{-6(1+\sqrt{1+1/\rho})}{s+6+\sqrt{1+1/\rho}}\right] \tag{10.134}$$

For small ρ (cheap control), this becomes

$$PK_{\text{opt}} \approx \frac{1}{s-1}\left[\frac{-6(1/\sqrt{\rho})}{s+1/\sqrt{\rho}}\right] \tag{10.135}$$

Loop Transfer Recovery (*LTR*). From this, we see that as control weighting parameter ρ approaches zero (cheap control), the open loop transfer function approaches the KBF open loop transfer function G_{KF}; that is,

$$\lim_{\rho \to 0^+} G_{22}K_{\text{opt}} = \frac{-6}{s-1} \tag{10.136}$$

$$= G_{\text{KF}} \tag{10.137}$$

This shows that as ρ approaches zero (cheap control), the actual open loop transfer function PK_{opt} approaches the target open loop transfer function G_{KF}. The above procedure of recovering a target open loop transfer function (with desirable closed loop properties) using an LQG controller is called *LQG with loop transfer recovery or LQG/LTR.*

Selection of Far Away Closed Loop Regulator Pole. For small ρ, the closed loop system is stable with closed loop poles at $s = -5$ and $s \approx -1/\sqrt{\rho}$. A good selection for ρ might be $\rho = 1/2500$. This results in a fast closed loop pole at $s \approx -50$ and makes the closed loop filter pole at $s = -5$ the dominant closed loop pole, as required.

Stability Robustness Margins. It is well known that $\mathcal{H}^2$ and LQG designs need not possess good stability robustness margins. In fact, they can be arbitrarily bad [3]. LQG/LTR designs for minimum phase plants (such as ours: P = 1/(s − 1) have guaranteed stability robustness margins. LQG/LTR designs provide margins that approach those associated with LQR and KBF designs; i.e., infinite upward gain margin, at least 6 dB downward gain margin, and at least ±60° phase margin. Our final LQG/LTR design

$$PK_{\text{opt}} = \frac{-6}{s-1}\left[\frac{50}{s+50}\right] \tag{10.138}$$

offers an infinite upward gain margin and a downward gain margin of 1/6 (−15.56 dB). The resulting unity gain crossover frequency is $\omega_g = \sqrt{35} = 5.92$ rad/s and the associated phase margin is about 99.59°. Not bad.

The following example extends the LQG/LTR ideas presented in Example 10.3 to the general MIMO setting—enabling the design of feedback loops (with nominal robustness margins) via $\mathcal{H}^2$ optimization. ■

Example 10.4 (MIMO LQG and LQG/LTR Control Design Via $\mathcal{H}^2$ Optimization)

We consider a MIMO plant P defined by the state space representation

$$\dot{x} = Ax + Bu \tag{10.139}$$

$$y = Cx \tag{10.140}$$

It is assumed that the plant $P = [A, B, C]$ is stabilizable and detectable.

The goal is to demonstrate how the $\mathcal{H}^2$ optimal output feedback solution that has been presented may be used to solve MIMO LQG control problems. We specifically would like to present a method which lends itself to the concept of LTR—whereby we use a model-based LQG controller to recover a target loop transfer function matrix with desirable closed loop properties. Our motivation is not optimal stochastic LQG control problems; it is the design of control laws with desirable closed loop properties.

Construction of Generalized Plant G. With our final objective being a model-based compensator defined by a control gain matrix G_c and a filter gain matrix H_f, we consider the following generalized plant:

$$\dot{x} = Ax + L\xi + Bu \tag{10.141}$$

$$z = \begin{bmatrix} \dfrac{Mx}{\sqrt{\rho} I_{n_u \times n_u}} \end{bmatrix} \tag{10.142}$$

$$y = Cx + \sqrt{\mu}\theta \tag{10.143}$$

where u is the control, x is the (generalized) plant state, $w_1 = \xi$ represents process noise in the state equation, $w_2 = \theta$ represents sensor noise in the measurement equation, $A \in \mathcal{R}^{n \times n}$, $L \in \mathrm{R}^{n \times n_u}$, $B \in \mathrm{R}^{n \times n_u}$, $M \in \mathrm{R}^{n_y \times n}$, $C \in \mathrm{R}^{n_y \times n}$, $n_y = n_u$, $\rho > 0$, $\mu > 0$.

Design Parameter Assumptions. It is assumed that either:

(A, L) has no imaginary uncontrollable modes and (A, M) is detectable, or
(A, L) is stabilizable and (A, M) has no imaginary unobservable modes.

Here, L, M, μ, and ρ should be viewed as "design parameters" that are selected in order to obtain control and filter gain matrices G, and H_f such that the resulting model-based compensator exhibits desirable closed loop properties.

Two-Port State Space Representation for Generalized Plant G. The above model may be rewritten in two-port state space form as follows:

$$\begin{bmatrix} \dot{x} \\ \hline z \\ \hline y \end{bmatrix} = \left[\begin{array}{c|c|c} A & \begin{bmatrix} L & 0_{n \times n_y} \end{bmatrix} & B \\ \hline \begin{bmatrix} M \\ 0_{n_u \times n} \end{bmatrix} & \begin{bmatrix} 0_{n_y \times n_u} & 0_{n_y \times n_y} \\ 0_{n_u \times n_u} & 0_{n_u \times n_y} \end{bmatrix} & \begin{bmatrix} 0_{n_y \times n_u} \\ \sqrt{\rho} I_{n_u \times n_u} \end{bmatrix} \\ \hline C & \begin{bmatrix} 0_{n_y \times n_u} & \sqrt{\mu} I_{n_y \times n_y} \end{bmatrix} & \end{array}\right] \begin{bmatrix} x \\ \hline \xi \\ \theta \\ \hline u \end{bmatrix} \tag{10.144}$$

Check on $\mathcal{H}^2$ Output Feedback Assumptions. We now make sure that all of the $\mathcal{H}^2$ output feedback problem assumptions in Assumption 10.2 are satisfied:

- *Plant $P = G_{22}$ Assumptions.* Since the plant $P = G_{22} = [\ A,\ B,\ C\ ]$ is stabilizable and detectable, it follows that $(A,\ B_2 = B,\ C_2 = C)$ is stabilizable and detectable.
- *Regulator Assumptions.* Since

$$D_{12} = \begin{bmatrix} 0_{n_y \times n_u} \\ \sqrt{\rho} I_{n_u \times n_u} \end{bmatrix}$$

has full column rank, it follows that the control weighting matrix $R = D_{12}^T D_{12} = \rho I_{n_u \times n_u} > 0$ is nonsingular.

Since $D_{12}^T C_1 = 0$, it follows that the imaginary axis (column) rank condition involving $(A,\ B_2,\ C_1,\ D_{12})$ in Assumption 10.2 is equivalent to $(A - B_2 R^{-1} D_{12}^T C_1, (I - D_{12} R^{-1} D_{12}^T) C_1) = (A, \mathrm{C}_1)$ having no unobservable imaginary modes. Since $(A,\ M)$ is either detectable or has no imaginary unobservable modes, it follows that

$$A, \mathrm{C}_1 = \begin{bmatrix} M \\ 0_{n_u \times n} \end{bmatrix}$$

has no unobservable imaginary modes. The associated Hamiltonian H_{con} will, therefore, yield a Riccati solution and control gain matrix G_c such that $A - BG_c$ is stable.

- *Filter Assumptions.* Since $D_{21} = [0_{n_y \times n_u}\ \sqrt{\mu} I_{n_y \times n_y}]$ has full row rank, it follows that the measurement weighting matrix $\Theta = D_{21} D_{21}^T = \mu I_{n_y \times n_y} > 0$ is nonsingular.

Since $B_1 D_{21}^T = 0$, it follows that the imaginary axis (row) rank condition involving $(A,\ B_1,\ C_2,\ D_{21})$ in Assumption 10.2 is equivalent to $(A - B_1 D_{21}^T \Theta^{-1} C_2, B_1 (I - D_{21} \Theta^{-1} D_{21})) = (A, B_1)$ having no uncontrollable imaginary modes. Since $(A,\ L)$ is either stabilizable or has no uncontrollable imaginary modes, it follows that $(A, \mathrm{B}_1 = [L\ \ 0_{n \times n_y}])$ has no uncontrollable imaginary modes. The associated Hamiltonian H_{fil} will therefore yield a Riccati solution and filter gain matrix H_f such that $A - H_f C$ is stable.

Given the above, it follows that all of the $\mathcal{H}^2$ output feedback problem assumptions in Assumption 10.2 are satisfied.

Control Gain Matrix. It follows that the control gain matrix G_c is given by

$$G_c = R^{-1} B^T X \tag{10.145}$$

where $X \geq 0$ is the unique (at least) positive semi-definite solution of the CARE:

$$A^T X + XA + C^T C - XBR^{-1} BX = 0 \tag{10.146}$$

Moreover, $A - BG$ is stable.

Filter Gain Matrix. It follows that the filter gain matrix H_f is given by

$$H_f = YC^T \Theta^{-1} \tag{10.147}$$

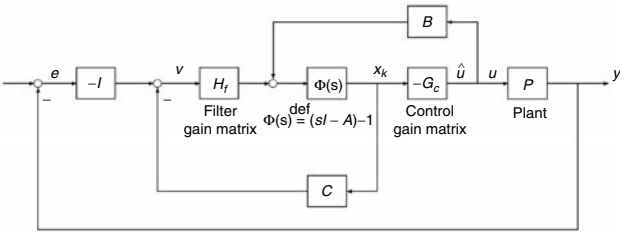


FIGURE 10.7 Negative feedback loop with LQG model-based compensator and plant.

where $Y > 0$ is the unique (at least) positive semi-definite solution of the FARE:

$$AY + YA^T + LL^T - YC^T\Theta^{-1}CY = 0 \tag{10.148}$$

Moreover, $A - HC$ is stable.

$\mathcal{H}^2$ Optimal (LQG) Compensator. The $\mathcal{H}^2$ optimal compensator that minimizes the $\mathcal{H}^2$ norm of the transfer function matrix from the exogenous signals $w = \begin{bmatrix} \xi \\ \theta \end{bmatrix}$ to the regulated signals $z = \begin{bmatrix} C_x \\ \sqrt{\rho} I_{n_u \times n_u} \end{bmatrix}$ is then given by

$$K_{\text{opt}} = \left[\begin{array}{c|c} A - BG_c - H_f C & H_f \\ \hline G_c & 0_{n_u \times n} \end{array}\right] \tag{10.149}$$

Note that the minus sign on G_c (lower left hand entry of K_{opt}) has been removed in anticipation of the negative feedback system implementation shown in Figure 10.7. By the separation principle, the closed loop poles are the eigenvalues of $A - BG_c$ and $A - H_fC$.

Stability Robustness Margins. It should be emphasized that the resulting controller K_{opt}, although stabilizing, may possess arbitrarily bad stability robustness margins [3]. This is despite the fact that the associated regulator loop

$$G_{LQ} = G_c(sI - A)^{-1}B \tag{10.150}$$

and filter loop

$$G_{\text{KF}} = C(sI - A)^{-1}H_f \tag{10.151}$$

when viewed as MIMO open loop transfer function matrices within their own negative feedback loops, possess the following well-known stability robustness margins: infinite upward gain margin, at least 6 dB downward gain margin, and at least ±60° phase margin. This gives rise to the following natural question:

Is there a way that we can select the control gain matrix G_c and the filter gain matrix H_f so that the resulting model-based compensator K_{opt} results in a feedback loop which possesses the above nice margins?

Fortunately, the answer to this is a definitive yes! Two methods which result in comparable stability margins at the plant input or at the plant output (but not both simultaneously) are now presented.

Loop Transfer Recovery (LTR) Methods. The approach we take to achieve a feedback design with good stability margins is as follows. The process involves two steps.

1. *Target Loop Design.* The first step is to design a target open loop transfer function matrix that possesses desired closed loop properties. The target loop may be associated with the plant output. If so, we denote it L_o. In such a case, L_o represents our desired PK_{opt}. If associated with the plant input, we denote it L_i. In such a case, L_i represents our desired K_{opt}. (In general, $PK_{\text{opt}}P \neq K_{\text{opt}}P$.)

2. *Target Loop Recovery Via Model-Based Compensator.* The second step is to use a model-based compensator $K_{opt} = [A - BG_c - H_fC, H, G]$ to recover the target loop (either L_o or L_i).

   If we want to recover L_o (i.e., good properties at that plant output), then we want $PK_{opt} \approx L_o$. This is called loop transfer recovery at the plant output (LTRO).

   If we want to recover L_i (i.e., good properties at that plant input), then we want $K_{opt}P \approx L_i$. This is called loop transfer recovery at the plant input (LTRI).

   Note: In general, the properties associated with breaking the loop at the plant output (properties of PK_{opt}) are different (perhaps very different) from those associated with breaking the loop at the plant input (properties of $K_{opt}P$). It is usually very difficult for PK_{opt} and $K_{opt}P$ to both possess great properties (e.g., margins, etc.). Typically, a designer must trade off nice properties at the plant output for nice properties at the plant input, or vice versa.

$\mathcal{H}^2$-based methods for LTRO and LTRI are now presented.

a. *Loop Transfer Recovery at Plant Output (LTRO)*

1. *Design of Target Loop L_o.* The first step is to design a target loop $L_o = C(sI - A)^{-1}H_f$ with desirable closed loop properties (e.g., stability, sensitivity, complementary sensitivity, stability robustness margins, etc). This may be done using any method! (Any method you feel comfortable enough with.)

   One procedure that results in good properties at the plant output is based on *KBF* methods. The idea is to select the design (shaping) matrix L so that the singular values $G_{FOL} = C(sI - A)^{-1}L$ look nice; for example, large minimum singular value at low frequencies, small maximum singular value at high frequencies, singular values cross 0 dB with slopes of −20 dB/dec, etc.

   We then solve the FARE with A, L, C, $\Theta = \mu I_{n_y \times n_y}$—using $\mu > 0$ to adjust the bandwidth of our target loop $L_o = G_{KF} = C(sI - A)^{-1}H_f$. A smaller (larger) μ results in a larger (smaller) bandwidth.

   Guidelines for Shaping of Target Loop $L_o = G_{KF}$.

- The so-called Kalman Frequency Domain Equality (KFDE) guides our loop shaping:

$$[I + G_{KF}(j\omega)][I + G_{KF}(j\omega)]^H = I + \left[\frac{1}{\sqrt{\mu}}G_{FOL}(j\omega)\right]\left[\frac{1}{\sqrt{\mu}}G_{FOL}(j\omega)\right]^H \tag{10.152}$$

From this, it follows that

$$\sigma_i[I + G_{KF}(j\omega)] = \sqrt{1 + \frac{1}{\mu}\sigma_i^2[G_{FOL}(j\omega)]} \tag{10.153}$$

This suggests that by shaping G_{FOL}, we can shape the target loop $L_o = G_{KF}$. Specifically, if G_{FOL} is large at low frequencies, then we expect

$$G_{KF}(j\omega) \approx \frac{1}{\sqrt{\mu}}G_{FOL}(j\omega) \tag{10.154}$$

at low frequencies. This shows that the matrix L should be used for shaping the target loop $L_o = G_{KF}$ while $\mu > 0$ is used to adjust the target loop bandwidth—decreasing/increasing μ to raise/lower the target loop bandwidth. The resulting loop $L_o = G_{KF}$ is guaranteed to possess nice closed loop properties as described below.

- The above singular value relation implies that

$$\sigma_{min}[I + G_{KF}(j\omega)] \geq 1 \tag{10.155}$$

for all ω. This, in turn, implies that the associated sensitivity singular values satisfy

$$\sigma_{\max}[S_{KF}(j\omega)] = \frac{1}{\sigma_{\min}[S_{KF}(j\omega)]^{-1}} \leq 1 \quad (0\text{dB}) \tag{10.156}$$

for all ω, where

$$S_{KF}(j\omega) = [I + G_{KF}(j\omega)]^{-1} \tag{10.157}$$

- From the above sensitivity singular value relationship, we obtain the follow celebrated KBF loop margins:

  infinite upward gain margin
  at least $\frac{1}{2}$ (6 dB) downward gain margin
  at least $\pm 60°$ phase margin

  The above gain margins apply to simultaneous and independent gain perturbations when the loop is broken at the output. The same holds for the above phase margins. The above margins are NOT guaranteed for simultaneous gain and phase perturbations. It should be noted that these margins can be easily motivated using elementary SISO Nyquist stability arguments [2,8].
- From the above sensitivity singular value relations, we obtain the following complementary sensitivity singular value relationship:

$$\sigma_{\max}[T_{KF}(j\omega)] = \sigma_{\max}[I - S_{KF}(j\omega)] \leq 1 + \sigma_{\max}[S_{KF}(j\omega)] \leq 2 \quad (6\text{dB}) \tag{10.158}$$

for all ω, where

$$T_{KF} = I - S_{KF} = G_{KF}[1 + G_{KF}]^{-1} \tag{10.159}$$

2. *Recovery of Target Loop L_o Using Model-Based Compensator.* The second step is to use a model-based compensator $K_{opt} = [\, A - BG_c - H_fC,\ H_f,\ G_c \,]$ where G_c is found by solving the CARE with A, B, $M = C$, $R = \rho I_{n_u \times n_u}$ with ρ a small positive scalar. Since ρ is small, we call this a cheap control problem.
   - If the plant $P = [A, B, C]$ is minimum phase, then it can be shown that

$$\lim_{\rho \to 0^+} X = 0 \tag{10.160}$$

$$\lim_{\rho \to 0^+} \sqrt{\rho} G_c = WC \tag{10.161}$$

   for some orthonormal W (i.e., $W^TW = WW^T = I$)

$$\lim_{\rho \to 0^+} PK_{opt} = L_o \tag{10.162}$$

   In such a case, $PK_{opt} \approx L_o$ for small ρ and hence PK_{opt} will possess stability margins that are close to those of L_o (at the plant output)—whatever method was used to design L_o. It must be noted that the minimum phase condition on the plant P is a suficient condition. It is not necessary. Moreover, G_c need not be computed using a CARE. In fact, any G_c which (1) satisfies a limiting condition $\lim_{\rho \to 0^+} \sqrt{\rho} G_c = WC$ for some invertible matrix W and which (2) ensures that $A - BG_c$ is stable (for small ρ), will result in LTR at the plant output. This result

is a consequence of the structure of model-based compensators and has nothing to do with optimal control and filtering problems.

- Assuming that a limiting condition $\lim_{\rho \to 0^+} \sqrt{\rho} G_c = WC$ holds for some invertible matrix W, loop transfer recovery of the target loop $L_o = C(sI - A)^{-1}H_f$ may be proven as follows: For small ρ we have

$$G_c \approx \frac{WC}{\sqrt{\rho}} \tag{10.163}$$

which gives yields the following:

$$PK_{\text{opt}} = PG_c(sI - A + BG_c + H_f C)^{-1}H_f \tag{10.164}$$

$$\approx P\frac{WC}{\sqrt{\rho}}\left(sI - A + B\frac{WC}{\sqrt{\rho}}\right)^{-1}H_f \tag{10.165}$$

$$\approx P\frac{WC}{\sqrt{\rho}}(sI - A)^{-1}\left[I + B\frac{WC}{\sqrt{\rho}}(sI - A)^{-1}\right]^{-1}H_f \tag{10.166}$$

$$\approx P\frac{WC(sI - A)^{-1}}{\sqrt{\rho}}\left[I + B\frac{WC(sI - A)^{-1}}{\sqrt{\rho}}\right]^{-1}H_f \tag{10.167}$$

$$\approx P\left[I + \frac{WC(sI - A)^{-1}}{\sqrt{\rho}}B\right]^{-1}\frac{WC(sI - A)^{-1}}{\sqrt{\rho}}H_f \tag{10.168}$$

$$\approx P\left[\frac{WP}{\sqrt{\rho}}\right]^{-1}W\frac{C(sI - A)^{-1}H_f}{\sqrt{\rho}} \tag{10.169}$$

$$\approx C(sI - A)^{-1}H_f = L_o \tag{10.170}$$

The central idea (underneath the algebra) is that as ρ goes to zero, the C feedback path within the compensator $K_{\text{opt}} = [A - BG_c - H_f C, G_c, H_f]$ is broken (see Figure10.7) and the nice properties that hold at the so-called innovations v (e.g., open loop transfer function matrix at v is $L_o = C(sI - A)^{-1}H_f$) in Figure 10.7 get transferred to the error signal e (compensator input, or plant output) within the feedback loop.

b. *Loop Transfer Recovery at Plant Input (LTRI).*

1. *Design of Target Loop L_i.* The first step is to design a target loop $L_i = G_C(sI - A)^{-1}B$ with desirable closed loop properties (e.g., stability, sensitivity, complementary sensitivity, stability robustness margins, etc). This may be done using any method! (Any method you feel comfortable enough with.)

   One procedure that results in good properties at the plant input is based on LQR methods. The idea is to select the design (shaping) matrix M so that the singular values of $G_{\text{OL}} = M(sI - A)^{-1}B$ look nice; for example, large minimum singular value at low frequencies, small maximum singular value at high frequencies, singular values cross 0 dB with slopes of −20 dB/dec, etc.

   We then solve the CARE with A, B, M, $R = \rho I_{n_u \times n_u}$, using $\rho > 0$ to adjust the bandwidth of our target loop $L_i = G_{\text{LQR}}G_c(sI - A)^{-1}B$. A smaller (larger) ρ results in a larger (smaller) bandwidth.

   Guidelines for Shaping of Target Loop $L_i = G_{\text{LQ}}$.

- The so-called LQ frequency domain equality (LQFDE) guides our loop shaping:

$$[I + G_{\text{LQ}}(j\omega)]^H[I + G_{\text{LQ}}(j\omega)] = I + \left[\frac{1}{\sqrt{\rho}}G_{\text{OL}}(j\omega)\right]^H\left[\frac{1}{\sqrt{\rho}}G_{\text{OL}}(j\omega)\right] \tag{10.171}$$

From this, it follows that

$$\sigma_i[I + G_{LQ}(j\omega)] = \sqrt{I + \frac{1}{\rho}\sigma_i^2[G_{OL}(j\omega)]} \tag{10.172}$$

This suggests that by shaping G_{OL}, we can shape the target loop $L_i = G_{LQ}$. Specifically, if G_{OL} is large at low frequencies, then we expect

$$G_{LQ}(j\omega) \approx \frac{1}{\sqrt{\rho}} G_{OL}(j\omega) \tag{10.173}$$

at low frequencies. This shows that the matrix M should be used for shaping the target loop $L_i = G_{LQ}$ while $\rho > 0$ is used to adjust the target loop bandwidth—decreasing/increasing ρ to raise/lower the target loop bandwidth. The resulting loop $L_i = G_{LQ}$ is guaranteed to possess nice closed loop properties as described below.

- The above singular value relation implies that

$$\sigma_{\min}[I + G_{LQ}(j\omega)] \geq 1 \tag{10.174}$$

for all ω. This, in turn, implies that the associated sensitivity singular values satisfy

$$\sigma_{\max}[S_{LQ}(j\omega)] = \frac{1}{\sigma_{\min}[S_{LQ}(j\omega)^{-1}]} \leq 1 \quad (0\text{dB}) \tag{10.175}$$

for all ω, where

$$S_{LQ} = [I + G_{LQ}]^{-1} \tag{10.176}$$

- From the above sensitivity singular value relationship, we obtain the follow celebrated LQR loop margins:

  infinite upward gain margin,
  at least $\frac{1}{2}$(6 dB) downward gain margin,
  at least $\pm 60°$ phase margin.

  The above gain margins apply to simultaneous and independent gain perturbations when the loop is broken at the input. The same holds for the above phase margins. The above margins are NOT guaranteed for simultaneous gain and phase perturbations. It should be noted that these margins can be easily motivated using elementary SISO Nyquist stability arguments [2,8].
- From the above sensitivity singular value relations, we obtain the following complementary sensitivity singular value relationship:

$$\sigma_{\max}[T_{LQ}(j\omega)] = \sigma_{\max}[I - S_{LQ}(j\omega)] \leq 1 + \sigma_{\max}[S_{LQ}(j\omega)] \leq 2 \quad (6\text{dB}) \tag{10.177}$$

for all ω, where

$$T_{LQ} = I - S_{LQ} = G_{LQ}[1 + G_{LQ}]^{-1} \tag{10.178}$$

2. Recovery of Target Loop L_i Using Model-Based Compensator. The second step is to use $K_{opt} = [A - BG_c, -H_fC, H_f, G_c]$ where H_f is found by solving the FARE with A, $L = B$, C, $\Theta = \mu I_{n_y \times n_y}$ with μ a small positive scalar. Since μ is small, we call this an expensive sensor problem.

- If the plant $P = [A, B, C]$ is minimum phase, then it can be shown that

$$\lim_{\mu \to 0^+} Y = 0 \tag{10.179}$$

$$\lim_{\mu \to 0^+} \sqrt{\mu} H_f = BV \tag{10.180}$$

for some orthonormal V (i.e., $V^T V = VV^T = I$)

$$\lim_{\mu \to 0+} K_{\text{opt}} P = L_i \tag{10.181}$$

In such a case, $K_{\text{opt}} P \approx L_i$ for small μ and hence $K_{\text{opt}} P$ will possess stability margins that are close to those of L_i (at the plant input)—whatever method was used to design L_i.

It must be noted that the minimum phase condition on the plant P is a suficient condition. It is not necessary. Moreover, H_f need not be computed using a FARE. In fact, any H_f which (1) satisfies a limiting condition $\lim_{\mu \to 0^+} \sqrt{\mu} H_f = BV$ for some invertible matrix V and which (2) ensures that $A - H_f C$ is stable (for small μ), will result in LTR at the plant input; that is, $\lim_{\mu \to 0^+} K_{\text{opt}} P = L_i$. This result is a consequence of the structure of model-based compensators and has nothing to do with optimal control and filtering problems.

- Assuming that a limiting condition $\lim_{\mu \to 0^+} \sqrt{\mu} H_f = BV$ holds for some invertible matrix V, loop transfer recovery of the target loop $L_i = G_c(sI - A)^{-1} B$ may be proven as follows. For small μ we have

$$H_f \approx \frac{BV}{\sqrt{\mu}} \tag{10.182}$$

which gives the following:

$$K_{\text{opt}} P = G_c(sI - A + BG_c + H_f C)^{-1} H_f P \tag{10.183}$$

$$\approx G_c\left(sI - A + BG_c + \frac{BV}{\sqrt{\mu}} C\right)^{-1} \frac{BV}{\sqrt{\mu}} P \tag{10.184}$$

$$\approx G_c\left(sI - A + \frac{BV}{\sqrt{\mu}} C\right)^{-1} \frac{BV}{\sqrt{\mu}} P \tag{10.185}$$

$$\approx G_c(sI - A)^{-1} + \left[I + \frac{BV}{\sqrt{\mu}} C(sI - A)^{-1}\right]^{-1} \frac{BV}{\sqrt{\mu}} P \tag{10.186}$$

$$\approx G_c(sI - A)^{-1} \frac{BV}{\sqrt{\mu}} \left[I + C(sI - A)^{-1} \frac{BV}{\sqrt{\mu}}\right]^{-1} P \tag{10.187}$$

$$\approx G_c(sI - A)^{-1} \frac{BV}{\sqrt{\mu}} \left[C(sI - A)^{-1} \frac{BV}{\sqrt{\mu}}\right]^{-1} P \tag{10.188}$$

$$\approx G_c(sI - A)^{-1} B \frac{V}{\sqrt{\mu}} \left[P \frac{V}{\sqrt{\mu}}\right]^{-1} P \tag{10.189}$$

$$\approx G_c(sI - A)^{-1} B = L_i \tag{10.190}$$

The central idea (underneath the algebra) is that as μ goes to zero, the B feedback path within the compensator $K_{\text{opt}} = [A - BG_c - H_f C, G_c, H_f]$ is broken (see Figure 10.7) and the nice properties

that hold at $\hat{u}$ (e.g., open loop transfer function matrix at $\hat{u}$ is $L_i = G_c(sI - A)^{-1}B$) in Figure 10.7 get transferred to the plant input u (compensator output) within the feedback loop. ■

Comment 10.13 (Stability Margins and Peak Sensitivity)

The peak on the sensitivity plot is very important in the design of a feedback system. A large peak, for example, may be due to a closed loop pole near the imaginary axis. This certainly is undesirable. We thus want the peak to be "small." It can be shown that the peak necessarily establishes gain and phase margin bounds.

Suppose that the peak sensitivity is bounded above by $\alpha \geq 1$; that is, $\sigma_{\max} S(j\omega) < \alpha$ for all ω. It can be shown that the feedback loop then enjoys the following nominal multivariable stability robustness (gain and phase) margin bounds:

$$\uparrow GM > \frac{\alpha}{\alpha - 1} \tag{10.191}$$

$$\downarrow GM < \frac{\alpha}{\alpha + 1} \tag{10.192}$$

$$|PM| > 2\sin^{-1}\left(\frac{1}{2\alpha}\right) \tag{10.193}$$

These bounds may be easily motivated using SISO Nyquist [2,8] ideas as follows:

If

$$|S(j\omega)| < \alpha \tag{10.194}$$

for all ω, then it follows that

$$\frac{1}{\alpha} < |1 + L(j\omega)| \tag{10.195}$$

for all ω. This, however, implies that the Nyquist plot associated with L cannot penetrate a circle centered at -1 with radius $1/\alpha$, left most end point at $-[(\alpha+1)/\alpha]$, and right most end point at $-[(\alpha-1)/\alpha]$. The upward gain margin bound follows from the right most point of the circle. The downward gain margin bound follows from the left most point of the circle. The phase margin bound can be obtained with a little geometry. ■

The following example considers the application of $\mathcal{H}^2$ theory to a robotic manipulator.

Example 10.5 ($\mathcal{H}^2$-LQG/LTR Design for PUMA 560 Robotic Manipulator)

In this example, we show how $\mathcal{H}^2$ optimization may be used to design an LQG/LTR controller for a PUMA 560 robotic manipulator. The manipulator is shown in Figure 10.8.

A two degree-of-freedom (dof) linear model $P = [A_p, B_p, C_p]$ was used to initiate the design process. Linearizing the PUMA's nonlinear model [9] about the equilibrium point $\theta_1 = 90°$ $\theta_2 = 0°$ (both links vertical), results in the following linear model:

$$\dot{x}_p = A_p x_p + B_p u_p \tag{10.196}$$

$$y_p = C_p x_p \tag{10.197}$$

$$u_p = [\tau_1 \quad \tau_2]^T \tag{10.198}$$

$$x_p = [\theta_1 \quad \theta_2 \quad \dot{\theta}_1 \quad \dot{\theta}_2]^T \tag{10.199}$$

$$y_p = [\theta_1 \quad \theta_2]^T \tag{10.200}$$

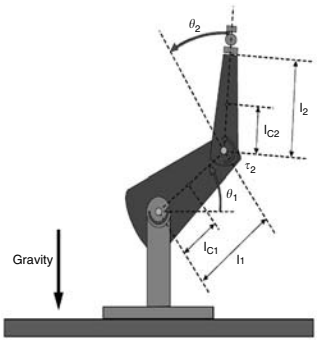


FIGURE 10.8 Two degree-of-freedom PUMA 560 robotic manipulator.

where

$$A_p = \begin{bmatrix} 0.0000 & 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \\ 31.7613 & -33.0086 & 0.0000 & 0.0000 \\ -56.9381 & 187.7089 & 0.0000 & 0.0000 \end{bmatrix} \tag{10.201}$$

$$B_p = \begin{bmatrix} 0.0000 & 0.0000 \\ 0.0000 & 0.0000 \\ 1037.7259 & -3919.6674 \\ -3919.6674 & 2030.8306 \end{bmatrix} \tag{10.202}$$

$$C_p = [I_{2\times 2} \quad 0_{2\times 2}] \tag{10.203}$$

The system poles are $s = \pm 14.1050$, $s = \pm 4.5299$. Eigenvector analysis shows that the fast instability at $s = 14.1050$ is primarily associated with the upper (shorter) link, while the slower instability at $s = 4.5299$ is primarily associated with the lower (longer) link. The system does not possess any natural integrators (i.e., no zero eigenvalues) and, as expected, the singular values $\sigma_i[P(j\omega)]$ are flat at low frequencies (see Figure 10.9).

Closed Loop Objectives

A controller to be implemented within a negative feedback loop is sought. The closed loop system should exhibit the following properties: (1) closed loop stability, (2) zero steady state error to step reference commands, (3) good low frequency reference command following (step commands followed with little overshoot within 3 s), (4) good low frequency disturbance attenuation, (5) good high frequency noise attenuation, (6) good stability robustness margins at the plant output.

Each step of the control system design process is now described. A central idea is the formation of a so-called design plant P_d from the original plant P. The design plant P_d is what is submitted to our $\mathcal{H}^2$-LQG/LTR design machinery.

Step 1: Augment Plant P with Integrators to Get Design Plant Pd = [A, B, C]

In order to guarantee zero steady-state error to step reference commands, we begin by augmenting the plant $P = [A_p, B_p, C_p]$ with integrators—one in each control channel—to form the design plant $P_d = [A, B, C]$; that is, $P_d = P(I_{2\times 2}/s)$. This is done as follows:

$$A = \begin{bmatrix} 0_{2\times 2} & 0_{2\times 4} \\ B_p & A_p \end{bmatrix} \tag{10.204}$$

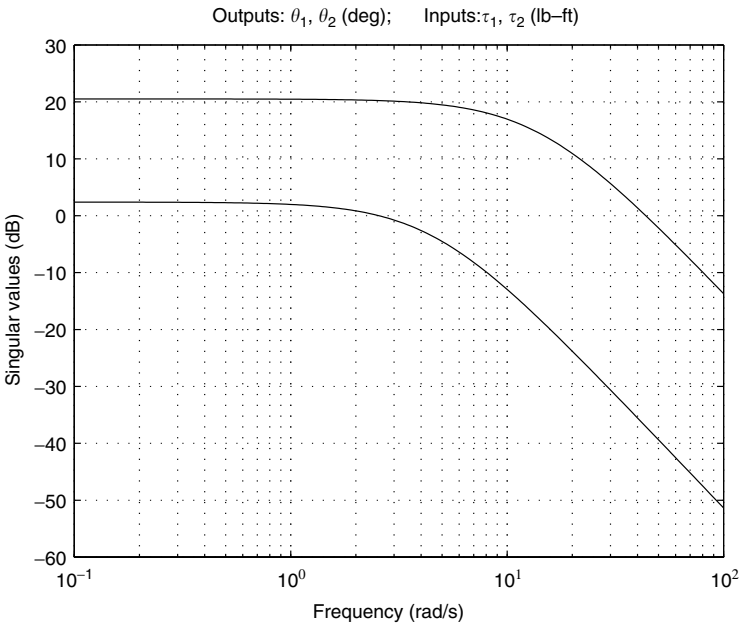


FIGURE 10.9 PUMA 560 robotic manipulator singular values.

$$B = \begin{bmatrix} 1_{2\times 2} \\ 0_{4\times 2} \end{bmatrix} \tag{10.205}$$

$$C = [0_{2\times 2} \quad C_p] \tag{10.206}$$

The state of this system is $x = \begin{bmatrix} x_i \\ x_p \end{bmatrix}$ where x_i is the integrator state and x_p is the plant state. The singular values for the augmented system P_d exhibit a slope of −20 dB/dec at low frequencies as expected (see Figure 10.10). The minimum singular value crosses zero dB just above 1 rad/s. The maximum singular value crosses zero dB at about 8 rad/s.

Step 2: Design Target Open Loop Transfer Function Matrix $L_o = G_{KF} = C(sI - A)^{-1}H_f$

Next we design a target open loop transfer function matrix $L_o = G_{KF} = C(sI - A)^{-1}H_f$ that has desirable closed loop properties (e.g., sensitivity singular values, pole locations, stability margins, etc.) at the output. To do this, we use Kalman Filtering ideas. Like LQR loops designed without a cross-state-control-coupling penalty, Kalman Filter loops designed in similar fashion exhibit desirable stability robustness margins (e.g., infinite upward gain margin, at least 6 dB downward gain margin, at least ±60° phase margin). This target loop design is carried out as follows:

- Consider the augmented system shown in Figure 10.11.

  It will be used to design a target loop transfer function matrix $L_o = G_{KF}$ with desirable closed loop properties at the output. To do so, we begin by forming an augmented system $G_{FOL} = C(sI - A)^{-1}L$ with

$$L = \begin{bmatrix} L_L \\ L_H \end{bmatrix} \tag{10.207}$$

$$L_L = [C_p(-A_p)^{-1}B_p]^{-1} \tag{10.208}$$

$$L_H = (-A_p)^{-1}B_pL_L \tag{10.209}$$

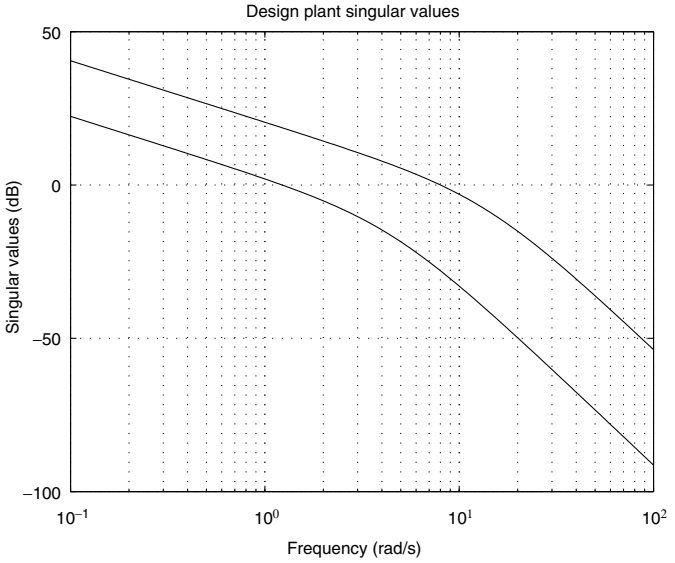


FIGURE 10.10 PUMA 560 robotic manipulator design plant singular values.

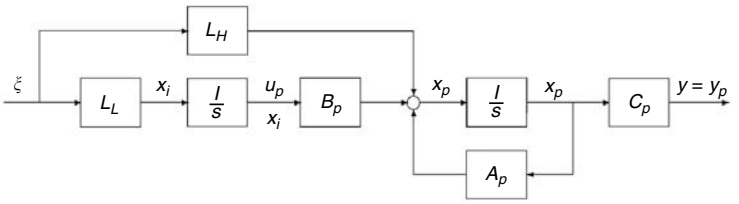


FIGURE 10.11 Augmented system used for designing target loop.

The matrix L_L matches the singular values of $G_{\text{FOL}} = C(sI - A)^{-1}L$ at low frequencies. The matrix L_H matches the singular values at high frequencies. Together, L_L and L_H match the singular values of $G_{\text{FOL}} = C(sI - A)^{-1}L$ at all frequencies (see Figure 10.12). Why is this?
This selection for L_L and L_H results in

$$G_{\text{FOL}} = C_p(sI - A_p)^{-1}L_H + C_p(sI - A_p)^{-1}B_p\left(\frac{I}{s}\right)L_L \tag{10.210}$$

$$= C_p(sI - A_p)^{-1}\left[L_H + B_p\left(\frac{I}{s}\right)L_L\right] \tag{10.211}$$

$$= C_p(sI - A_p)^{-1}\left[(-A_p)^{-1}B_pL_L + B_p\left(\frac{I}{s}\right)L_L\right] \tag{10.212}$$

$$= C_p(sI - A_p)^{-1}[sI - A_p](-A_p)^{-1}B_pL_L\left(\frac{I}{s}\right) \tag{10.213}$$

$$= C_p(-A_p)^{-1}B_pL_L\left(\frac{I}{s}\right) \tag{10.214}$$

$$= \frac{I}{s} \tag{10.215}$$

The resulting gain crossover frequency in Figure 10.12 is 1 rad/s, as expected.

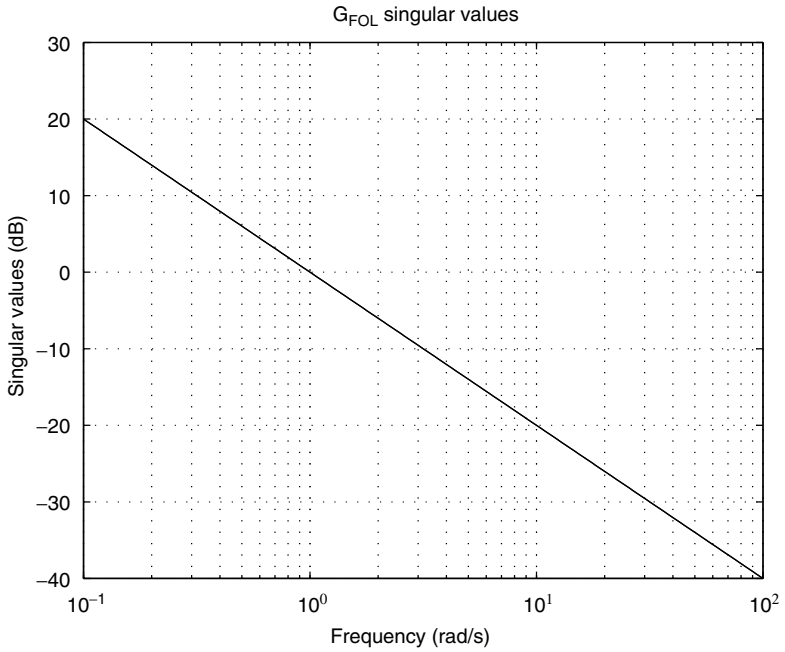


FIGURE 10.12 PUMA 560 robotic manipulator G_{FOL} singular values.

Why match the singular values of G_{FOL} in this manner? From the so-called Kalman Frequency Domain Equality (KFDE), it follows that

$$\sigma_i[I + G_{\text{KF}}(j\omega)] = \sqrt{1 + \frac{1}{\mu}\sigma_i^2[G_{\text{FOL}}(j\omega)]} \tag{10.216}$$

This suggests that by shaping G_{FOL}, we can shape the target $L_o = G_{\text{KF}}$. Specifically, if G_{FOL} is large at low frequencies, then we expect (from KFDE)

$$L_o(j\omega) = G_{\text{KF}}(j\omega) \approx \frac{1}{\sqrt{\mu}} G_{\text{FOL}}(j\omega) \tag{10.217}$$

at low frequencies. This shows that the matrix L should be used for shaping the target loop $L_o = G_{\text{KF}}$ while $\mu > 0$ is used to adjust the target loop bandwidth—decreasing/increasing μ to raise/lower the target loop bandwidth.

Note that through our selection of L, we have made all of the plant's unstable modes uncontrollable through L. Hence, (A, L) is NOT stabilizable! While this might appear to be troublesome, it is not. What matters is that the associated Hamiltonian belongs to $dom(Ric)$ so that a stabilizing H_f exists. A necessary and suficient condition for this, however, is that (A, C) be detectable and (A, L) has no unobservable modes on the imaginary axis. Since each of these conditions are indeed satisfied, we can use the "are" command to find a stabilizing solution to the FARE.

- Next we solved the FARE with $\Theta = \mu I_{2\times 2}$ $(\mu = 0.1)$:

$$AY + YA^T + LL^T - YC^T\Theta^{-1}CY = 0 \tag{10.218}$$

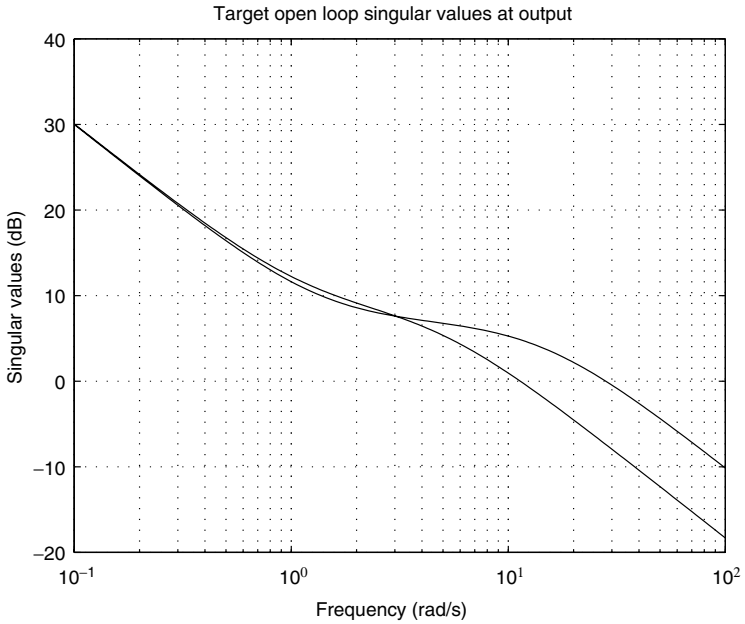


FIGURE 10.13 PUMA 560 robotic manipulator target loop G_{KF} singular values.

for $Y \geq 0$. The "are" command was used to do this, as it returns a stabilizing solution (provided that one exists). We then formed the filter gain matrix

$$H_f = YC^T\Theta^{-1} \tag{10.219}$$

$$= \begin{bmatrix} 2.3635 & 0.0384 \\ 0.4085 & 0.3091 \\ 13.1371 & -4.2300 \\ -4.2300 & 30.4572 \\ 90.2377 & -83.4384 \\ -100.9668 & 467.7679 \end{bmatrix} \tag{10.220}$$

Doing so results in the following target closed loop poles ($\lambda_i(A - H_fC)$):

$$s = -3.1623, -3.1623, -4.5299, -4.5299, -14.1050, -14.1050 \tag{10.221}$$

The singular values for the resulting target open loop transfer function matrix $L_o = G_{\mathrm{KF}} = C(sI - A)^{-1}H_f$ are shown in Figure 10.13.

The target open loop singular values—as expected from the KFDE—are matched at low frequencies with a slope of −20 dB/dec. They remain matched til about 1 rad/s, then they separate. This is expected since $G_{\mathrm{FOL}} = I/s$ is not an achievable loop. (Not if closed loop stability matters!) The resulting filter gain matrix provides the necessary bandwidth to stabilize the unstable robotic manipulator, with open loop instabilities at $s = 14.1050, 4.5299$. One singular value crosses 0 dB just above 10 rad/s, the other just below 30 rad/s. μ was used to adjust the bandwidth.

The corresponding target sensitivity $S_{\mathrm{KF}} = [I + G_{\mathrm{KF}}]^{-1}$ singular values and complementary sensitivity $T_{\mathrm{KF}} = G_{\mathrm{KF}}[I + G_{\mathrm{KF}}]^{-1}$ singular values are shown in Figures 10.14 and 10.15, respectively. The associated

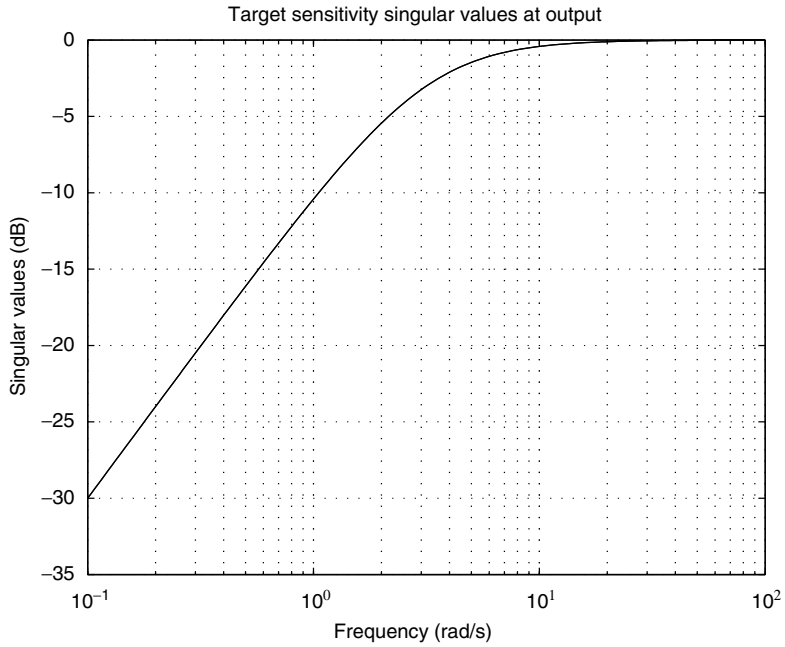


FIGURE 10.14 PUMA 560 robotic manipulator target sensitivity $S_{KF} = [I + G_{KF}]^{-1}$ singular values.

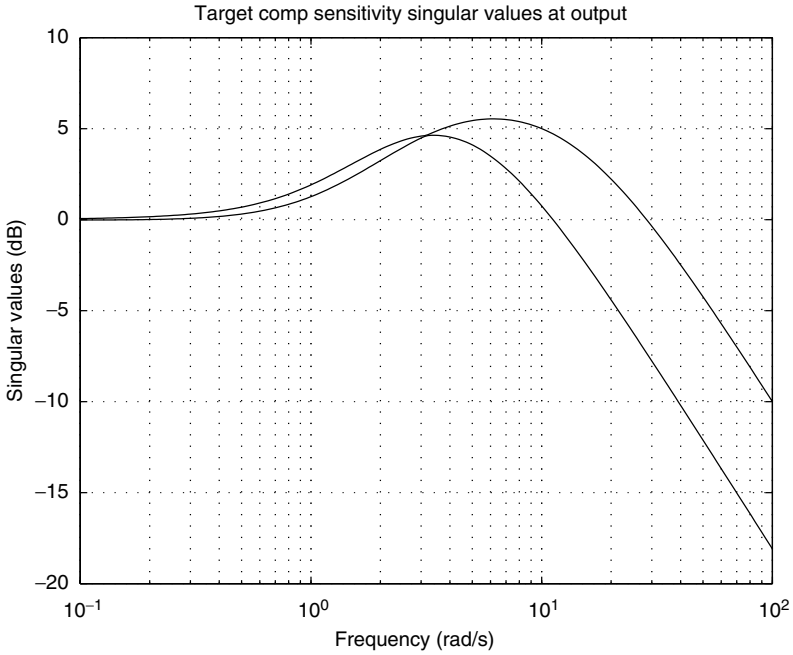


FIGURE 10.15 PUMA 560 robotic manipulator target complementary sensitivity $T_{KF} = G_{KF}[I + G_{KF}]^{-1}$ singular values.

sensitivity and complementary sensitivity singular values are desirable in that they suggest that the target loop will possess:

- Good low frequency command following properties
- Good low frequency disturbance attenuation properties
- Good high frequency sensor noise attenuation properties
- Good MIMO stability margins (nearly infinite upward gain margin, at least 6 dB downward gain margin, and at least ±60° phase margin) at the output

The complementary sensitivity singular values suggest that a reference command prefilter W would reduce overshoot due to step reference commands. The design of such a filter will be considered below.

Step 3: Solve Cheap Control Problem to Recover Target Loop at Plant Output

Next we solve an appropriately formulated "cheap LQR control problem" that would produce a control gain matrix G_c such that the $\mathcal{H}^2$ optimal model-based compensator $K_d = [A - BG_c - H_fC, H_f, G_c]$ with $P_d = [A, B, C]$ approximates ("recovers") the target loop transfer function matrix $L_o = G_{KF}$; that is,

$$P_d K_d \approx L_o = G_{KF} \tag{10.222}$$

This was done by solving the following CARE (using the "lqr" command) with $R = \rho I_{2\times 2} (\rho = 10^{-13})$:

$$XA + A^T X + C^T C - XBR^{-1}B^T X = 0 \tag{10.223}$$

for $X \geq 0$ and forming the control gain matrix

$$G_c = R^{-1}B^T X \tag{10.224}$$

$$= \begin{bmatrix} 987.9832 & -543.0034 & 3162945.2928 & 56.9921 & 13941.9005 & 2069.8324 \\ -543.0034 & 3657.5891 & 11.7867 & 3162634.3919 & 2069.7987 & 3765.7978 \end{bmatrix} \tag{10.225}$$

Doing so yields the following closed loop regulator poles ($\lambda_i(A - BG_c)$):

$$s = -440.8808, -220.4404 \pm j381.7871, -1881.9053, -940.9527 \pm j1629.7168 \tag{10.226}$$

All have damping factors greater than or equal to $\zeta = 0.5$. As a practical note to facilitate real-time implementation of the resulting controller, one might use model reduction techniques [10] to remove some of the very high frequency poles in the compensator. Doing so would permit using a larger integration step size in any real-time embedded system or microprocessor implementation.

Step 4: Construct Final Controller K

Next we form the final controller as follows:

$$K = \frac{K_d}{s} \tag{10.227}$$

$$= \frac{[A - BG_c - H_fC, H_f, G_c]}{s} \tag{10.228}$$

$$= [A_k, B_k, C_k] \tag{10.229}$$

A state space representation for this controller is given by

$$A_K = \begin{bmatrix} 0_{2\times 2} & G_c \\ 0_{6\times 2} & A - BG_c - H_fC \end{bmatrix}, \quad B_K = \begin{bmatrix} 0_{2\times 2} \\ H_f \end{bmatrix} \tag{10.230}$$

$$C_K = \begin{bmatrix} I_{2\times 2} & 0_{2\times 6} \end{bmatrix} \tag{10.231}$$

With this selection, we have

$$PK = P\frac{K_d}{s} \tag{10.232}$$

$$= P\frac{I_{2\times 2}}{s}K_d \tag{10.233}$$

$$= P_d K_d \tag{10.234}$$

$$\approx L_o = G_{\mathrm{KF}} \tag{10.235}$$

Through this selection of K, we have recovered the target loop transfer function matrix $L_o = G_{\mathrm{KF}}$. That is, K has approximately inverted P (from the right) in order to achieve $PK \approx L_o = G_{\mathrm{KF}}$. An examination of the singular values for the actual loop PK shows that the actual singular values agree with the target singular values up to and beyond 100 rad/s.

Loop Transfer Recovery. Why were we able to recover the target loop? The recovery was permitted by the model-based structure of the compensator K, the Riccati equations used to obtain the gain matrices G_c and H_f, and the fact that the plant $P = [A_p, B_p, C_p]$ (and hence the design plant $P_d = [A, B, C] = P(I/s)$ is minimum phase. The minimum phase condition, specifically, is a suficient condition which guarantees that there exists an orthonormal matrix $U(U^TU = UU^T = I)$ such that

$$\lim_{\rho\to 0^+} \sqrt{\rho} G_c = UC \tag{10.236}$$

This limiting behavior relating the control gain matrix and the design plant's C matrix, however, can be used to prove that loop transfer recovery takes place; that is,

$$\lim_{\rho\to 0^+} P_d K_d = \lim_{\rho\to 0^+} PK = L_o = G_{\mathrm{KF}} \tag{10.237}$$

Step 5: Design Command Pre-filter W
The MATLAB command

$$t_0(a - b * g - h * c, h, g) \tag{10.238}$$

can be used to find the compensator's transmission zeros. These are also zeros of the closed loop transfer function matrix from r to y. The final compensator (as well as the target loop G_{KF}) has zeros near $s \approx -1.2$. Given this, a reference command prefilter

$$W = \frac{1.2}{s + 1.2} I_{2\times 2} \tag{10.239}$$

was added outside the loop to filter reference commands. By so doing, we ensure that step reference commands for θ_1 and θ_2 are followed in the steady state (due to integrators in controller) without excessive overshoot during the transient.

Sensitivity Frequency Response
The resulting sensitivity singular values are plotted in Figure 10.16. The plot suggests that low frequency reference commands r will be followed and low frequency output disturbances d_o will be attenuated. More precisely, reference commands r with frequency content below 0.3 rad/s should be followed to within about 20 dB; that is, with a steady-state error of about 10%. Similarly, output disturbances d_o with frequency content below 0.3 rad/s should be attenuated by approximately 20 dB.

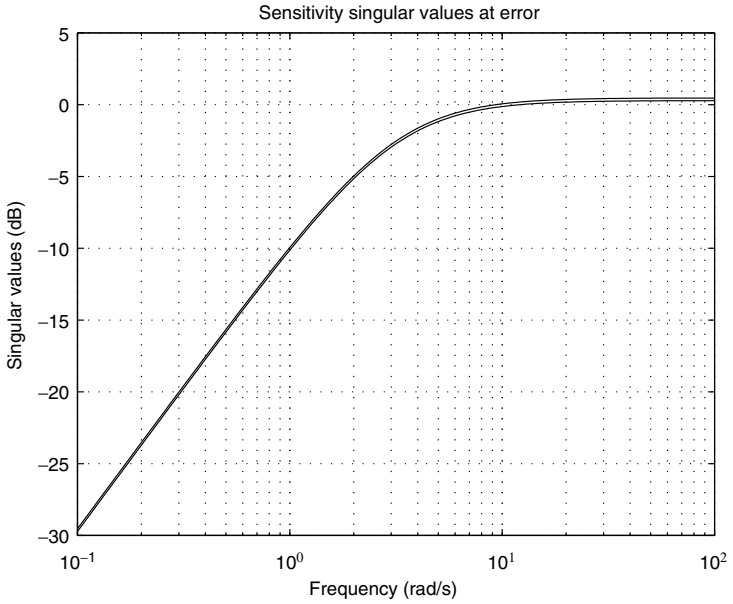


FIGURE 10.16 PUMA 560 sensitivity frequency response at error.

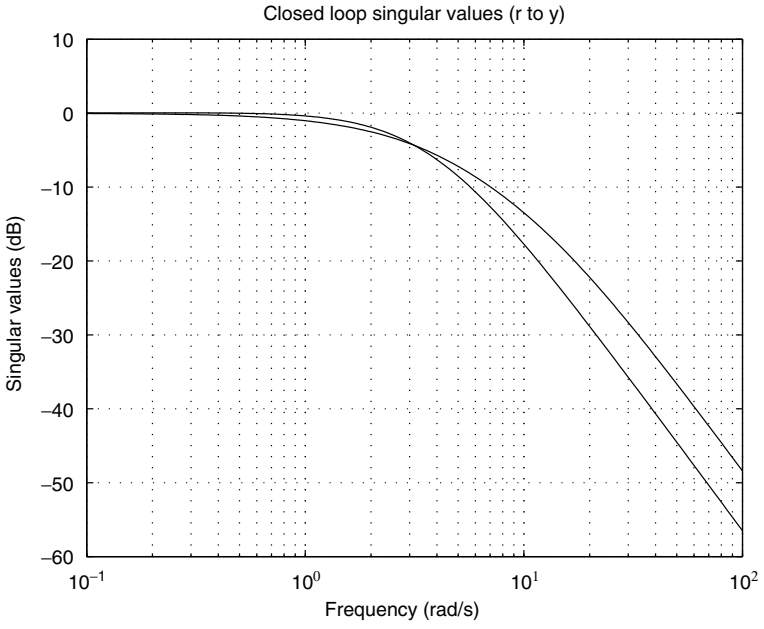


FIGURE 10.17 PUMA 560 reference to output frequency response.

Reference to Output Frequency Response

The transfer function matrix from reference commands r to link angles y is

$$T_{ry} = [I + PK]^{-1} PKW \tag{10.240}$$

Its singular values are plotted in Figure 10.17. The plot suggests that low frequency reference commands will be followed in the steady state and that little overshoot will result during the transient.

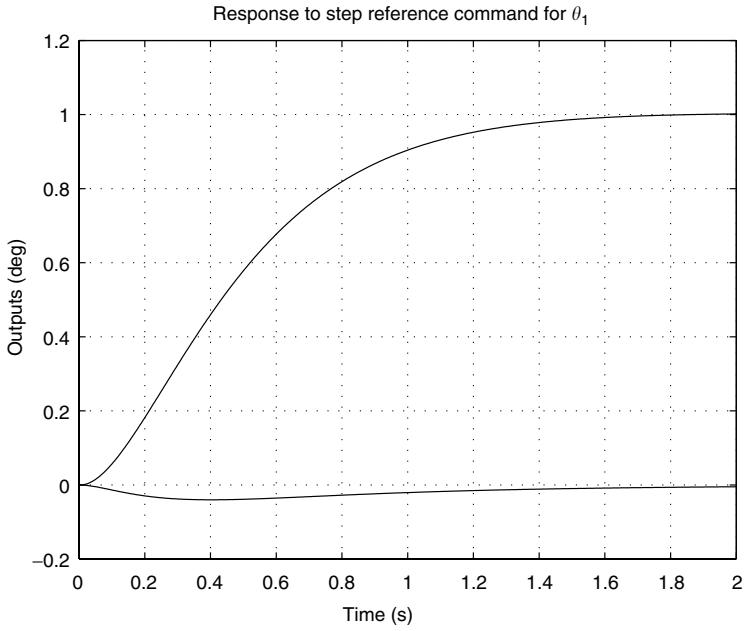


FIGURE 10.18 PUMA 560 outputs: response to θ_1 reference command.

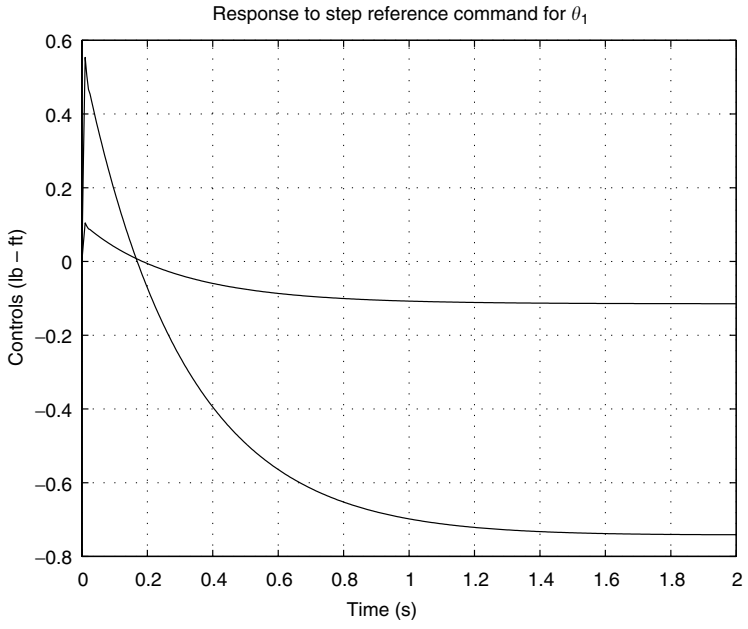


FIGURE 10.19 PUMA 560 controls: response to θ_1 reference command.

Response to Θ_1 *Step Reference Command*

The response to a unit step θ_1 command is plotted in Figure 10.18. As expected, θ_1 follows the step command well, with no overshoot and settling in about 1.6 s. The associated θ_2 response is small, indicating little cross coupling in the final closed loop system. The corresponding controls are plotted in Figure 10.19. They are acceptable in size.

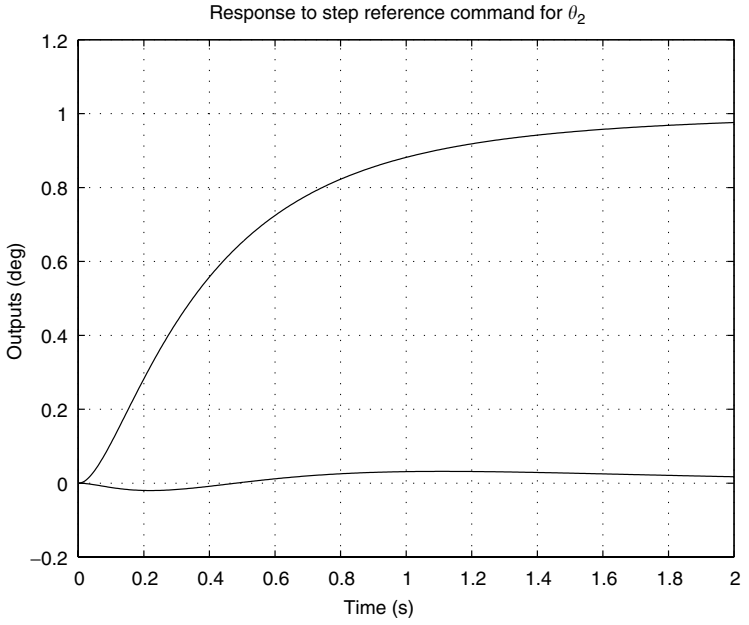


FIGURE 10.20 PUMA 560 outputs: response to θ_2 reference command.

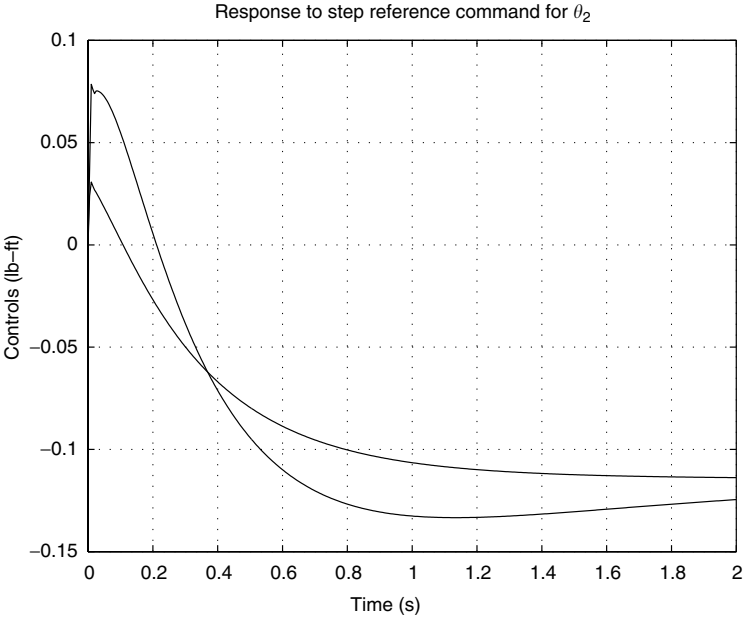


FIGURE 10.21 PUMA 560 controls: response to θ_2 reference command.

Response to Θ_2 Step Reference Command

The response to a unit step θ_2 command is plotted in Figure 10.20. As expected, θ_2 follows the step command well, with no overshoot and settling in about 3 s. The associated θ_1 response is small, indicating little cross coupling in the final closed loop system. The corresponding controls are plotted in Figure 10.21. They are acceptable in size. ■

10.4 $\mathcal{H}^2$ State Feedback Problem

This section shows that the methods presented for output feedback may be readily adopted to permit the design of $\mathcal{H}^2$ optimal constant gain state feedback control laws (control gain matrices G_c) as well.

10.4.1 Generalized Plant Structure for State Feedback

For this case, the generalized plant G (including plant P and weighting functions) takes the following form:

$$G = \left[\begin{array}{c|c} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right] = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & 0_{n_z \times n_w} & D_{12} \\ I_{n\times n} & 0_{n_y \times n_w} & 0_{n_y \times n_u} \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{10.241}$$

This implies that the measured signals y are the states x of the generalized plant G. As such, all of the modes of A are observable through $C_2 = I_{n\times n}$.

10.4.2 State Feedback Assumptions

The standard state feedback assumptions are a subset of those required for the output feedback problem formulation. The state feedback assumptions are as follows.

Assumption 10.2 ($\mathcal{H}^2$ State Feedback Problem)

Throughout this section, it will be assumed that

1. *Plant G_{22} Assumption.* (A, B_2) stabilizable.
2. *Nonsingular Control Weighting Assumption.* $R = D_{12}^T D_{12} > 0$ (D_{12} full column rank).
3. *Regulator Assumption.* $\begin{bmatrix} j\omega I - A & -B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all ω. ■

It should be noted that if $D_{12}^T C_1 = 0$, then (3) is equivalent to (A, C_1) having no unobservable imaginary modes. If (A, C_1) is detectable, then this is satisfied.

$\mathcal{H}^2$ Optimal State Feedback Control Law

The $\mathcal{H}^2$ optimal controller is given by

$$K_{\text{opt}} = -G_c \tag{10.242}$$

where the control gain matrix $G_c \in \mathrm{R}^{n_u \times n}$ is given by

$$G_c = R^{-1}[B_2^T X + D_{12}^T C_1] \tag{10.243}$$

where $X \geq 0$ is the unique (at least) positive semi-definite solution of the CARE:

$$(A - B_2 R^{-1} D_{12}^T C_1)^T X + X(A - B_2 R^{-1} D_{12}^T C_1) + C_1^T (I - D_{12}^T R^{-1} D_{12}^T) C_1 - X B_2 R^{-1} B_2^T X = 0 \tag{10.244}$$

The closed loop poles that result from the above constant gain state feedback control law are the eigenvalues of $A - B_2 G_c$. The minimum closed loop norm is given by

$$\min_K \|T_{wz}\|_{\mathrm{H}^2} = \sqrt{trace(B_1^T X B_1)} \tag{10.245}$$

State Feedback Loop Shaping

If one selects

$$B_2 = B \tag{10.246}$$

$$C_1 = \begin{bmatrix} M \\ 0_{n_u \times n} \end{bmatrix} \tag{10.247}$$

$$D_{12} = \begin{bmatrix} 0_{n_y \times n_u} \\ \sqrt{\rho} I_{n_u \times n_u} \end{bmatrix} \tag{10.248}$$

$$R = \rho I_{n_u \times n_u} \tag{10.249}$$

then $D_{12}^T C_1 = 0$ and we have

$$G_c = \frac{1}{\rho} B_2^T X \tag{10.250}$$

where $X \geq 0$ is the unique (at least) positive semi-definite solution of the CARE:

$$A^T X + XA + M^T M - XB\frac{1}{\rho}B^T X = 0 \tag{10.251}$$

The following LQFDE may be derived from the CARE:

$$[I + G_{\mathrm{LQ}}(j\omega)]^H [I + G_{\mathrm{LQ}}(j\omega)] = I + \left[\frac{1}{\sqrt{\rho}} G_{\mathrm{OL}}(j\omega)\right]^H \left[\frac{1}{\sqrt{\rho}} G_{\mathrm{OL}}(j\omega)\right] \tag{10.252}$$

where

$$G_{\mathrm{OL}} = M(sI - A)^{-1} B \tag{10.253}$$

$$G_{\mathrm{LQ}} = G_c(sI - A)^{-1} B \tag{10.254}$$

Given this, the loop shaping ideas discussed earlier are applicable. A designer may use the matrix M and the scalar $\rho > 0$ to shape G_{OL} in an effort to get a desirable loop G_{LQ}. The matrix M, specifically, may be used to match singular values at low frequencies, high frequencies, all frequencies, etc. Assuming that (A, B) is stabilizable and (A, M) has no imaginary modes that are unobservable, a stabilizing solution is guaranteed to exist. Moreover, the resulting G_{LQ} loop will possess nominal sensitivity and stability robustness properties—a consequence of the LQFDE. The resulting control gain matrix G_c may be used within a state feedback loop, a modified state feedback loop, or within a model-based compensator.

10.5 $\mathcal{H}^2$ Output Injection Problem

This section shows how the methods presented for output feedback may be readily adopted to permit the design of $\mathcal{H}^2$ optimal state estimators (filter gain matrices H_f) as well.

10.5.1 Generalized Plant Structure for Output Injection

For this case (dual to the state feedback case), the generalized plant G (including plant P and weighting functions) takes the following form:

$$G = \left[\begin{array}{c|c} G_{11} & G_{12} \\ \hline G_{21} & G_{22} \end{array}\right] = \left[\begin{array}{c|cc} A & B_1 & I_{n\times n} \\ \hline C_1 & 0_{n_z\times n_w} & 0_{n_z\times n_u} \\ C_2 & D_2 & 0_{n_y\times n_u} \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{10.255}$$

This implies that the control signals u directly impact all of the generalized plant states x. As such, all of the modes of A are controllable through $B_2 = I_{n\times n}$.

10.5.2 Output Injection Assumptions

The standard output injection assumptions are a subset of those required for the output feedback problem formulation. The output injection assumptions are as follows.

Assumption 10.3 ($\mathcal{H}^2$ Output Injection Problem)

Throughout this section, it will be assumed that

1. *Plant G_{22} Assumption.* (A, C_2) detectable.
2. *Nonsingular Measurement Weighting Assumption.* $\Theta = D_{21}D_{21}^T > 0$ (D_{21} full row rank).
3. *Filter Assumption.* $\begin{bmatrix} j\omega I - A & -B \\ C_1 & D_{21} \end{bmatrix}$ has full row rank for all ω ■

It should be noted that if $B_1 D_{21}^T = 0$, then (3) is equivalent to (A, B_1) having no uncontrollable imaginary modes. If (A, B_1) is stabilizable, then this is satisfied.

$\mathcal{H}^2$ Optimal Output Injection Law

The $\mathcal{H}^2$ optimal controller is then given by

$$K_{\text{opt}} = -H_f \tag{10.256}$$

where the filter gain matrix $H_f \in \mathbf{R}^{n\times n_y}$ is given by

$$H_f = [YC_2^T + B_1 D_{21}^T]\Theta^{-1} \tag{10.257}$$

where $Y \geq 0$ is the unique (at least) positive semi-definite solution of the FARE:

$$(A - B_1 D_{21}^T \Theta^{-1} C_2)Y + Y(A - B_1 D_{21}^T \Theta^{-1} C_2)^T + B_1(I - D_{21}^T \Theta^{-1} D_{21})B_1^T - YC_2^T \Theta^{-1} C_2 Y = 0 \tag{10.258}$$

The closed loop poles that result from the above output injection law are the eigenvalues of $A - H_f C_2$. The minimum closed loop norm is given by

$$\min_K \|T_{wz}\|_{H^2} = \sqrt{trace\left(C_1 Y C_1^T\right)} \tag{10.259}$$

where Y is the solution to the FARE.

Estimator (Filter) Loop Shaping
If one selects

$$B_1 = [L \quad 0_{n\times n_y}] \tag{10.260}$$

$$D_{21} = [0_{n_y\times n_u} \quad \sqrt{\mu} I_{n_y\times n_y}] \tag{10.261}$$

$$C_2 = C \tag{10.262}$$

$$\Theta = \mu I_{n_y\times n_y} \tag{10.263}$$

then $B_1 D_{21}^T = 0$ and we have

$$H_f = YC_2^T \frac{1}{\mu} \tag{10.264}$$

where $Y \geq 0$ is the unique (at least) positive semi-definite solution of the FARE:

$$YA^T + AY + LL^T - YC^T\Theta^{-1}CY = 0 \tag{10.265}$$

Given this, the following KFDE may be derived from the FARE:

$$[I + G_{\mathrm{KF}}(j\omega)][I + G_{\mathrm{KF}}(j\omega)]^H = I + \left[\frac{1}{\sqrt{\mu}} G_{\mathrm{FOL}}(j\omega)\right]\left[\frac{1}{\sqrt{\mu}} G_{\mathrm{FOL}}(j\omega)\right]^H \tag{10.266}$$

where

$$G_{\mathrm{FOL}} = C(sI - A)^{-1}L \tag{10.267}$$

$$G_{\mathrm{KF}} = C(sI - A)^{-1}H_f \tag{10.268}$$

Given this, the loop shaping ideas discussed earlier are applicable. A designer may use the matrix L and the scalar $\mu > 0$ to shape G_{FOL} in an effort to get a desirable loop G_{KF}. The matrix L, specifically, may be used to match singular values at low frequencies, high frequencies, all frequencies, etc. Assuming that (A, C) is detectable and (A, L) has no imaginary modes that are uncontrollable, then a stabilizing solution is guaranteed to exist. Moreover, the resulting G_{KF} loop will possess nominal sensitivity and stability robustness properties—a consequence of the KFDE. The resulting filter (output injection) gain matrix H_f may be used within an estimator (feedback) loop, a modified estimator (feedback) loop, or within a model-based compensator.

10.6 Summary

This chapter has presented a general framework for control system design via $\mathcal{H}^2$ optimization. While the focus has been on continuous time LTI systems, the methods are very flexible and have wide application. They may be used to design constant gain state feedback control laws, constant gain state estimators, dynamic output feedback controllers, and much more. Weighting functions are easily accommodated within the generalized plant framework presented. Such functions may be used to achieve closed loop design objectives. All of the ideas presented may be extended with subtle (all be it very important)

modifications, to accommodate control system design via $\mathcal{H}^{\times}$ optimization. Additional details may be found in.

The methods presented in this chapter may be extended to discrete time linear shift invariant (LSI) systems. Extensions to sampled data systems are also possible [1].

References

1. Chen, T. and Francis, B., *Optimal Sampled-Data Control Systems,* Springer, London, 1995.
2. Dorf, R.C. and Bishop, R.H., *Modern Control Systems,* 8th ed, Addison Wesley, CA, 1998.
3. Doyle, J.C., "Guaranteed margins for LQG regulators," *IEEE Transactions on Automatic Control,* Vol. AC-23, No. 4, August 1978, pp. 756–757.
4. Doyle, J.C., Glover, K., Khargonekar, P.P., and Francis, B.A., "State-space solutions to standard $\mathcal{H}^2$ and $\mathcal{H}^{\infty}$ control problems," *IEEE Transactions on Automatic Control,* Vol. AC-34, No. 8, 1989, pp. 831–847. Also see *Proceedings of the 1988 American Control Conference,* Atlanta, Georgia, June, 1988.
5. Kalman, R.E., "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering,* Vol. 85, 1960, pp. 34–45.
6. Kalman, R.E. and Bucy, R.S., "New results in linear filtering and prediction problems," *ASME Journal of Basic Engineering,* 1960, pp. 95–108.
7. Kwakernaak, H. and Sivan, R., *Linear Optimal Control Systems,* Wiley-Interscience, New York, 1972.
8. Rodriguez, A.A., *A Practical Neo-Classical Approach to Feedback Control System Analysis and Design,* Control3D, 2000.
9. Spong, M.W. and Vidyasagar, M., *Robot Dynamics and Control,* John Wiley & Sons, New York, 1989.
10. Zhou, K., Doyle, J.C., and Glover, K., *Robust and Optimal Control,* Prentice-Hall, NJ, 1996.
11. Zhou, K. and Doyle, J.C., *Essentials of Robust Control*, Prentice-Hall, NJ, 1998.

11

Adaptive and Nonlinear Control Design

Maruthi R. Akella
The University of Texas at Austin

11.1 Introduction

The most important challenge for modern control theory is that it should deliver acceptable performance while dealing with poor models, high nonlinearities, and low-cost sensors under a large number of operating conditions. The difficulties encountered are not peculiar to any single class of systems and they appear in virtually every industrial application. Invariably, these systems contain such a large amount of model and parameter uncertainty that "fixed"controllers can no longer meet the stability and performance requirements. Any reasonable solution for such problems must be a suitable amalgamation between nonlinear control theory, adaptive elements, and information processing. Such are the factors behind the birth and evolution of the field of adaptive control theory, strongly motivated by several practical applications such as chemical process control and design of autopilots for high-performance aircraft, which operate with proven stability over a wide variety of speeds and altitudes.

A commonly accepted definition for an adaptive system is that it is any physical system that is designed from an adaptive standpoint![1] All existing stability and convergence results, in the field of adaptive control theory, hinge on the crucial assumption that the unknown parameters must occur linearly within the plant containing known nonlinearities. Conceptually, the overall process makes the parameter estimates themselves as state variables, thus enlarging the dimension of the state space for the original system. By nature, adaptive control solutions for both linear and nonlinear dynamical systems lead to nonlinear time-varying formulations wherein the estimates of the unknown parameters are updated using input–output data. A parameter adaptation mechanism (typically nonlinear) is used to update the parameters within the

control law. Given the nonlinearity due to adaptive feedback, there is the need to ensure that the closed-loop stability is preserved. It is thus an unmistakable fact that the fields of adaptive control and nonlinear system stability are intrinsically related to one another and any new insights gained in one field would be of potential benefit to the other. Many formalisms in nonlinear stability theory can be employed such as the Lyapunov direct method and passivity-based methods. We will first present some important mathematical and analytical tools for studying the stability of nonlinear dynamical systems.

11.2 Lyapunov Theory for Time-Invariant Systems

The Lyapunov direct method is a commonly adopted and arguably one of the most popular methods for proving closed-loop stability in the adaptive control area. It is not restricted to local system behavior and determines the stability properties of the nonlinear system by considering the time evolution of the system solutions with respect to an "energy-like" scalar function, often known as the Lyapunov function.

Consider any dynamical system represented by the following nonlinear autonomous differential equation

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}), \qquad f(0) = 0 \tag{11.1}$$

Obviously $\boldsymbol{x}(t) = 0$ is a solution. A sufficient condition for the existence and uniqueness of solutions for Equation 11.1 is that $f(\boldsymbol{x})$ be locally Lipschitz, that is,

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\| \tag{11.2}$$

for all $\boldsymbol{x}$ and $\boldsymbol{y}$ in a finite neighborhood of the origin. We are interested in the stability of the solutions of Equation 11.1 in the presence of perturbations. Before discussing the main Lyapunov stability theorems, we present some important definitions.

Definition: **Lyapunov stability**
The solution $\boldsymbol{x}(t) = 0$ of Equation 11.1 is called *stable* in the sense of Lyapunov if for all $\varepsilon > 0$, there exists a $\delta(\varepsilon) > 0$ such that for all initial conditions satisfying $\|\boldsymbol{x}(0)\| < \delta$, we have $\|\boldsymbol{x}(0)\| < \varepsilon$ for $t \in [0, \infty)$. The solution is *unstable* if it is not stable. The solution is *asymptotically stable* if it is stable and there exists a $\delta > 0$ such that every initial condition that satisfies $\|\boldsymbol{x}(0)\| < \delta$ has the property

$$\lim_{t\to\infty} \|\boldsymbol{x}(t)\| = 0$$

The solution is *globally asymptotically stable* if it is asymptotically stable for all initial conditions. These definitions refer to stability of particular solutions of Equation 11.1 with respect to initial conditions and not to the stability of differential equations.

Definition: **Positive definite and semidefinite functions**
Any continuously differentiable function $V: \mathbb{R}^n \to \mathbb{R}$ is called *positive definite* if (i) $V(0) = 0$ and (ii) $V(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \neq 0$. A function is *positive semidefinite* if condition (ii) is replaced by $V(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \neq 0$.

Theorem: **Lyapunov's stability theorem for time-invariant systems**
If there exists a positive definite function $V: \mathbb{R}^n \to \mathbb{R}$ such that the time derivative of V along the solution of $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ given by

$$\frac{d}{dt}V = \left[\frac{\partial V}{\partial \boldsymbol{x}}\right]^T \dot{\boldsymbol{x}} = \left[\frac{\partial V}{\partial \boldsymbol{x}}\right]^T f(\boldsymbol{x})$$

is negative semidefinite, then the solution $\boldsymbol{x}(t) = 0$ of Equation 11.1 is stable. In this case, the solution converges to the set $\{\boldsymbol{x} \in \mathbb{R}^n : \dot{V}(\boldsymbol{x}) = 0\}$. If $\dot{V}$ is negative definite, then the solution is asymptotically stable. Furthermore, if $\dot{V}$ is negative definite and $V(\boldsymbol{x}) \to \infty$ as $\|\boldsymbol{x}\| \to \infty$, then the solution is globally asymptotically stable. The function $V(\boldsymbol{x})$ is called a Lyapunov function for the system described by Equation 11.1.

Remark: Lyapunov's theorem, though simple to state, has powerful applications in the stability analysis of nonlinear systems. However, since the theorem provides only a sufficient condition in terms of the Lyapunov function, we are often encountered with the difficult problem of finding a suitable Lyapunov function. In the special case when Equation 11.1 is a stable linear system,

$$\dot{\boldsymbol{x}} = A_m \boldsymbol{x}$$

a quadratic Lyapunov function $V = \boldsymbol{x}^T P \boldsymbol{x}$ exists where P is a symmetric positive definite matrix satisfying the so-called Lyapunov equation

$$A_m^T P + P A_m = -Q \tag{11.3}$$

for any symmetric positive definite Q matrix. On the other hand, there is no general recipe for construction of Lyapunov functions for nonlinear systems. As a rule of thumb, in the case of mechanical systems, "energy-like"quantities are good candidates for a first attempt.

11.3 Lyapunov Theory for Time-Varying Systems

We are now ready to consider the stability of solutions for a time-varying (nonautonomous) differential equation

$$\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, t), \quad \boldsymbol{g}(0, t) = 0 \quad \forall t \le 0 \tag{11.4}$$

The function $\boldsymbol{g}$ is assumed to be piecewise continuous with respect to t and locally Lipschitz in $\boldsymbol{x}$ about a neighborhood of the solution $\boldsymbol{x}(t) = 0$. This would guarantee that the origin is an equilibrium for Equation 11.4. In order to investigate the stability of equilibrium for this nonautonomous system, it is important to recognize that any solution of Equation 11.4 depends not only on time t but also on the initial time t_0. Thus, we need to revisit our previous definitions of stability.

Definition: **Uniform Lyapunov stability**
The solution $\boldsymbol{x}(t) = 0$ for Equation 11.4 is *uniformly stable* if for every $\varepsilon > 0$, there exists a $\delta(\varepsilon) > 0$ that is independent of the initial time t_0 such that

$$\|\boldsymbol{x}(t_0)\| < \delta \quad \text{implies} \quad \|\boldsymbol{x}(t)\| < \varepsilon \quad \forall t \ge t_0 \ge 0$$

The solution is *uniformly asymptotically stable* if it is uniformly stable and there is a positive constant ρ independent of t_0 such that $\|\boldsymbol{x}(t)\| \to 0$ as $t \to \infty$ for all $\|\boldsymbol{x}(t_0)\| < \rho$. The solution is *globally uniformly asymptotically stable* if it is uniformly asymptotically stable for all initial conditions.

The main stability theorem for nonautonomous systems requires the definition of certain *class K functions*.

Definition: **Class K functions**
A continuous function α: $[0, a) \to [0, \infty)$ is said to belong to *class K* if it is strictly increasing and $\alpha(0) = 0$. It is said to belong to class K_∞, or radially unbounded, if $a = \infty$ in such a way that $\alpha(r) \to \infty$ as $r \to \infty$.

Theorem: **Lyapunov's stability theorem for time-varying systems**
Consider a set $D = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \le R\}$ about the equilibrium $\boldsymbol{x}(t) = 0$ for Equation 11.4. If there exists a scalar function V: $\mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$ with continuous partial derivatives such that

(i) $\alpha_1(\|\boldsymbol{x}\|) \le V(\boldsymbol{x}, t) \le \alpha_2(\|\boldsymbol{x}\|)$ *positive definite* and *decrescent*

(ii) $\dot{V} = \dfrac{\partial V}{\partial t} + \left[\dfrac{\partial V}{\partial \boldsymbol{x}}\right]^T \boldsymbol{g}(\boldsymbol{x}, t) \le \alpha_3(\|\boldsymbol{x}\|)$

for all $t \ge 0$, where α_1, α_2, and α_3 are class K functions, then the equilibrium point $\boldsymbol{x} = 0$ is uniformly asymptotically stable.

Remark: Note that in order to show stability for nonautonomous systems, it is necessary to bound the function $V(\boldsymbol{x}, t)$ by the class K functions that do not depend upon time t. A detailed treatment of all the definitions and proof for this theorem can be found in Slotine and Li[2] and Khalil.[3]

Remark: In the recent years, several interesting converse Lyapunov results have been obtained. In particular, for every uniformly stable (or uniformly asymptotic stable) system, there exists a positive definite Lyapunov function with a negative semidefinite time derivative (see Sastry and Bodson[4]). These results are particularly useful from a closed-loop performance point of view because they allow us to explicitly estimate the convergence rates in some cases of nonlinear adaptive control systems.

The application of Lyapunov's stability theorem for nonautonomous systems arising out of adaptive control often leads us to negative semidefinite time derivatives of the Lyapunov function. Therefore, asymptotic stability analysis is a much harder problem and the following result, known as Barbalat's Lemma, is extremely useful in such situations.

Lemma: **Barbalat.**
Consider a uniformly continuous function $\phi: - \rightarrow -$ defined at all real values of $t \geq 0$. If

$$\lim_{t \to \infty} \int_0^t \phi(s)\, ds$$

exists and is finite, then $\phi(t) \rightarrow 0$ as $t \rightarrow \infty$.

Remark: A consequence of this result is that if $\phi \in \mathcal{L}_2$ and $\dot{\phi} \in \mathcal{L}_\infty$, then $\phi(t) \rightarrow 0$ as $t \rightarrow \infty$ (see Slotine and Li[2] and Tao[5] for discussion and proof).

11.4 Adaptive Control Theory

In contrast to a fixed or ordinary controller, an adaptive controller is one with adjustable parameters and an adjustment mechanism. The following are some basic concepts that are necessary for any discussion of adaptive control theory.

11.4.1 Regulation and Tracking Problems

The desired objective for any control problem is to maintain the plant output either at its desired value or within specified/acceptable bounds of the desired value. If these desired values are constant with respect to time, we have a regulation problem, otherwise it is a tracking problem.

11.4.2 Certainty Equivalence Principle

This principle has been the bedrock of most adaptive control design methods and has received considerable attention during the past two decades.[4,6,7] Adaptive controllers based on this approach are obtained by independently designing a control law that meets the control objective assuming complete knowledge of all the unknown plant parameters (deterministic case), along with a parameter update law, which is usually a differential equation that generates online parameter estimates that are used to replace the unknown parameters within the control law. Such a controller would have perfect output tracking capability in the case when the plant parameters are exactly known. In the presence of parameter uncertainty, the adaptation mechanism will adjust the controller parameters so that the tracking objective is asymptotically achieved. The main issue, thus, in adaptive controller design is to synthesize the adaptation mechanism (parameter update law) that will guarantee that the control system remains stable and the output tracking error converges to zero as the parameter values are updated.

11.4.3 Direct and Indirect Adaptive Control

There exist two philosophically distinct approaches within adaptive control for plants containing unknown or uncertain parameters. The first of those is the so-called *direct approach* where the controller parameters are directly adjusted by the adaptation mechanism in such a way to optimize some pre-specified performance index based on the output. The second approach is the *indirect approach* wherein plant parameters are directly estimated and updated by the adaptation law and these estimated values are then used to compute the controller parameters. Direct adaptive control eliminates the need for this additional computation. Consequently, indirect adaptive control is plant parameter adaptive, whereas direct adaptive control is output performance adaptive. The plant parameter identification process is explicit within the direct approach while implicit in the direct approach. Hence, they have also been referred to as explicit and implicit approaches. In both of these cases, the controller structure remains the same and is determined from the certainty equivalence principle.

11.4.4 Model Reference Adaptive Control (MRAC)

The model reference adaptive control (MRAC) framework consists of four parts: (i) the plant containing the unknown parameters, (ii) a suitable reference model for specifying the desired output characteristics, (iii) a feedback control law that contains adjustable parameters, and (iv) an adaptation mechanism that updates the adjustable parameters within the control law. A schematic sketch for this framework is shown in Figure 11.1.

The plant is assumed to have a known structure with unknown parameters. For the case of linear systems, this means that the number of poles and zeros are assumed to be known, but the exact locations of poles and zeros are unknown. In the case of nonlinear systems, the structure of the governing equations of motion is assumed to be known, but some of the parameters appearing linearly within those equations can be unknown. The reference model specifies the desired output behavior expected from the plant as a result of the external reference input. It provides the ideal plant response which the adaptation mechanism should seek to track while updating the parameter estimates. The choice of the reference plant lies at the heart of any MRAC design and any acceptable selection must essentially satisfy two crucial requirements. The first of these requirements is that the reference model must accurately reflect the closed-loop performance specifications such as rise time, settling time, overshoot, and other transient performance characteristics. The other requirement is that given the assumed structure of the plant dynamics, the reference model's output behavior should be asymptotically achievable by the adaptive control system implying certain extra conditions on the relative degree of the reference model and persistent excitation conditions on the reference input. The controller structure is dictated by the certainty equivalence

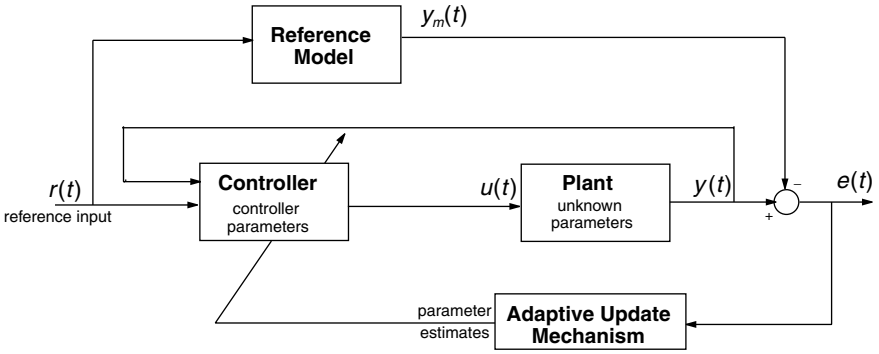


FIGURE 11.1 The model reference adaptive control framework.

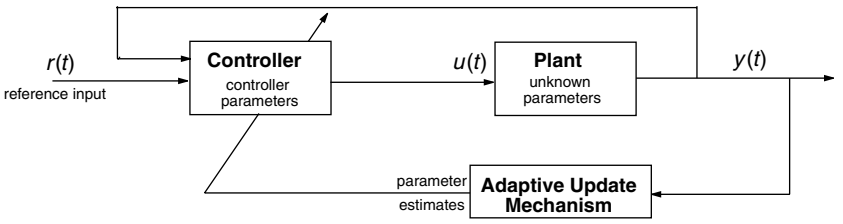


FIGURE 11.2 The self-tuning control architecture.

principle and both direct and indirect parameter update procedures can be adopted within the MRAC framework. Much of the work in this area deals with continuous time systems.

11.4.5 Self-Tuning Controller (STC)

In contrast to MRAC, there is no reference model in the self-tuning controller (STC) design. A schematic sketch is shown in Figures 11.2. In this formulation, the controller parameters of the plant parameters are estimated in real time, depending on whether it is a direct or indirect approach. These estimates are then used as if they are equal to the true parameters (certainty equivalence design). Parameter estimation involves finding the *best-fit* set of parameters based on the plant input–output data. This is different from the MRAC parameter adaptation scheme, where the parameter estimates are updated in such a way to achieve asymptotic tracking between the tracking error between the plant and the reference model. In several STC estimation schemes, it is also possible to quantify a measure of the quality of the parameter estimates, which can be used in the design of the controller. Many different combinations of the estimation methods can be adopted and can be applied to both continuous time and discrete time plants. Due to the "separation" between parameter estimation and control in STC, there is greater fiexibility in design. However, stability and convergence are difficult to prove and stronger conditions on input signals are required (persistent excitation) to guarantee parameter convergence. Historically speaking, STC designs arose in the study of the stochastic regulation problem and much of the literature is devoted to discrete time plants using an indirect approach. In spite of the seeming difference between MRAC and STC, a direct correspondence exists between problems from both the areas.[8]

11.5 Nonlinear Adaptive Control Systems

For the most general case of nonlinear systems, there exists very limited theory in the field of adaptive control. Even though there is great interest in this area due to potential applications in a wide variety of complex machanical systems, theoretical diffculties exist because of the lack of general analysis tools. However, some important special cases are well understood by now, and we summarize the conditions that these classes of systems satisfy:

1. The unknown parameters within the nonlinear plant are linearly parameterized.
2. The complete state vector is measured.
3. When the unknown parameters are assumed known, the control input can cancel all the nonlinearities in a feedback-linearization sense and any remaining internal dynamics should be stable. The adaptive design is then accomplished by certainty equivalence.

We now show a typical nonlinear MRAC methodology to deal with a situation in which the nonlinear plant model has unknown parameters. Consider the nonlinear system

$$\dot{x} = \theta f(x) + u \tag{11.5}$$

where θ is a constant and unknown matrix parameter, and f is a known and differentiable nonlinear vector function. In analogy with the MRAC methodology, we assume that it is desired to have the state x asymptotically track the state x_m of a reference system that satisfies

$$\dot{x}_m = A_m x_m + r \tag{11.6}$$

where $r(t)$ is any piecewise continuous and bounded reference input and A_m is a Hurwitz matrix. Introduce an error vector $e = x - x_m$ so that the error dynamics can be established by taking the difference between Equations 11.5 and 11.6 as follows:

$$\dot{e} = \theta f(x) - A_m x_m - r + u \tag{11.7}$$

If the parameter θ is assumed to be known, selecting the control input $u = A_m x + r - \theta f(x)$ would render the following structure for the error dynamics:

$$\dot{e} = A_m e$$

which would achieve the control objective. However, such a choice of control law is not impossible because θ is unknown. Hence we retain the same structure for the control law except for replacing θ by its time-varying estimate $\hat{\theta}$ so that the certainty-equivalence-based adaptive control law is given by

$$u = A_m x + r - \hat{\theta} f(x) \tag{11.8}$$

Application of the control law in Equation 11.7 leads us to the following closed-loop error dynamics:

$$\dot{e} = A_m e - \tilde{\theta} f(x) \tag{11.9}$$

where we have introduced the variable $\tilde{\theta}(t)$ to represent the parameter estimation error $\hat{\theta}(t) - \theta$. There are two things remaining to be done: (i) to show the stability and asymptotic convergence of $e(t)$ to zero as $t \to \infty$, (ii) to provide an appropriate parameter adaptation mechanism for $\theta(t)$. We accomplish both these tasks by adopting the Lyapunov method. Given that A_m is Hurwitz, for any choice of symmetric and positive definite matrix Q, there exists a symmetric, positive definite matrix P that satisfies the Lyapunov equation given in Equation 11.3. Choosing a Lyapunov function in terms of such a P matrix,

$$V = e^T P e + \mathrm{tr}[\tilde{\theta}^T \Gamma^{-1} \tilde{\theta}] \tag{11.10}$$

where Γ is a symmetric positive definite learning rate matrix. Taking the time derivative of V along the solutions of Equation 11.9 we find that

$$\dot{V} = e^T (P A_m + A_m^T P) e - 2 e^T P \tilde{\theta} f(x) + 2\, \mathrm{tr}[\tilde{\theta}^T \Gamma^{-1} \dot{\hat{\theta}}] \tag{11.11}$$

Using several matrix trace identities,[9] it is possible to show that

$$e^T P \tilde{\theta} f(x) = \mathrm{tr}[P \tilde{\theta} f(x) e^T] = \mathrm{tr}[\tilde{\theta} f(x) e^T P] = \mathrm{tr}[\tilde{\theta}^T P e f^T(x)]$$

so that we can combine the last two terms on the right-hand side of Equation 11.11 as follows:

$$\dot{V} = e^T \underbrace{(P A_m + A_m^T P)}_{-Q} e + 2\, \mathrm{tr}\,[\tilde{\theta}^T \{\Gamma^{-1} \dot{\hat{\theta}} - P e f^T(x)\}] \tag{11.12}$$

Since θ is constant, $\dot{\theta} = 0$ and $\dot{\hat{\theta}} = \dot{\tilde{\theta}}$. Thus, if the adaptive law for updating $\hat{\theta}$ is chosen as

$$\dot{\hat{\theta}} = \Gamma P e f^T(x) \tag{11.13}$$

then the derivative of the Lyapunov function in Equation 11.12 becomes

$$\dot{V} = -e^T Q e \tag{11.14}$$

which is negative semidefinite, but not negative definite. This implies that $V(t) \leq V(0)$ for all $t \geq 0$, and thus, e and $\hat{\theta}$ must be bounded. This further implies that $x = e + x_m$ also is bounded. Also, $V \geq 0$ and $\dot{V} \leq 0$, which means $\lim_{t\to\infty} V(t) \doteq V_\infty$ exists and is finite. Now,

$$\int_0^\infty \dot{V}(\tau)d\tau = -\int_0^\infty e^T(\tau)Qe(\tau)d\tau = V_\infty - V(0)$$

implying that $e \in \mathcal{L}_2 \cap \mathcal{L}_\infty$. From Equation 11.9, it is obvious that $\dot{e} \in \mathcal{L}_\infty$. Thus, we can invoke Barbalat's lemma to claim that $e(t) \to 0$ as $t \to \infty$. Notice, however, that the parameter error $\tilde{\theta} = \hat{\theta} - \theta$ will not necessarily converge to zero. True parameter convergence can occur only when the reference input $r(t)$ satisfies certain uniform observability and persistent excitation conditions.[4]

Remark: Note in the above MRAC design that while stability and tracking error convergence are guaranteed for any value of A_m, Q, and Γ, the performance of the controller will depend critically on the learning rate Γ. "Smaller" learning rates mean that the adaptation will be slow leading to large tracking errors and large transients. Conversely, the upper limit on the learning rate is limited by the presence of unmodeled dynamics, because too large a value for the learning rate will lead to highly oscillatory parameter estimates that can adversely excite the high frequency unmodeled plant dynamics.

Remark: The controller design methodology is based upon three crucial steps: (i) finding the appropriate controller structure in the spirit of feedback linearization, (ii) derivation of the tracking error dynamics that depend upon the parameter error terms, and (iii) finding a suitable Lyapunov function that can be used to derive the parameter update law such that the tracking error will go to zero. Determining the controller structure for the known parameter case is probably the most crucial step within any adaptive design because it turns out that adaptive feedback linearization cannot be alway applied to systems that are linearizable by feedback in the known parameter case. This happens because higher derivatives of the parameter estimates appear in the control law for systems of higher order, making difficult the application of the certainty equivalence principle. Other relatively new approaches deviate from the conventional certainty equivalence principles by adopting integrator backstepping, nonlinear damping, and tuning functions.[10] In these methods, the adaptive law estimates the unknown plant parameters directly, thereby permitting full utilization of any prior knowledge and therefore eliminating the possibility for overparameterization introduced by traditional direct MRAC methods. The design methodology and stability proof are obtained through a recursive process,[10,11] an overview for which can be obtained from Kokotovic[12] and subsequent results by his research group.

Remark: Given the fact that there always exist model errors and other unknown disturbance effects in addition to the unknown parameters, adaptive control solutions would have to address the robustness question. Since the parameter error is always unknown, the Lyapunov function time derivative is always negative semidefinite. This implies that the closed-loop equations are not exponentially stable, nor even uniformly asymptotically stable. Any external or unmodeled disturbance would immediately make $\dot{V}$ indefinite, and most methods that modify the stability proof for robustness are fixed to ensure $\dot{V}$ to be negative outside a compact neighborhood of the equilibrium state. By introducing an additional term in the adaptive law Equation 11.13 (referred to as σ-modification), Ioannou[7] accomplishes robust stability. This method, though very popular, suffered from the drawback that when the disturbance is absent, the tracking error would not converge to zero. To overcome this problem, other schemes such as the Π-modification[6] have been suggested to ensure robustness within the adaptive designs.

11.6 Spacecraft Adaptive Attitude Regulation Example

Consider the problem of a rigid spacecraft with an initial nonzero attitude and body angular velocity vector that has to be brought to rest at a zero attitude vector. This rigid body adaptive attitude regulation problem based on the feedback linearization approach has been derived by Schaub, Akella, and Junkins.[13] The governing equations are described by Euler's rotational equations of motion and the desired linear closed-loop dynamics (LCLD) can be of either PD or PID form.[13, 14] Only a crude estimate of the moment of inertia matrix is assumed to be known. An adaptive control law is presented, which includes an integral feedback term in the desired closed-loop dynamics and achieves asymptotic stability even in the presence of unmodeled external disturbances.

The resulting simulation is illustrated in Figure 11.3. The attitude vector is specified in terms of the modified Rodrigues parameter (MRP) whose components σ_i are shown in Figure 11.3a. Without any adaptation, the open-loop control is still asymptotically stable. However, the transient attitude errors don't match those of the desired LCLD well at all. With adaptation turned on, the performance matches that of the ideal LCLD very closely.

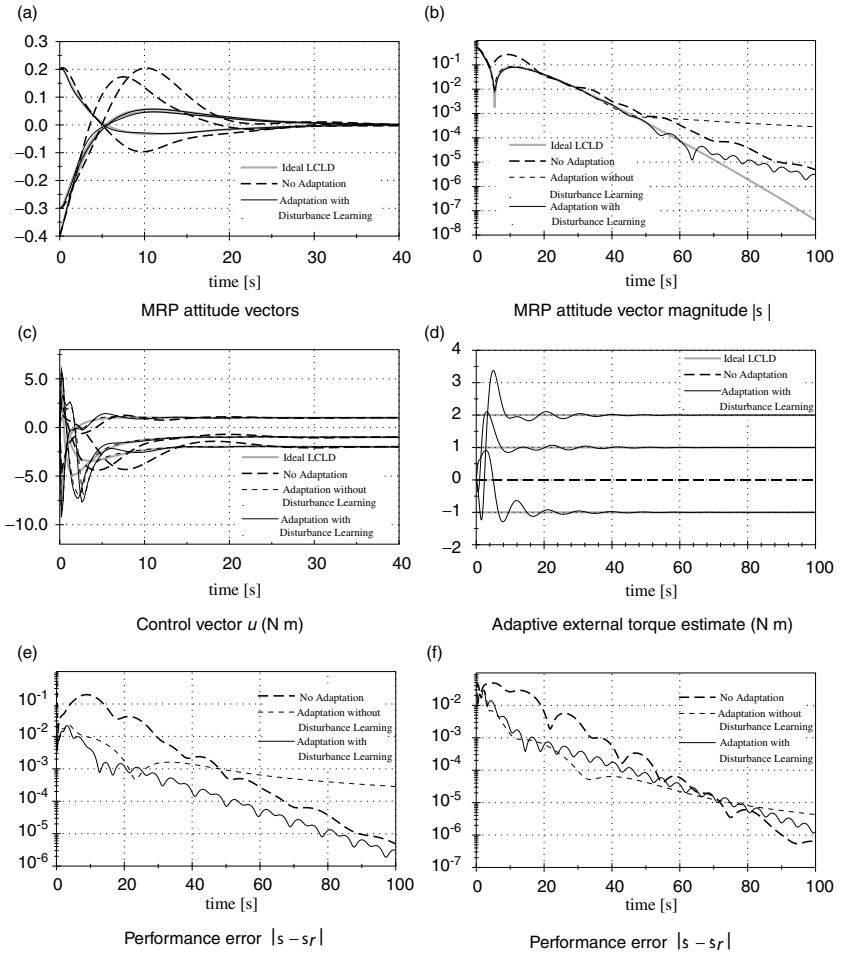


FIGURE 11.3 Rigid body stabilization while enforcing LCLD in the presence of large inertia and external disturbance ignorance.

Figure 11.3b shows the magnitude of the MRP attitude error vector σ on a logarithmic scale. Again the large transient errors of the open-loop, adaptation-free control law are visible during the first 20 s of the maneuver along with the good final convergence characteristics. The ideal LCLD performance is indicated again through the dotted line. Two versions of the adaptive control law are compared here, which differ only by whether or not the external disturbance is adaptively estimated too. On this figure both adaptive laws appear to enforce the desired LCLD very well for the first 40 s of the maneuver. After this the adaptive law without disturbance learning starts to decay at a slower rate, slower even than the open-loop (nonadaptive) solution. Including the external disturbance, adaptation clearly improves the final convergence rate. Note, however, that neither adaptive case starts to deviate from the ideal LCLD case until the MRP attitude error magnitude has decayed to roughly 10^{-3}. This corresponds to having a principal rotation error of roughly 0.23°. With external disturbance adaptation, the tracking error at which the LCLD deviations appear is about two orders of magnitude smaller.

The performance of the adaptive control law can be greatly varied by choosing different learning rates. However, since *large* initial inertia matrix and external disturbance model errors are present, the adaptive learning rates were reduced to avoid radical transient torques. The control torque vector components u_i for various cases are shown in Figure 11.3c. The open-loop torques don't approach the ideal LCLD torque during the transient part of the maneuver. The torques required by either adaptive case are very similar. The difference is that the case with external disturbance learning is causing some extra oscillation of the control about the LCLD case. However, note that with the chosen adaptive learning rates neither control law exhibits any radical transient torques about the ideal LCLD torque profile. Figure 11.3d illustrates that the adaptive external disturbance estimate $\boldsymbol{F}_e$ indeed asymptotically approaches the true external disturbance $\boldsymbol{F}_e^*$. By reducing the external disturbance adaptive learning rate γ_{F_e} the transient adaptive estimate errors are kept within a reasonable range.

Figure 11.3f shows the absolute performance error in attitude rates. Both cases with adaptation added show large reductions in attitude rate errors compared to the nonadaptive case.

The purpose of the adaptive control discussed in this example is to enforce the desired LCLD. The previous figures illustrate that the resulting overall system remains asymptotically stable. Figure 11.3e illustrates the absolute performance error between the actual motion $\sigma(t)$ and the desired linear reference motion $\sigma_r(t)$. This figure demonstrates again the large performance error that results from using the open-loop control law with the incorrect system model. Adding adaptation improves the transient performance tracking by up to two orders of magnitude. Without including the external disturbance learning, the final performance error decay rate flattens out. This error will decay to zero. However, with the given learning gains, it does so at a slower rate than if no adaptation is taking place. Adding the external disturbance learning greatly improves the final performance error decay since the system is obtaining an accurate model of the actual constant disturbance. If the initial model estimates were more accurate, more aggressive adaptive learning rates could be used, resulting in even better LCLD performance tracking. This simulation illustrates though that even in the presence of large system uncertainty it is possible to track the desired LCLD very well.

11.7 Output Feedback Adaptive Control

In contrast to the state-space approaches, the input–output approach treats the plant as a black box that transforms the applied inputs into the corresponding output space. Stability theory for nonlinear systems from an input–output viewpoint is important in the context of adaptive output feedback control design. Solution to the problem of adaptive observer design involving state estimation of systems with unknown parameters is often the stepping stone towards resolving the output feedback control problem. There has been fairly recent breakthroughs in this area where the nonlinear adaptive observer design procedure has been extended to a slightly more general case of systems where the coefficients of the unknown parameters can depend on the entire state, and not just on the measured part.[15]

To a large extent, some powerful results have been made possible by exploiting certain "passivity-like"conditions coupled with the usual persistent excitation conditions. Crucial to this discussion is the

concept of passivity, which is really an abstract representation of the idea of energy dissipation in both linear and nonlinear systems. Passive systems are most common in mechanical and electrical engineering applications. A mechanical system consisting of masses, springs, and viscous dashpots is a common example for a passive system. We now give the following definitions.

Definition: **Truncation of a signal**
Let Y be the space of real-valued functions defined on $[0, \infty)$. Let $\boldsymbol{x}$ be an element of Y. Then the *truncation* of $\boldsymbol{x}$ at some $T > 0$ is defined by

$$\boldsymbol{x}_T(t) = \begin{cases} \boldsymbol{x}(t) & \text{for } 0 \le t \le T \\ 0 & \text{for } t > T \end{cases}$$

Definition: **Extended space**
If X is a normed linear subspace of Y, then the *extended space* X_e is defined by the set

$$\{\boldsymbol{x} \in Y : \boldsymbol{x}_T \in X \text{ for some fixed } T \ge 0\}$$

The extended $\mathcal{L}_2$ space is denoted by $\mathcal{L}_{2e}$.

Definition: **Scalar product between two signals**
The scalar product between two real-valued time signals $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{L}_{2e}$ is defined as

$$\langle \boldsymbol{x} \mid \boldsymbol{y} \rangle = \int_0^\infty \boldsymbol{x}^T(\tau)\boldsymbol{y}(\tau)\, d\tau = \int_0^T \boldsymbol{x}^T(\tau)\boldsymbol{y}(\tau)\, d\tau$$

Definition: **Passive systems**
A system with input $\boldsymbol{u}(t)$ and output $\boldsymbol{y}(t)$ is *passive* if

$$\langle \boldsymbol{y} | \boldsymbol{u} \rangle \ge 0$$

The system is *input strictly passive* if $\exists\ \delta > 0$ such that

$$\langle \boldsymbol{y} | \boldsymbol{u} \rangle \ge \varepsilon \| \boldsymbol{u} \|^2$$

The system is said to be *output strictly passive* if $\exists\ \delta > 0$ such that

$$\langle \boldsymbol{y} | \boldsymbol{u} \rangle \ge \varepsilon \| \boldsymbol{y} \|^2$$

11.8 Adaptive Observers and Output Feedback Control

We now state the nonlinear adaptive observer problem formulated by Besancon:[15]

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) + \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t)\,\theta \\ \boldsymbol{y} &= \boldsymbol{h}(\boldsymbol{x}) \end{aligned} \tag{11.15}$$

where functions $\boldsymbol{f}$ and $\boldsymbol{g}$ are C^∞ with respect to all their arguments and θ is a constant and unknown parameter. Variables $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{y}$ respectively denote the state, input, and output vectors. The input signals may be assumed to belong to some set of measurable and bounded functions. By the phrase adaptive observer, we imply the problem of reconstructing a state estimate $\hat{\boldsymbol{x}}(t)$ using the input $\boldsymbol{u}$ and output $\boldsymbol{y}$

in the presence of the unknown parameter θ such that $\lim_{t\to\infty}\|\hat{\boldsymbol{x}}(t) - \boldsymbol{x}(t)\| = 0$. The conditions for the existence of such an observer are now available,[15] which can be stated as follows. If a corresponding observer exists in the case when θ is known, and if this deterministic case observer is such that when a parameter error $\tilde{\theta} \doteq \hat{\theta} - \theta$ is made and the state estimation error system is passive between the "input" $\dot{\tilde{\theta}}$ and the output error $\boldsymbol{h}(\hat{\boldsymbol{x}}) - \boldsymbol{y}$, then an asymptotic state observer can be designed even when θ is unknown. In addition to this passivity requirement, parameter error convergence, as usual, would further need persistence of excitation with respect to $\boldsymbol{u}$.

This powerful result finds immediate applications within the problem of spacecraft attitude tracking in the absence of angular velocity measurements.[16] It is now well known that the governing equations of the rigid-body attitude control problem in terms of the MRP vector satisfy certain passivity conditions[17, 18] between the angular velocity vector and the MRP vector. A very important consequence of passivity in this context is the fact that feedback control laws for attitude control can be implemented in a Lyapunov-based construction without requiring angular velocity measurements. In such a case, the only signal needed for feedback purposes would be the attitude vector. The resulting control laws provide *almost* global asymptotic stability in the sense of Tsiotras.[18]

11.9 Concluding Remarks

Historically speaking, the development and application of modern adaptive control theory for generic nonlinear systems adopted the philosophical approach of extending existing linear system methodologies. In some limited cases such as regulator theory, this approach of paralleling linear system methods has been highly successful. However, obtaining the same degree of success has been elusive in other research areas such as trajectory tracking, controller synthesis, and state reconstruction.

It is not difficult to fathom the reason for this bottleneck. Nonlinear systems occur in a vast variety of ways, and not all of them can be handled by simple extensions to existing linear adaptive control methodologies. One promising approach for the purpose of future research would be to specialize the study to mechanical systems, thereby restricting the class of nonlinear systems considered, and thus enabling the introduction of "structure" and additional constraints. Whereas in the case of output feedback control for general nonlinear systems, separate designs of stable observers and controllers do not necessarily guarantee stability for their combination (no separation principle), some structured approaches utilizing state transformations have already been shown to help recover the separation properties in some cases.[15] As a result, these so-called structured approaches also enabled the formulation of global and semi-global tracking controllers based on output (partial state) feedback. It is quite possible that a focused pursuit of the same approach has the potential for providing a key to solving several other problems arising out of electromechanical systems that are otherwise intractable.

References

1. Narendra, K. S., "Parameter adaptive control—The End … or The Beginning?," *Proceedings of the 33rd Conference on Decision and Control.* Lake Buena Vista, Florida, December 1994.
2. Slotine, J. E. and Li, W., *Applied Nonlinear Control.* Prentice-Hall, Englewood Cliffs, NJ, 1991.
3. Khalil, H. K., *Nonlinear Systems.* Macmillan, New York, NY, 1992.
4. Sastry, S. and Bodson, M., *Adaptive Control: Stability, Convergence and Robustness.* Prentice-Hall, 1989.
5. Tao, G., "A simple alternative proof to the Barbalat Lemma," *IEEE Transactions on Automatic Control,* Vol. 42, No. 5, May 1997, p. 698.
6. Narendra, K. S. and Annaswamy, A. M., *Stable Adaptive Systems.* Prentice-Hall, 1989.
7. Ioannou, P. A. and Sun, J., *Stable and Robust Adaptive Control.* Prentice-Hall, Upper Saddle River, NJ, 1995, pp. 85–134.
8. Astrom, K. J. and Wittenmark, B., *Adaptive Control.* Addison-Wesley, Reading, MA, 1995.
9. Gantmacher. *The Theory of Matrices,* Vol. I. Chelsea Publishing Company, NY, 1977, pp. 353–354.

10. Krstić, M., Kanellakopoulos, I., and Kokotović, P. V., "Transient performance improvement with a new class of adaptive controllers," *Systems & Control Letters,* Vol. 21, 1993, pp. 451–461.
11. Krtić, M., Kanellakopoulos, I., and Kokotović, P. V., "Nonlinear design of adaptive controllers for linear systems," *IEEE Transactions on Automatic Control,* Vol. 39, 1994, pp. 738–752.
12. Kokotovic, P. V., "The joy of feedback: nonlinear and adaptive control," *IEEE Control Systems Magazine,* Vol. 12, No. 3, 1992, pp. 7–17.
13. Schaub, H., Akella, M. R., and Junkins, J. L., "Adaptive control of nonlinear attitude motions realizing linear closed loop dynamics," *Journal of Guidance, Control and Dynamics,* Vol. 24, No. 1, Jan.–Feb. 2001.
14. Akella, M. R., Schaub, H., and Junkins, J. L., "Adaptive realization of linear closed loop tracking dynamics in the presence of large system model errors," *Journal of Astronautical Sciences,* Vol. 48, No. 4, 2000.
15. Besançon, G., "Global output feedback tracking control for a class of Lagrangian systems," *Automatica,* Vol. 36, 2000, pp. 1915–1921.
16. Akella, M. R., "Rigid body attitude tracking without angular velocity feedback," *Systems & Control Letters,* Vol. 42, No. 4, 2001.
17. Lizarralde, F. and Wen, J. T., "Attitude control without angular velocity measurement: a passivity approach," *IEEE Transactions on Automatic Control,* Vol. 41, No. 3, 1996, pp. 468–472.
18. Tsiotras, P., "Further passivity results for the attitude control problem," *IEEE Transactions on Automatic Control,* Vol. 43, No. 11, 1998, pp. 1597–1600.

12

Neural Networks and Fuzzy Systems

Bogdan M. Wilamowski
Auburn University

12.1 Neural Networks and Fuzzy Systems

New and better electronic devices have inspired researchers to build intelligent machines operating in a fashion similar to the human nervous system. Fascination with this goal started when McCulloch and Pitts (1943) developed their model of an elementary computing neuron and when Hebb (1949) introduced his *learning rules*. A decade later Rosenblatt (1958) introduced the **perceptron** concept. In the early 1960s Widrow and Holf (1960, 1962) developed intelligent systems such as ADALINE and MADALINE. Nillson (1965) in his book *Learning Machines* summarized many developments of that time. The publication of the Mynsky and Paper (1969) book, with some discouraging results, stopped for some time the fascination with artificial neural networks, and achievements in the mathematical foundation of the **backpropagation** algorithm by Werbos (1974) went unnoticed. The current rapid growth in the area of neural networks started with the Hopfield (1982, 1984) recurrent network, Kohonen (1982) unsupervised training algorithms, and a description of the backpropagation algorithm by Rumelhart et al. (1986).

12.2 Neuron Cell

A biological neuron is a complicated structure, which receives trains of pulses on hundreds of *excitatory* and *inhibitory* inputs. Those incoming pulses are summed with different weights (averaged) during the time period of *latent summation*. If the summed value is higher than a threshold, then the neuron itself is generating a pulse, which is sent to neighboring neurons. Because incoming pulses are summed with time, the neuron generates a pulse train with a higher frequency for higher positive excitation. In other words, if the value of the summed weighted inputs is higher, the neuron generates pulses more frequently. At the same time, each neuron is characterized by the nonexcitability for a certain time after the firing pulse. This so-called *refractory period* can be more accurately described as a phenomenon where after excitation the threshold value increases to a very high value and then decreases gradually with a certain time constant. The refractory period sets soft upper limits on the frequency of the output pulse train. In the biological neuron, information is sent in the form of frequency modulated pulse trains.

This description of neuron action leads to a very complex neuron model, which is not practical. McCulloch and Pitts (1943) show that even with a very simple neuron model, it is possible to build logic and memory circuits. Furthermore, these simple neurons with thresholds are usually more powerful than typical logic gates used in computers. The McCulloch–Pitts neuron model assumes that incoming and outgoing signals may have only binary values 0 and 1. If incoming signals summed through positive or negative weights have a value larger than threshold, then the neuron output is set to 1. Otherwise, it is set to 0.

$$T = \begin{cases} 1, & \text{if } net \geq T \\ 0, & \text{if } net < T \end{cases} \tag{12.1}$$

where T is the threshold and *net* value is the weighted sum of all incoming signals:

$$net = \sum_{i=1}^{n} w_i x_i \tag{12.2}$$

Examples of McCulloch–Pitts neurons realizing OR, AND, NOT, and MEMORY operations are shown in Figure 12.1. Note that the structure of OR and AND gates can be identical. With the same structure, other logic functions can be realized, as Figure 12.2 shows.

The perceptron model has a similar structure. Its input signals, the weights, and the thresholds could have any positive or negative values. Usually, instead of using variable threshold, one additional constant input with a negative or positive weight can added to each neuron, as Figure 12.3 shows. In this case,

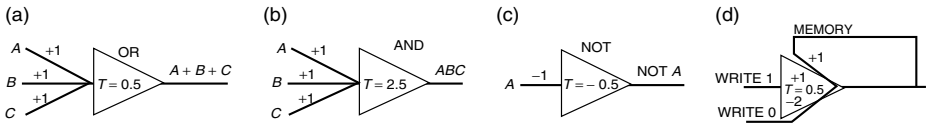


FIGURE 12.1 OR, AND, NOT, and MEMORY operations using networks with McCulloch–Pitts neuron model.

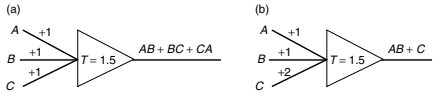


FIGURE 12.2 Other logic function realized with McCulloch–Pitts neuron model.

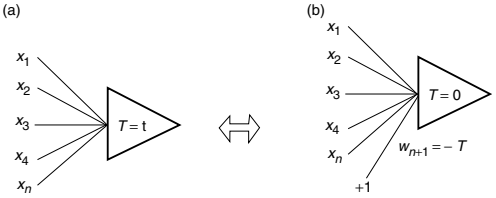


FIGURE 12.3 Threshold implementation with an additional weight and constant input with +1 value: (a) neuron with threshold T, (b) modified neuron with threshold $T = 0$ and additional weight equal to $-T$.

the threshold is always set to be zero and the net value is calculated as

$$net = \sum_{i=1}^{n} w_i x_i + w_{n+1} \tag{12.3}$$

where w_{n+1} has the same value as the required threshold and the opposite sign. Single-layer perceptrons were successfully used to solve many pattern classification problems. The hard threshold activation functions are given by

$$o = f(net) = \frac{\mathrm{sgn}(net) + 1}{2} = \begin{cases} 1, & \text{if } net \geq 0 \\ 0, & \text{if } net < 0 \end{cases} \tag{12.4}$$

for **unipolar** neurons and

$$o = f(net) = \mathrm{sgn}(net) = \begin{cases} 1, & \text{if } net \geq 0 \\ -1, & \text{if } net < 0 \end{cases} \tag{12.5}$$

for **bipolar** neurons. For these types of neurons, most of the known training algorithms are able to adjust weights only in single-layer networks.

Multilayer neural networks usually use continuous activation functions, either unipolar

$$o = f(net) = \frac{1}{1 + \exp(-\lambda net)} \tag{12.6}$$

or bipolar

$$o = f(net) = \tanh(0.5\lambda net) = \frac{2}{1 + \exp(-\lambda net)} - 1 \tag{12.7}$$

These continuous activation functions allow for the gradient-based training of multilayer networks. Typical activation functions are shown in Figure 12.4. In the case when neurons with additional threshold input are used (Figure 12.3b), the λ parameter can be eliminated from Equations 12.6 and 12.7 and the steepness of the neuron response can be controlled by the weight scaling only. Therefore, there is no real need to use neurons with variable gains.

Note, that even neuron models with continuous activation functions are far from an actual biological neuron, which operates with frequency modulated pulse trains.

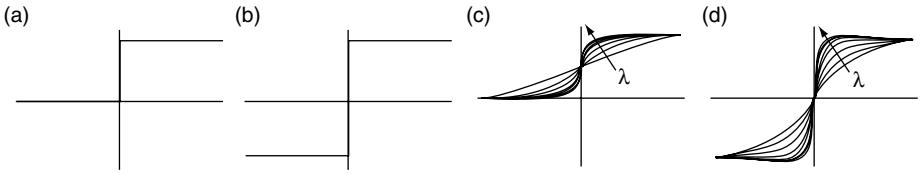


FIGURE 12.4 Typical activation functions: (a) hard threshold unipolar, (b) hard threshold bipolar, (c) continuous unipolar, (d) continuous bipolar.

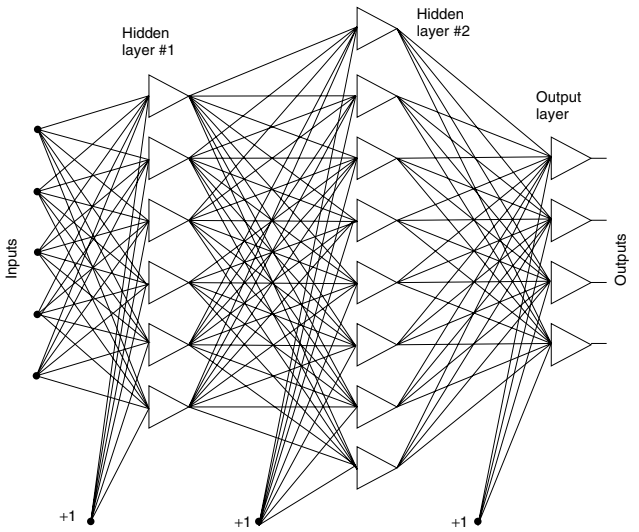


FIGURE 12.5 An example of the three-layer feedforward neural network, which is sometimes known also as the backpropagation network.

12.3 Feedforward Neural Networks

Feedforward neural networks allow only one-directional signal flow. Furthermore, most feedforward neural networks are organized in layers. An example of the three-layer feedforward neural network is shown in Figure 12.5. This network consists of input nodes, two *hidden layers*, and an output layer.

A single neuron is capable of separating input patterns into two categories, and this separation is linear. For example, for the patterns shown in Figure 12.6, the separation line is crossing x_1 and x_2 axes at points x_{10} and x_{20}. This separation can be achieved with a neuron having the following weights: $w_1 = 1/x_{10}$, $w_2 = 1/x_{20}$, and $w_3 = -1$. In general for n dimensions, the weights are

$$w_i = \frac{1}{x_{i0}} \quad \text{for } w_{n+1} = -1 \tag{12.8}$$

One neuron can divide only linearly separated patterns. To select just one region in n-dimensional input space, more than $n + 1$ neurons should be used. If more input clusters are to be selected, then the number of neurons in the input (hidden) layer should be properly multiplied. If the number of neurons in the input (hidden) layer is not limited, then all classification problems can be solved using the three-layer network. An example of such a neural network, classifying three clusters in the two-dimensional space, is shown in Figure 12.7. Neurons in the first hidden layer create the separation lines between input clusters.

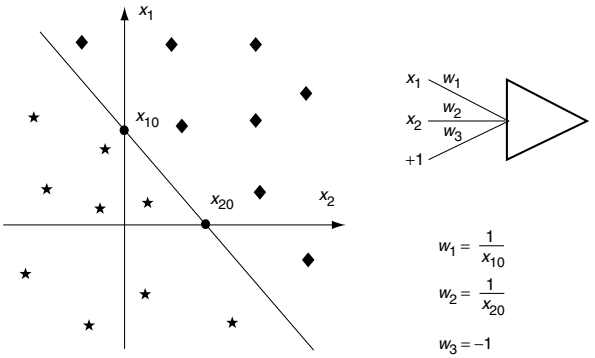


FIGURE 12.6 Illustration of the property of linear separation of patterns in the two-dimensional space by a single neuron.

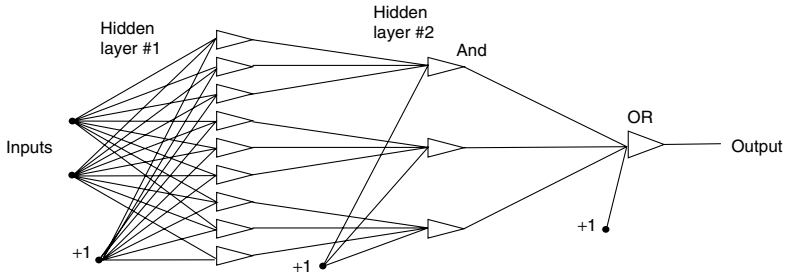


FIGURE 12.7 An example of the three-layer neural network with two inputs for classification of three different clusters into one category. This network can be generalized and can be used for solution of all classification problems.

Neurons in the second hidden layer perform the AND operation, as shown in Figure 12.1b. Output neurons perform the OR operation, as shown in Figure 12.1a, for each category. The linear separation property of neurons makes some problems especially difficult for neural networks, such as exclusive OR, parity computation for several bits, or to separate patterns laying on two neighboring spirals.

The feedforward neural network is also used for nonlinear transformation (mapping) of a multidimensional input variable into another multidimensional variable in the output. In theory, any input–output mapping should be possible if the neural network has enough neurons in hidden layers. (size of output layer is set by the number of outputs required). In practice, this is not an easy task. Presently, there is no satisfactory method to define how many neurons should be used in hidden layers. Usually, this is found by the trial-and-error method. In general, it is known that if more neurons are used, more complicated shapes can be mapped. On the other hand, networks with large numbers of neurons lose their ability for generalization, and it is more likely that such networks will also try to map noise supplied to the input.

12.4 Learning Algorithms for Neural Networks

Similarly to the biological neurons, the weights in artificial neurons are adjusted during a training procedure. Various learning algorithms were developed, and only a few are suitable for multilayer neuron networks. Some use only local signals in the neurons, others require information from outputs; some require a supervisor who knows what outputs should be for the given patterns, and other unsupervised algorithms need no such information. Common learning rules are described in the following sections.

12.4.1 Hebbian Learning Rule

The Hebb (1949) learning rule is based on the assumption that if two neighbor neurons must be activated and deactivated at the same time, then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease. If there is no signal correlation, the weight should remain unchanged. This assumption can be described by the formula

$$\Delta w_{ij} = c x_i o_j \tag{12.9}$$

where

w_{ij} = weight from ith to jth neuron,
c = learning constant,
x_i = signal on the ith input,
o_j = output signal.

The training process starts usually with values of all weights set to zero. This learning rule can be used for both soft and hard threshold neurons. Since desired responses of neurons are not used in the learning procedure, this is the **unsupervised learning** rule. The absolute values of the weights are usually proportional to the learning time, which is undesired.

12.4.2 Correlation Learning Rule

The correlation learning rule is based on a similar principle as the Hebbian learning rule. It assumes that weights between simultaneously responding neurons should be largely positive, and weights between neurons with opposite reaction should be largely negative. Contrary to the Hebbian rule, the correlation rule is the **supervised learning**. Instead of actual response, o_j, the desired response, d_j, is used for the weight change calculation

$$\Delta w_{ij} = c x_i d_j \tag{12.10}$$

This training algorithm usually starts with initialization of weights to zero values.

12.4.3 Instar Learning Rule

If input vectors and weights are normalized, or they have only binary bipolar values (−1 or +1), then the *net* value will have the largest positive value when the weights and the input signals are the same. Therefore, weights should be changed only if they are different from the signals

$$\Delta w_i = c(x_i - w_i) \tag{12.11}$$

Note, that the information required for the weight is taken only from the input signals. This is a very local and unsupervised learning algorithm.

12.4.4 Winner Takes All (WTA)

The winner takes all (WTA) is a modification of the instar algorithm where weights are modified only for the neuron with the highest *net* value. Weights of remaining neurons are left unchanged. Sometimes this algorithm is modified in such a way that a few neurons with the highest net values are modified at the same time. Although this is an unsupervised algorithm because we do not know what are desired outputs, there is a need for a "judge" or "supervisor" to find a winner with a largest net value. The WTA algorithm, developed by Kohonen (1982), is often used for automatic clustering and for extracting statistical properties of input data.

12.4.5 Outstar Learning Rule

In the outstar learning rule, it is required that weights connected to a certain node should be equal to the desired outputs for the neurons connected through those weights

$$\Delta w_{ij} = c(d_j - w_{ij}) \tag{12.12}$$

where d_j is the desired neuron output and c is the small learning constant, which further decreases during the learning procedure. This is the supervised training procedure because desired outputs must be known. Both instar and outstar learning rules were developed by Grossberg (1969).

12.4.6 Widrow–Hoff LMS Learning Rule

Widrow and Hoff (1960, 1962) developed a supervised training algorithm, which allows training a neuron for the desired response. This rule was derived so the square of the difference between the net and output value is minimized.

$$Error_j = \sum_{p=1}^{P} (net_{jp} - d_{jp})^2 \tag{12.13}$$

where

$Error_j$ = error for jth neuron,
P = number of applied patterns,
d_{jp} = desired output for jth neuron when pth pattern is applied,
net = given by Equation 12.2.

This rule is also known as the least mean square (LMS) rule. By calculating a derivative of Equation 12.13 with respect to w_{ij}, a formula for the weight change can be found:

$$\Delta w_{ij} = cx_i \sum_{p=1}^{P} (d_{jp} - net_{jp}) \tag{12.14}$$

Note that weight change Δw_{ij} is a sum of the changes from each of the individual applied patterns. Therefore, it is possible to correct the weight after each individual pattern was applied. This process is known as *incremental updating; cumulative updating* is when weights are changed after all patterns have been applied. Incremental updating usually leads to a solution faster, but it is sensitive to the order in which patterns are applied. If the learning constant c is chosen to be small, then both methods give the same result. The LMS rule works well for all types of activation functions. This rule tries to enforce the *net* value to be equal to desired value. Sometimes this is not what the oberver is looking for. It is usually not important what the *net* value is, but it is important if the *net* value is positive or negative. For example, a very large *net* value with a proper sign will result in correct output and in large error as defined by Equation 12.13 and this may be the preferred solution.

12.4.7 Linear Regression

The LMS learning rule requires hundreds or thousands of iterations, using formula 12.14, before it converges to the proper solution. Using the linear regression rule, the same result can be obtained in only one step.

Considering one neuron and using vector notation for a set of the input patterns **X** applied through weight vector **w**, the vector of *net* values **net** is calculated using

$$\mathbf{Xw} = \mathbf{net} \tag{12.15}$$

where

$\mathbf{X}$ = rectangular array $(n+1) \times p$,
n = number of inputs,
p = number of patterns.

Note that the size of the input patterns is always augmented by one, and this additional weight is responsible for the threshold (see Figure 12.3b). This method, similar to the LMS rule, assumes a linear activation function, and so the *net* values **net** should be equal to desired output values **d**

$$\mathbf{Xw} = \mathbf{d} \tag{12.16}$$

Usually $p > n + 1$, and the preceding equation can be solved only in the least mean square error sense. Using the vector arithmetic, the solution is given by

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{d} \tag{12.17}$$

When traditional method is used, the set of p equations with $n + 1$ unknowns, Equation 12.16, has to be converted to the set of $n + 1$ equations with $n + 1$ unknowns

$$\mathbf{Yw} = \mathbf{z} \tag{12.18}$$

where elements of the **Y** matrix and the **z** vector are given by

$$y_{ij} = \sum_{p=1}^{P} x_{ip}x_{jp}, \qquad z_i = \sum_{p=1}^{P} x_{ip}d_p \tag{12.19}$$

Weights are given by Equation 12.17 or they can be obtained by a solution of Equation 12.18.

12.4.8 Delta Learning Rule

The LMS method assumes linear activation function **net** = **0**, and the obtained solution is sometimes far from optimum, as is shown in Figure 12.8 for a simple two-dimensional case, with four patterns belonging to two categories. In the solution obtained using the LMS algorithm, one pattern is misclassified. If error

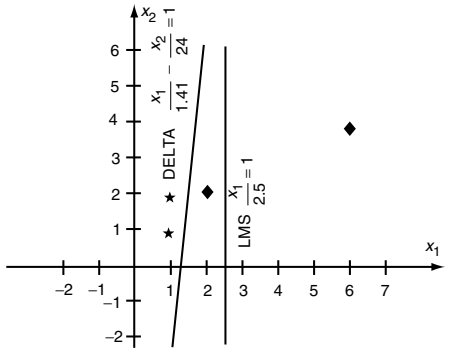


FIGURE 12.8 An example with a comparison of results obtained using LMS and delta training algorithms. Note that LMS is not able to find the proper solution.

is defined as

$$Error_j = \sum_{p=1}^{P} (o_{jp} - d_{jp})^2 \tag{12.20}$$

then the derivative of the error with respect to the weight w_{ij} is

$$\frac{d\ Error_j}{dw_{ij}} = 2\sum_{p=1}^{P} (o_{jp} - d_{jp}) \frac{df(net_{jp})}{d\ net_{jp}} x_i \tag{12.21}$$

since $o = f(net)$ and the *net* is given by Equation 12.2. Note that this derivative is proportional to the derivative of the activation function $f'(net)$. Thus, this type of approach is possible only for continuous activation functions and this method cannot be used with hard activation functions (12.4) and (12.5). In this respect the LMS method is more general. The derivatives' most common continuous activation functions are

$$f' = o(1 - o) \tag{12.22}$$

for the unipolar, Equation 12.6, and

$$f' = 0.5(1 - o^2) \tag{12.23}$$

for the bipolar, Equation 12.7.

Using the cumulative approach, the neuron weight w_{ij} should be changed with a direction of gradient

$$\Delta w_{ij} = cx_i \sum_{p=1}^{P} (d_{jp} - o_{jp}) f'_{jp} \tag{12.24}$$

In case of the incremental training for each applied pattern

$$\Delta w_{ij} = cx_i f'_j (d_j - o_j) \tag{12.25}$$

the weight change should be proportional to input signal x_i, to the difference between desired and actual outputs $d_{jp} - o_{jp}$, and to the derivative of the activation function f'_{jp}. Similar to the LMS rule, weights can be updated in both the incremental and the cumulative methods. In comparison to the LMS rule, the delta rule always leads to a solution close to the optimum. As it is illustrated in Figure 12.8, when the delta rule is used, all four patterns are classified correctly.

12.4.9 Error Backpropagation Learning

The delta learning rule can be generalized for multilayer networks. Using an approach similiar to the delta rule, the gradient of the global error can be computed with respect to each weight in the network. Interestingly,

$$\Delta w_{ij} = cx_i f'_j E_j \tag{12.26}$$

where

c = learning constant,
x_i = signal on the ith neuron input,
f'_j = derivative of activation function.

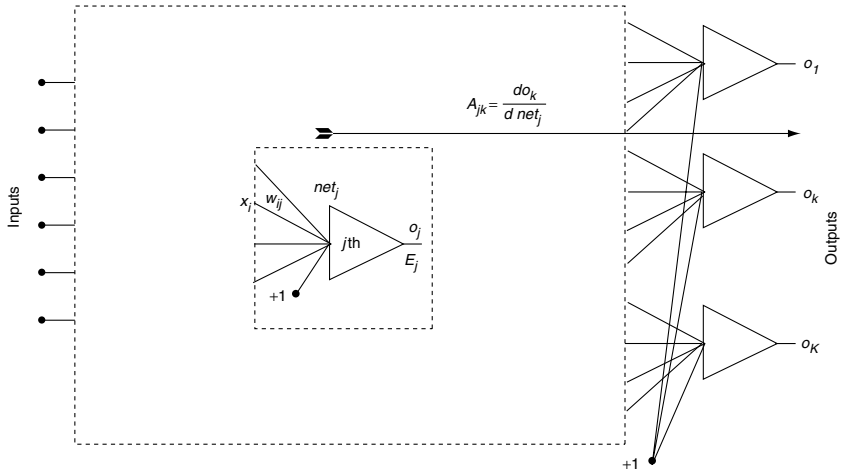


FIGURE 12.9 Illustration of the concept of gain computation in neural networks.

The cumulative error E_j on neuron output is given by

$$E_j = \frac{1}{f_j'} \sum_{k=1}^{K} (o_k - d_k) A_{jk} \tag{12.27}$$

where K is the number of network outputs and A_{jk} is the small signal gain from the input of the jth neuron to the kth network output, as Figure 12.9 shows. The calculation of the backpropagating error starts at the output layer and cumulative errors are calculated layer by layer to the input layer. This approach is not practical from the point of view of hardware realization. Instead, it is simpler to find signal gains from the input of the jth neuron to each of the network outputs (Figure 12.9). In this case, weights are corrected using

$$\Delta w_{ij} = c x_i \sum_{k=1}^{K} (o_k - d_k) A_{jk} \tag{12.28}$$

Note that this formula is general, regardless of whether the neurons are arranged in layers or not. One way to find gains A_{jk} is to introduce an incremental change on the input of the jth neuron and observe the change in the kth network output. This procedure requires only forward signal propagation, and it is easy to implement in a hardware realization. Another possible way is to calculate gains through each layer and then find the total gains as products of layer gains. This procedure is equally or less computationally intensive than a calculation of cumulative errors in the error backpropagation algorithm.

The backpropagation algorithm has a tendency for oscillation. To smooth the process, the weights increment Δw_{ij} can be modified according to Rumelhart et al. (1986):

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) + \alpha \Delta w_{ij}(n-1) \tag{12.29}$$

or according to Sejnowski and Rosenberg (1987),

$$w_{ij}(n+1) = w_{ij}(n) + (1-\alpha)\Delta w_{ij}(n) + \alpha \Delta w_{ij}(n-1) \tag{12.30}$$

where α is the momentum term.

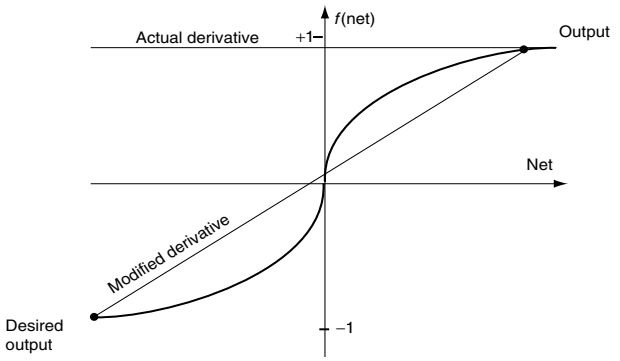


FIGURE 12.10 Illustration of the modified derivative calculation for faster convergency of the error backpropagation algorithm.

The backpropagation algorithm can be significantly sped up, when, after finding components of the gradient, weights are modified along the gradient direction until a minimum is reached. This process can be carried on without the necessity of a computationally intensive gradient calculation at each step. The new gradient components are calculated once a minimum is obtained in the direction of the previous gradient. This process is only possible for cumulative weight adjustment. One method of finding a minimum along the gradient direction is the *tree step process* of finding error for three points along gradient direction and then, using a parabola approximation, jump directly to the minimum. The fast learning algorithm using the described approach was proposed by Fahlman (1988) and is known as the *quickprop*.

The backpropagation algorithm has many disadvantages, which lead to very slow convergency. One of the most painful is that in the backpropagation algorithm, the learning process almost perishes for neurons responding with the maximally wrong answer. For example, if the value on the neuron output is close to +1 and desired output should be close to −1, then the neuron gain $f'(net) \approx 0$ and the error signal cannot backpropagate, and so the learning procedure is not effective. To overcome this difficulty, a modified method for derivative calculation was introduced by Wilamowski and Torvik (1993). The derivative is calculated as the slope of a line connecting the point of the output value with the point of the desired value, as shown in Figure 12.10.

$$f_{\text{modif}} = \frac{o_{\text{desired}} - o_{\text{actual}}}{net_{\text{desired}} - net_{\text{actual}}} \tag{12.31}$$

Note that for small errors, Equation 12.31 converges to the derivative of activation function at the point of the output value. With an increase of system dimensionality, the chances for local minima decrease. It is believed that the described phenomenon, rather than a trapping in local minima, is responsible for convergency problems in the error backpropagation algorithm.

12.5 Special Feedforward Networks

The multilayer backpropagation network, as shown in Figure 12.5, is a commonly used feedforward network. This network consists of neurons with the sigmoid type continuous activation function presented in Figures 12.4c and 12.4d. In most cases, only the one hidden layer is required, and the number of neurons in the hidden layer are chosen to be proportional to the problem complexity. The number of neurons in the hidden layer is usually found by a trial-and-error process. The training process starts with all weights randomized to small values, and the error backpropagation algorithm is used to find a solution. When the learning process does not converge, the training is repeated with a new set of randomly chosen weights.

Nguyen and Widrow (1990) proposed an experimental approach for the two-layer network weight initialization. In the second layer, weights are randomly chosen in the range from −0.5 to +0.5. In the first layer, initial weights are calculated from

$$w_{ij} = \frac{\beta z_{ij}}{\|z_j\|}, \qquad w_{(n+1)j} = \text{random}(-\beta, +\beta) \tag{12.32}$$

where z_{ij} is the random number from −0.5 to +0.5 and the scaling factor β is given by

$$\beta = 0.7P^{1/N} \tag{12.33}$$

where n is the number of inputs and N is the number of hidden neurons in the first layer. This type of weight initialization usually leads to faster solutions.

For adequate solutions with backpropagation networks, typically many tries are required with different network structures and different initial random weights. It is important that the trained network gains a generalization property. This means that the trained network also should be able to handle correctly patterns that were not used for training. Therefore, in the training procedure, often some data are removed from the training patterns and then these patterns are used for verification. The results with backpropagation networks often depend on luck. This encouraged researchers to develop feedforward networks, which can be more reliable. Some of those networks are described in the following sections.

12.5.1 Functional Link Network

One-layer neural networks are relatively easy to train, but these networks can solve only linearly separated problems. One possible solution for nonlinear problems was presented by Nilsson (1965) and was then elaborated by Pao (1989) using the functional link network shown in Figure 12.11. Using nonlinear terms with initially determined functions, the actual number of inputs supplied to the one-layer neural network is increased. In the simplest case, nonlinear elements are higher order terms of input patterns. Note that the functional link network can be treated as a one-layer network, where additional input data are generated off-line using nonlinear transformations. The learning procedure for one-layer is easy and fast. Figure 12.12 shows an *X*OR problem solved using functional link networks. Note that when the functional link approach is used, this difficult problem becomes a trivial one. The problem with the functional link network is that proper selection of nonlinear elements is not an easy task. In many practical cases, however, it is not difficult to predict what kind of transformation of input data may linearize the problem, and so the functional link approach can be used.

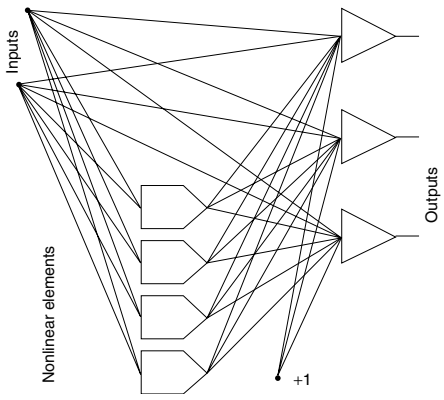


FIGURE 12.11 The functional link network.

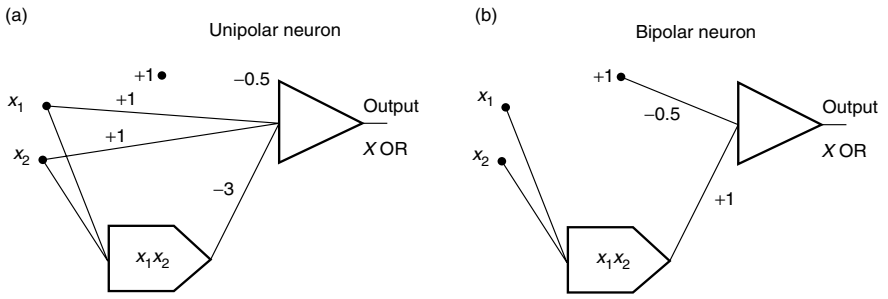


FIGURE 12.12 Functional link networks for solution of the *X* OR problem: (a) using unipolar signals, (b) using bipolar signals.

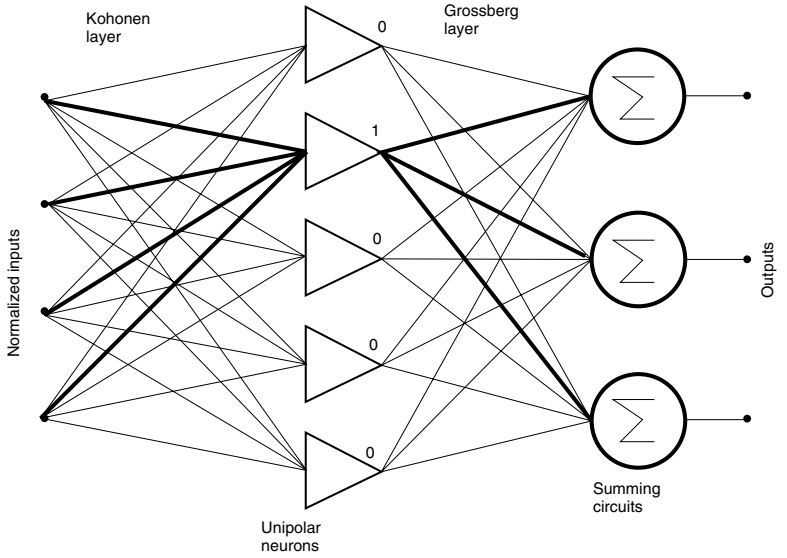


FIGURE 12.13 The counterpropagation network.

12.5.2 Feedforward Version of the Counterpropagation Network

The *counterpropagation network* was originally proposed by Hecht-Nilsen (1987). In this section a modified feedforward version as described by Zurada (1992) is discussed. This network, which is shown in Figure 12.13, requires numbers of hidden neurons equal to the number of input patterns, or more exactly, to the number of input clusters. The first layer is known as the Kohonen layer with unipolar neurons. In this layer only one neuron, the winner, can be active. The second is the Grossberg outstar layer. The Kohonen layer can be trained in the unsupervised mode, but that need not be the case. When binary input patterns are considered, the input weights must be exactly equal to the input patterns. In this case,

$$net = \mathbf{x}^t\mathbf{w} = [n - 2HD(\mathbf{x}, \mathbf{w})] \tag{12.34}$$

where

n = number of inputs,
$\mathbf{w}$ = weights,
$\mathbf{x}$ = input vector,
$HD(\mathbf{w}, \mathbf{x})$ = *Hamming distance* between input pattern and weights.

For a neuron in the input layer to be reacting just for the stored pattern, the threshold value for this neuron should be

$$w_{(n+1)} = -(n-1) \tag{12.35}$$

If it is required that the neuron must also react for similar patterns, then the threshold should be set to $w_{n+1} = -[n-(1 + \text{HD})]$, where HD is the Hamming distance defining the range of similarity. Since for a given input pattern only one neuron in the first layer may have the value of 1 and remaining neurons have 0 values, the weights in the output layer are equal to the required output pattern.

The network, with unipolar activation functions in the first layer, works as a lookup table. When the linear activation function (or no activation function at all) is used in the second layer, then the network also can be considered as an analog memory. For the address applied to the input as a binary vector, the stored set of analog values, as weights in the second layer, can be accurately recovered. The feedforward counterpropagation network may also use analog inputs, but in this case all input data should be normalized,

$$\mathbf{w}_i = \hat{\mathbf{x}}_i = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|} \tag{12.36}$$

The counterpropagation network is very easy to design. The number of neurons in the hidden layer is equal to the number of patterns (clusters). The weights in the input layer are equal to the input patterns, and the weights in the output layer are equal to the output patterns. This simple network can be used for rapid prototyping. The counterpropagation network usually has more hidden neurons than required. However, such an excessive number of hidden neurons are also used in more sophisticated feedforward networks such as the *probabilistic neural network* (PNN) Specht (1990) or the *general regression neural networks* (GRNN) Specht (1992).

12.5.3 WTA Architecture

The winner take all (WTA) network was proposed by Kohonen (1988). This is basically a one-layer network used in the unsupervised training algorithm to extract a statistical property of the input data, Figure 12.14a. At the first step, all input data are normalized so that the length of each input vector is the same and, usually, equal to unity, Equation 12.36. The activation functions of neurons are unipolar and continuous. The learning process starts with a weight initialization to small random values. During the learning process the weights are changed only for the neuron with the highest value on the output—the winner:

$$\Delta \mathbf{w}_w = c(\mathbf{x} - \mathbf{w}_w) \tag{12.37}$$

where

$\mathbf{w}_w$ = weights of the winning neuron,
$\mathbf{x}$ = input vector,
c = learning constant.

Usually, this single-layer network is arranged into a two-dimensional layer shape, as shown in Figure 12.14b. The hexagonal shape is usually chosen to secure strong interaction between neurons. Also, the algorithm is modified in such a way that not only the winning neuron but also neighboring neurons are allowed for the weight change. At the same time, the learning constant c in Equation 12.37 decreases with the distance from the winning neuron. After such an unsupervised training procedure, the Kohonen layer is able to organize data into clusters. Output of the Kohonen layer is then connected to the one- or two-layer feedforward network with the error backpropagation algorithm. This initial data organization in the WTA layer usually leads to rapid training of the following layer or layers.

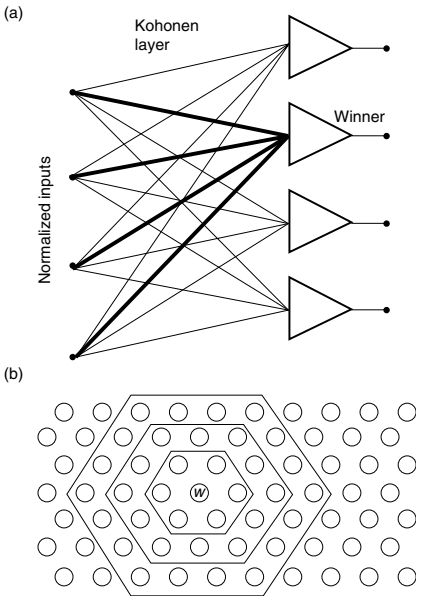


FIGURE 12.14 A winner take all architecture for cluster extracting in the unsupervised training mode: (a) network connections, (b) single-layer network arranged into a hexagonal shape.

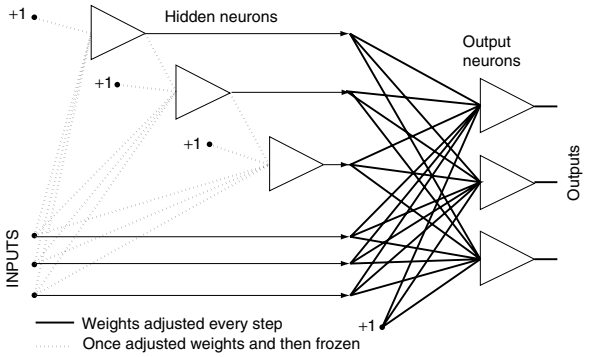


FIGURE 12.15 The cascade correlation architecture.

12.5.4 Cascade Correlation Architecture

The cascade correlation architecture was proposed by Fahlman and Lebiere (1990). The process of network building starts with a one-layer neural network and hidden neurons are added as needed. The network architecture is shown in Figure 12.15. In each training step, a new hidden neuron is added and its weights are adjusted to maximize the magnitude of the correlation between the new hidden neuron output and the residual error signal on the network output to be eliminated. The correlation parameter S must be maximized:

$$S = \sum_{o=1}^{O} \left| \sum_{p=1}^{P} (V_p - \overline{V})(E_{po} - \overline{E}_o) \right| \tag{12.38}$$

where

O = number of network outputs,
P = number of training patterns,
V_p = output on the new hidden neuron,
E_{po} = error on the network output.

$\overline{V}$ and $\overline{E}_o$ are average values of V_p and E_{po}, respectively. By finding the gradient, $\delta S/\delta w_i$, the weight, adjustment for the new neuron can be found as

$$\Delta w_i = \sum_{o=1}^{O} \sum_{p=1}^{P} \sigma_o (E_{po} - \overline{E}_o) f'_p x_{ip} \tag{12.39}$$

where

σ_o = sign of the correlation between the new neuron output value and network output,
f'_p = derivative of activation function for pattern p,
x_{ip} = input signal.

The output neurons are trained using the delta or quickprop algorithms. Each hidden neuron is trained just once and then its weights are frozen. The network learning and building process is completed when satisfactory results are obtained.

12.5.5 Radial Basis Function Networks

The structure of the radial basis network is shown in Figure 12.16. This type of network usually has only one hidden layer with special neurons. Each of these neurons responds only to the inputs signals close to the stored pattern. The output signal h_i of the ith hidden neuron is computed using formula

$$h_i = \exp\left(-\frac{\|\mathbf{x} - \mathbf{s}_i\|^2}{2\sigma^2}\right) \tag{12.40}$$

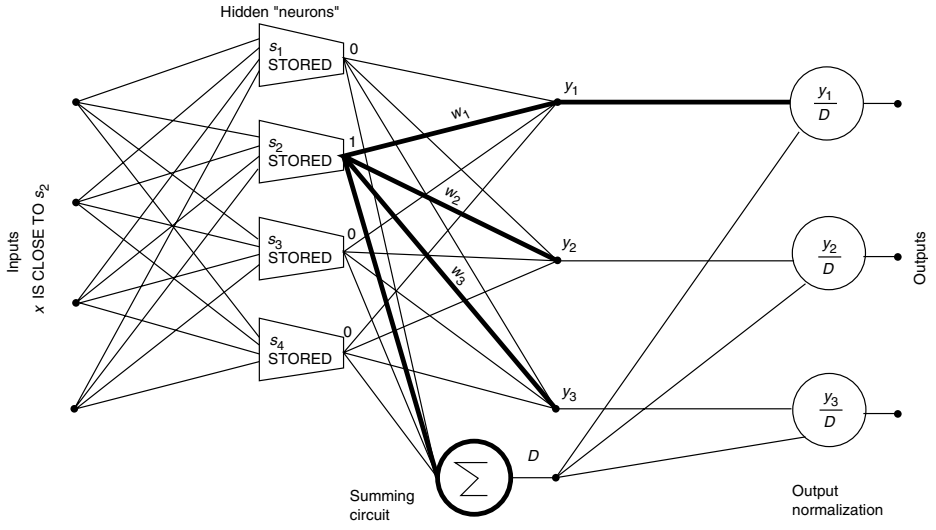


FIGURE 12.16 A typical structure of the radial basis function network.

where

$\mathbf{x}$ = input vector,
$\mathbf{s}_i$ = stored pattern representing the center of the i cluster,
σ_i = radius of the cluster.

Note that the behavior of this "neuron" significantly differs form the biological neuron. In this "neuron," excitation is not a function of the weighted sum of the input signals. Instead, the distance between the input and stored pattern is computed. If this distance is zero, the neuron responds with a maximum output magnitude equal to one. This neuron is capable of recognizing certain patterns and generating output signals that are functions of a similarity. Features of this neuron are much more powerful than a neuron used in the backpropagation networks. As a consequence, a network made of such neurons is also more powerful.

If the input signal is the same as a pattern stored in a neuron, then this neuron responds with 1 and remaining neurons have 0 on the output, as is illustrated in Figure 12.16. Thus, output signals are exactly equal to the weights coming out from the active neuron. This way, if the number of neurons in the hidden layer is large, then any input–output mapping can be obtained. Unfortunately, it may also happen that for some patterns several neurons in the first layer will respond with a nonzero signal. For a proper approximation, the sum of all signals from the hidden layer should be equal to one. To meet this requirement, output signals are often normalized, as shown in Figure 12.16.

The radial-based networks can be designed or trained. Training is usually carried out in two steps. In the first step, the hidden layer is usually trained in the unsupervised mode by choosing the best patterns for cluster representation. An approach similar to that used in the WTA architecture can be used. Also in this step, radii σ_i must be found for a proper overlapping of clusters.

The second step of training is the error backpropagation algorithm carried on only for the output layer. Since this is a supervised algorithm for one layer only, the training is very rapid, 100–1000 times faster than in the backpropagation multilayer network. This makes the radial basis-function network very attractive. Also, this network can be easily modeled using computers; however, its hardware implementation would be difficult.

12.6 Recurrent Neural Networks

In contrast to feedforward neural networks, with **recurrent networks** neuron outputs can be connected with their inputs. Thus, signals in the network can continuously circulate. Until recently, only a limited number of recurrent neural networks were described.

12.6.1 Hopfield Network

The single-layer recurrent network was analyzed by Hopfield (1982). This network, shown in Figure 12.17, has unipolar hard threshold neurons with outputs equal to 0 or 1. Weights are given by a symmetrical square matrix $\mathbf{W}$ with zero elements ($w_{ij} = 0$ for $i = j$) on the main diagonal. The stability of the system is usually analyzed by means of the *energy function*

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij} v_i v_j \tag{12.41}$$

It has been proved that during signal circulation the energy E of the network decreases and the system converges to the stable points. This is especially true when the values of system outputs are updated in the asynchronous mode. This means that at a given cycle, only one random output can be changed to the required values. Hopfield also proved that those stable points, which the system converges, can be

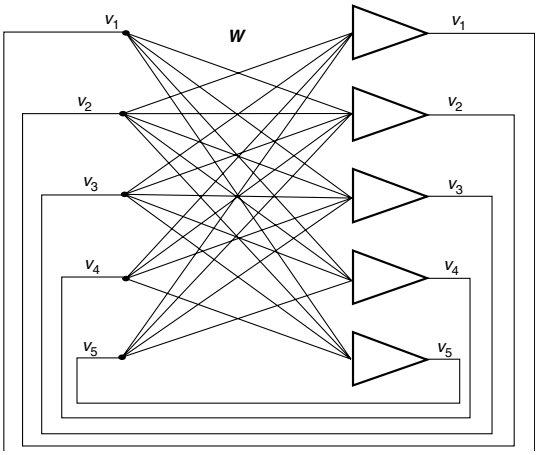


FIGURE 12.17 A Hopfield network or autoassociative memory.

programmed by adjusting the weights using a modified Hebbian rule,

$$\Delta w_{ij} = \Delta w_{ji} = (2v_i - 1)(2v_j - 1)c \tag{12.42}$$

Such memory has limited storage capacity. Based on experiments, Hopfield estimated that the maximum number of stored patterns is $0.15N$, where N is the number of neurons.

Later the concept of energy function was extended by Hopfield (1984) to one-layer recurrent networks having neurons with continuous activation functions. These types of networks were used to solve many optimization and linear programming problems.

12.6.2 Autoassociative Memory

Hopfield (1984) extended the concept of his network to autoassociative memories. In the same network structure as shown in Figure 12.17, the bipolar hard-threshold neurons were used with outputs equal to −1 or +1. In this network, pattern $\mathbf{s}_m$ are stored into the weight matrix $\mathbf{W}$ using the autocorrelation algorithm

$$W = \sum_{m=1}^{M} \mathbf{s}_m \mathbf{s}_m^T - M\mathbf{I} \tag{12.43}$$

where M is the number of stored patterns and $\mathbf{I}$ is the unity matrix. Note that $\mathbf{W}$ is the square symmetrical matrix with elements on the main diagonal equal to zero (w_{ji} for $i = j$). Using a modified formula 12.42, new patterns can be added or subtracted from memory. When such memory is exposed to a binary bipolar pattern by enforcing the initial network states, after signal circulation the network will converge to the closest (most similar) stored pattern or to its complement. This stable point will be at the closest minimum of the energy

$$E(\mathbf{v}) = -\frac{1}{2}\mathbf{v}^T\mathbf{W}\mathbf{v} \tag{12.44}$$

Like the Hopfield network, the autoassociative memory has limited storage capacity, which is estimated to be about $M_{\max} = 0.15N$. When the number of stored patterns is large and close to the memory capacity, the network has a tendency to converge to spurious states, which were not stored. These spurious states are additional minima of the energy function.

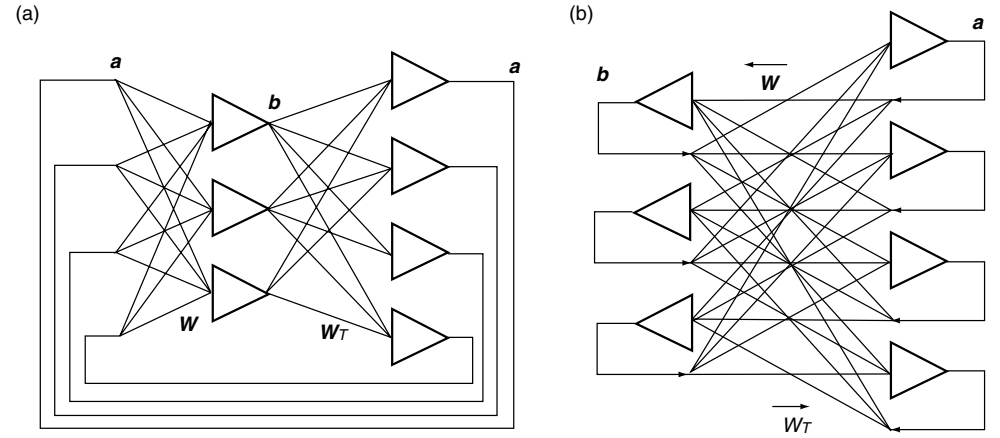


FIGURE 12.18 An example of the bidirectional autoassociative memory: (a) drawn as a two-layer network with circulating signals, (b) drawn as a two-layer network with bidirectional signal flow.

12.6.3 Bidirectional Associative Memories (BAM)

The concept of the autoassociative memory was extended to bidirectional associative memories (BAMs) by Kosko (1987, 1988). This memory, shown in Figure 12.18, is able to associate pairs of the patterns **a** and **b**. This is the two-layer network with the output of the second layer connected directly to the input of the first layer. The weight matrix of the second layer is $\mathbf{W}^T$ and **W** for the first layer. The rectangular weight matrix **W** is obtained as a sum of the cross-correlational matrices

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{a}_m \mathbf{b}_m \tag{12.45}$$

where M is the number of stored pairs, and $\mathbf{a}_m$ and $\mathbf{b}_m$ are the stored vector pairs. If the nodes **a** or **b** are initialized with a vector similar to the stored one, then after signal circulations, both stored patterns $\mathbf{a}_m$ and $\mathbf{b}_m$ should be recovered. The BAM has limited memory capacity and memory corruption problems similar to the autoassociative memory. The BAM concept can be extended for association of three or more vectors.

12.7 Fuzzy Systems

The main applications of neural networks are related to the nonlinear mapping of n-dimensional input variables into m-dimensional output variables. Such a function is often required in control systems, where, for specific measured variables, certain control variables must be generated. Another approach for nonlinear mapping of one set of variables into another set of variables is the *fuzzy controller*. The principle of operation of the fuzzy controller significantly differs from neural networks. The block diagram of a fuzzy controller is shown in Figure 12.19. In the first step, analog inputs are converted into a set of fuzzy variables. In this step, for each analog input, 3–9 fuzzy variables typically are generated. Each fuzzy variable has an analog value between zero and one. In the next step, a fuzzy logic is applied to the input fuzzy variables and a resulting set of output variables is generated. In the last step, known as *defuzzification*, from a set of output fuzzy variables, one or more output analog variables are generated, which are used as control variables.

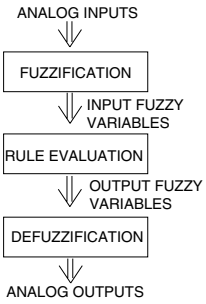


FIGURE 12.19 The block diagram of the fuzzy controller.

FIGURE 12.20 Fuzzification process: (a) typical membership functions for the fuzzification and the defuzzification processes, (b) example of converting a temperature into fuzzy variables.

12.7.1 Fuzzification

The purpose of fuzzification is to convert an analog variable input into a set of fuzzy variables. For higher accuracy, more fuzzy variables will be chosen. To illustrate the fuzzification process, consider that the input variable is the temperature and is coded into five fuzzy variables: cold, cool, normal, warm, and hot. Each fuzzy variable should obtain a value between zero and one, which describes a *degree of association* of the analog input (temperature) within the given fuzzy variable. Sometimes, instead of the term *degree of association*, the term *degree of membership* is used. The process of fuzzification is illustrated in Figure 12.20. Using Figure 12.20 we can find the degree of association of each fuzzy variable with the given temperature. For example, for a temperature of 57°F, the following set of fuzzy variables is obtained: [0, 0.5, 0.2, 0, 0], and for $T = 80$°F, it is [0, 0, 0.25, 0.7, 0]. Usually only one or two fuzzy variables have a value other than zero. In the example, trapezoidal functions are used for calculation of the degree of association. Various different functions such as triangular or Gaussian can also be used, as long as the computed value is in the range from zero to one. Each membership function is described by only three or four parameters, which have to be stored in memory.

For proper design of the fuzzification stage, certain practical rules should be used:

- Each point of the input analog variable should belong to at least one and no more than two membership functions.
- For overlapping functions, the sum of two membership functions must not be larger than one. This also means that overlaps must not cross the points of maximum values (ones).
- For higher accuracy, more membership functions should be used. However, very dense functions lead to frequent system reaction and sometimes to system instability.

12.7.2 Rule Evaluation

In contrary to boolean logic where variables can have only binary states, in fuzzy logic all variables may have any values between zero and one. The fuzzy logic consists of the same basic: ∧—AND, ∨—OR, and NOT operators:

$A \wedge B \wedge C \Rightarrow \min\{A, B, C\}$—smallest value of A or B or C
$A \vee B \vee C \Rightarrow \max\{A, B, C\}$—largest value of A or B or C
$\overline{A} \Rightarrow 1\ 1–A$—one minus value of A

(a)

	y_1	y_2	y_3
x_1	z_1	z_1	z_2
x_2	z_1	z_3	z_3
x_3	z_2	z_4	z_4
x_4	z_1	z_2	z_3
x_5	z_1	z_2	z_4

(b)

	y_1	y_2	y_3
x_1	t_{11}	t_{12}	t_{13}
x_2	t_{21}	t_{22}	t_{23}
x_3	t_{31}	t_{32}	t_{33}
x_4	t_{41}	t_{42}	t_{43}
x_5	t_{51}	t_{52}	t_{53}

FIGURE 12.21 Fuzzy tables: (a) table with fuzzy rules, (b) table with the intermediate variables t_{ij}.

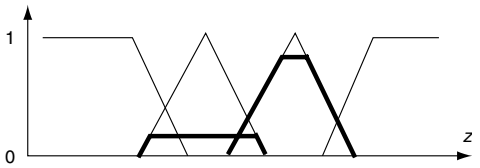


FIGURE 12.22 Illustration of the defuzzification process.

For example, $0.1 \wedge 0.7 \wedge 0.3 = 0.1$, $0.1 \vee 0.7 \vee 0.3 = 0.7$, and $\overline{0.3} = 0.7$. These rules are also known as *Zadeh* AND, OR, and NOT operators (Zadeh, 1965). Note that these rules are true also for classical binary logic.

Fuzzy rules are specified in the *fuzzy table* as it is shown for a given system. Consider a simple system with two analog input variables x and y, and one output variable z. The goal is to design a fuzzy system generating z as $f(x, y)$. After fuzzification, the analog variable x is represented by five fuzzy variables: x_1, x_2, x_3, x_4, x_5 and an analog variable y is represented by three fuzzy variables: y_1, y_2, y_3. Assume that an analog output variable is represented by four fuzzy variables: z_1, z_2, z_3, z_4. The key issue of the design process is to set proper output fuzzy variables z_k for all combinations of input fuzzy variables, as shown in the table in Figure 12.21. The designer has to specify many rules such as if inputs are represented by fuzzy variables x_i and y_j, then the output should be represented by fuzzy variable z_k. Once the fuzzy table is specified, the fuzzy logic computation proceeds in two steps. First, each field of the fuzzy table is filled with intermediate fuzzy variables t_{ij}, obtained from AND operator $t_{ij} = \min\{x_i, y_j\}$, as shown in Figure 12.21b. This step is independent of the required rules for a given system. In the second step, the OR (max) operator is used to compute each output fuzzy variable z_k. In the given example in Figure 12.21, $z_1 = \max\{t_{11}, t_{12}, t_{21}, t_{41}, t_{51}\}$, $z_2 = \max\{t_{13}, t_{31}, t_{42}, t_{52}\}$, $z_3 = \max\{t_{22}, t_{23}, t_{43},\}$, $z_4 = \max\{t_{32}, t_{34}, t_{53}\}$. Note that the formulas depend on the specifications given in the fuzzy table shown in Figure 12.21a.

12.7.3 Defuzzification

As a result of fuzzy rule evaluation, each analog output variable is represented by several fuzzy variables. The purpose of defuzzification is to obtain analog outputs. This can be done by using a membership function similar to that shown in Figure 12.20. In the first step, fuzzy variables obtained from rule evaluations are used to modify the membership function employing the formula

$$\mu_k^*(z) = \min\{\mu_k(z), z_k\} \tag{12.46}$$

For example, if the output fuzzy variables are 0, 0.2, 0.7, 0.0, then the modified membership functions have shapes shown by the thick line in Figure 12.22. The analog value of the z variable is found as a *center*

of gravity of modified membership functions $\mu_k^*(z)$,

$$z_{\text{analog}} = \frac{\sum_{k=1}^{n} \int_{2\infty}^{+\infty} \mu_k^*(z)\, z\, dz}{\sum_{k=1}^{n} \int_{2\infty}^{+\infty} \mu_k^*(z)\, dz} \tag{12.47}$$

In the case where shapes of the output membership functions $\mu_k(z)$ are the same, the equation can be simplified to

$$z_{\text{analog}} = \frac{\sum_{k=1}^{n} z_k zc_k}{\sum_{k=1}^{n} z_k} \tag{12.48}$$

where

n = number of membership function of z_{analog} output variable,
z_k = fuzzy output variables obtained from rule evaluation,
zc_k = analog values corresponding to the center of kth membership function.

Equation 12.47 is usually too complicated to be used in a simple microcontroller based system; therefore, in practical cases, Equation 12.48 is used more frequently.

12.7.4 Design Example

Consider the design of a simple fuzzy controller for a sprinkler system. The sprinkling time is a function of humidity and temperature. Four membership functions are used for the temperature, three for humidity, and three for the sprinkle time, as shown in Figure 12.23. Using intuition, the fuzzy table can be developed, as shown in Figure 12.24a.

Assume a temperature of 60°F and 70% humidity. Using the membership functions for temperature and humidity, the following fuzzy variables can be obtained for the temperature: [0, 0.2, 0.5, 0], and for the humidity: [0, 0.4, 0.6]. Using the min operator, the fuzzy table can be now filled with temporary fuzzy variables, as shown in Figure 12.24b. Note that only four fields have nonzero values. Using fuzzy rules, as shown in Figure 12.24a, the max operator can be applied in order to obtain fuzzy output variables: short $\rightarrow o_1 = \max\{0, 0, 0.2, 0.5, 0\} = 0.5$, medium $\rightarrow o_2 = \max\{0, 0, 0.2, 0.4, 0\} = 0.4$, long $\rightarrow o_3 = \max\{0, 0\} = 0$.

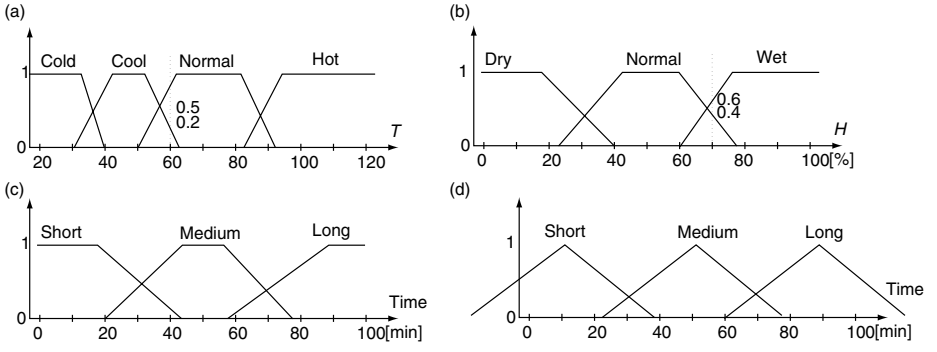


FIGURE 12.23 Membership functions for the presented example: (a) and (b) are membership functions for input variables, (c) and (d) are two possible membership functions for the output variable.

(a)

	Dry	Normal	Wet
Cold	M	S	S
Cool	M	M	S
Warm	L	M	S
Hot	L	M	S

(b)

		Dry	Normal	Wet
		0	0.4	0.6
Cold	0	0	0	0
Cool	0.2	0	0.2	0.2
Warm	0.5	0	0.4	0.5
Hot	0	0	0	0

FIGURE 12.24 Fuzzy tables: (a) fuzzy rules for the design example, (b) fuzzy temporary variables for the design example.

Using Equation 12.47 and Figure 12.23c, a sprinkle time of 28 min is determined. When the simplified approach is used with Equation 12.46 and Figure 12.23d, then sprinkle time is 27 min.

12.8 Genetic Algorithms

The success of the artificial neural networks encouraged researchers to search for other patterns in nature to follow. The power of genetics through evolution was able to create such sophisticated machines as the human being. Genetic algorithms follow the evolution process in nature to find better solutions to some complicated problems. The foundations of genetic algorithms are given by Holland (1975) and Goldberg (1989). After initialization, the steps *selection, reproduction with a crossover*, and *mutation* are repeated for each generation. During this procedure, certain strings of symbols, known as chromosomes, evaluate toward a better solution. The genetic algorithm method begins with coding and an initialization. All significant steps of the genetic algorithm will be explained using a simple example of finding a maximum of the function $(\sin^2(x) - 0.5 * x)^2$ with the range of x from 0 to 1.6. Note that in this range, the function has a global maximum at $x = 1.309$, and a local maximum at $x = 0.262$.

12.8.1 Coding and Initialization

At first, the variable x has to be represented as a string of symbols. With longer strings, the process usually converges faster, so the fewer symbols for one string field that are used, the better. Although this string may be the sequence of any symbols, the binary symbols 0 and 1 are usually used. In our example, 6-bit binary numbers are used for coding, having a decimal value of $40x$. The process starts with a random generation of the initial population given in Table 12.1.

12.8.2 Selection and Reproduction

Selection of the best members of the population is an important step in the genetic algorithm. Many different approaches can be used to rank individuals. In this example the ranking function is given. Highest rank has member number 6, and lowest rank has member number 3. Members with higher rank should have higher chances to reproduce. The probability of reproduction for each member can be obtained as a fraction of the sum of all objective function values. This fraction is shown in the last column

TABLE 12.1 Initial Population

String Number	String	Decimal Value	Variable Value	Function Value	Fraction of Total
1	101101	45	1.125	0.0633	0.2465
2	101000	40	1.000	0.0433	0.1686
3	010100	20	0.500	0.0004	0.0016
4	100101	37	0.925	0.0307	0.1197
5	001010	10	0.250	0.0041	0.0158
6	110001	49	1.225	0.0743	0.2895
7	100111	39	0.975	0.0390	0.1521
8	000100	4	0.100	0.0016	0.0062
Total				0.2568	1.0000

of Table 12.1. Note that to use this approach, our objective function should always be positive. If it is not, the proper normalization should be introduced at first.

12.8.3 Reproduction

The numbers in the last column of Table 12.1 show the probabilities of reproduction. Therefore, most likely members numbers 3 and 8 will not be reproduced, and members 1 and 6 may have two or more copies. Using a random reproduction process, the following population, arranged in pairs, could be generated:

$$101101 \rightarrow 45 \quad 110001 \rightarrow 49 \quad 100101 \rightarrow 37 \quad 110001 \rightarrow 49$$
$$100111 \rightarrow 39 \quad 101101 \rightarrow 45 \quad 110001 \rightarrow 49 \quad 101000 \rightarrow 40$$

If the size of the population from one generation to another is the same, two parents should generate two children. By combining two strings, two other strings should be generated. The simplest way to do this is to split in half each of the parent strings and exchange substrings between parents. For example, from parent strings 010100 and 100111, the following child strings will be generated: 010111 and 100100. This process is known as the *crossover*. The resultant children are

$$101111 \rightarrow 47 \quad 110101 \rightarrow 53 \quad 100001 \rightarrow 33 \quad 110000 \rightarrow 48$$
$$100101 \rightarrow 37 \quad 101001 \rightarrow 41 \quad 110101 \rightarrow 53 \quad 101001 \rightarrow 41$$

In general, the string need not be split in half. It is usually enough if only selected bits are exchanged between parents. It is only important that bit positions are not changed.

12.8.4 Mutation

In the evolutionary process, reproduction is enhanced with mutation. In addition to the properties inherited from parents, offspring acquire some new random properties. This process is known as mutation. In most cases mutation generates low-ranked children, which are eliminated in the reproduction process. Sometimes, however, the mutation may introduce a better individual with a new property. This prevents the process of reproduction from degeneration. In genetic algorithms, mutation usually plays a secondary role. For very high levels of mutation, the process is similar to random pattern generation, and such a searching algorithm is very inefficient. The mutation rate is usually assumed to be at a level well below 1%. In this example, mutation is equivalent to the random bit change of a given pattern. In this simple case, with short strings and a small population, and with a typical mutation rate of 0.1%, the patterns remain practically unchanged by the mutation process. The second generation for this example is shown in Table 12.2.

TABLE 12.2 Population of Second Generation

String Number	String	Decimal Value	Variable Value	Function Value	Fraction of Total
1	010111	47	1.175	0.0696	0.1587
2	100100	37	0.925	0.0307	0.0701
3	110101	53	1.325	0.0774	0.1766
4	010001	41	1.025	0.0475	0.1084
5	100001	33	0.825	0.0161	0.0368
6	110101	53	1.325	0.0774	0.1766
7	110000	48	1.200	0.0722	0.1646
8	101001	41	1.025	0.0475	0.1084
Total				0.4387	1.0000

Note that two identical highest ranking members of the second generation are very close to the solution $x = 1.309$. The randomly chosen parents for the third generation are:

$$\begin{array}{llll} 010111 \rightarrow 47 & 110101 \rightarrow 53 & 110000 \rightarrow 48 & 101001 \rightarrow 41 \\ 110101 \rightarrow 53 & 110000 \rightarrow 48 & 101001 \rightarrow 41 & 110101 \rightarrow 53 \end{array}$$

which produces the following children:

$$\begin{array}{llll} 010101 \rightarrow 21 & 110000 \rightarrow 48 & 110001 \rightarrow 49 & 101101 \rightarrow 45 \\ 110111 \rightarrow 55 & 110101 \rightarrow 53 & 101000 \rightarrow 40 & 110001 \rightarrow 49 \end{array}$$

The best result in the third population is the same as in the second one. By careful inspection of all strings from the second or third generation, it may be concluded that using crossover, where strings are always split in half, the best solution $110100 \rightarrow 52$ will never be reached, regardless of how many generations are created. This is because none of the population in the second generation has a substring ending with 100. For such crossover, a better result can be only obtained due to the mutation process, which may require many generations. Better results in the future generation also can be obtained when strings are split in random places. Another possible solution is that only randomly chosen bits are exchanged between parents.

The genetic algorithm is very rapid, and it leads to a good solution within a few generations. This solution is usually close to global maximum, but not the best.

Defining Terms

Backpropagation: Training technique for multilayer neural networks.
Bipolar neuron: Neuron with output signal between –1 and +1.
Feedforward network: Network without feedback.
Perceptron: Network with hard threshold neurons.
Recurrent network: Network with feedback.
Supervised learning: Learning procedure when desired outputs are known.
Unipolar neuron: Neuron with output signal between 0 and +1.
Unsupervised learning: Learning procedure when desired outputs are unknown.

References

Fahlman, S.E. 1988. Faster-learning variations on backpropagation: An empirical study. *Proceedings of the Connectionist Models Summer School,* D. Touretzky, G. Hinton, and T. Sejnowski, Eds., Morgan Kaufmann, San Mateo, CA.

Fahlman, S.E. and Lebiere, C. 1990. The cascade correlation learning architecture. *Adv. Ner. Inf. Proc. Syst.,* 2, D.S. Touretzky, ed., pp. 524–532. Morgan Kaufmann, Los Altos, CA.

Goldberg, D.E. 1989. *Genetic Algorithm in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

Grossberg, S. 1969. Embedding fields: a theory of learning with physiological implications. *Journal of Mathematical Psychology* 6:209–239.

Hebb, D.O. 1949. *The Organization of Behavior, a Neuropsychological Theory.* John Wiley & Sons, New York.

Hecht-Nielsen, R. 1987. Counterpropagation networks. *Appl. Opt.* 26(23):4979–4984.

Hecht-Nielsen, R. 1988. Applications of counterpropagation networks. *Neural Networks* 1:131–139.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, MI.

Hopfield, J.J. 1982. Neural networks and physical systems with emergent collective computation abilities. *Proceedings of the National Academy of Science* 79:2554–2558.

Hopfield, J.J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science* 81:3088–3092.

Kohonen, T. 1988. The neural phonetic typerater. *IEEE Computer* 27(3):11–22.

Kohonen, T. 1990. The self-organized map. *Proc. IEEE* 78(9):1464–1480.

Kosko, B. 1987. Adaptive bidirectional associative memories. *App. Opt.* 26:4947–4959.

Kosko, B. 1988. Bidirectional associative memories. *IEEE Trans. Sys. Man, Cyb.* 18:49–60.

McCulloch, W.S. and Pitts., W.H., 1943. A logical calculus of the ideas imminent in nervous activity. *Bull. Math. Biophys.* 5:115–133.

Minsky, M. and Papert, S. 1969. *Perceptrons.* MIT Press, Cambridge, MA.

Nilsson, N.J. 1965. *Learning Machines: Foundations of Trainable Pattern Classifiers.* McGraw-Hill, New York.

Nguyen, D. and Widrow, B. 1990. Improving the learning speed of 2-layer neural networks, by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference on Neural Networks* (San Diego), CA, June.

Pao, Y.H. 1989. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, Reading, MA.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psych. Rev.* 65:386–408.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. 1986. Learning internal representation by error propagation. *Parallel Distributed Processing.*, Vol. 1, pp. 318–362. MIT Press, Cambridge, MA.

Sejnowski, T.J. and Rosenberg, C.R. 1987. Parallel networks that learn to pronounce English text. *Complex Systems* 1:145–168.

Specht, D.F. 1990. Probalistic neural networks. *Neural Networks* 3:109–118.

Specht, D.F. 1992. General regression neural network. *IEEE Trans. Neural Networks* 2:568–576.

Wasserman, P.D. 1989. *Neural Computing Theory and Practice.* Van Nostrand Reinhold, New York.

Werbos, P., 1974. Beyond regression: new tools for prediction and analysis in behavioral sciences. Ph.D. diss., Harvard University.

Widrow, B. and Hoff, M.E. 1960. Adaptive switching circuits. 1960 IRE Western Electric Show and Convention Record, Part 4 (Aug. 23):96–104.

Widrow, B. 1962. Generalization and information storage in networks of adaline Neurons. In *Self-organizing Systems*, M.C. Jovitz, G.T. Jacobi, and G. Goldstein, eds., pp. 435–461. Sparten Books, Washington, D.C.

Wilamowski, M. and Torvik, L., 1993. Modification of gradient computation in the back-propagation algorithm. *ANNIE'93 - Artificial Neural Networks in Engineering.* November 14–17, 1993, St. Louis, Missou.; also in C.H Dagli, ed. 1993. *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol. 3, pp. 175–180. ASME Press, New York.

Zadeh, L.A. 1965. Fuzzy sets. *Information and Control* 8:338–353.

Zurada, J. 1992. *Introduction to Artificial Neural Systems.* West Publ.

13

Advanced Control of an Electrohydraulic Axis

Florin Ionescu
University of Applied Sciences

Crina Vlad
Politehnica University of Bucharest

Dragos Arotaritei
Aalborg University Esbjerg

13.1 Introduction

Due to the development of technology in the last few years, robots are seen as advanced mechatronic systems which require knowledge from mechanics, actuators, and control in order to perform very complex tasks. Different kinds of servo-systems, especially electrohydraulic, could be met at the executive level of the robots. Taking into account the most advanced control approaches, this paper deals with the implementation of advanced controllers besides conventional ones which are used in an electrohydraulic system. The considered electrohydraulic system is one of the axes of a robot. These robots possess three or more electrohydraulic axes, which are identical with the axis studied in this chapter.

An electrohydraulic axis whose mathematical model (MM) is described in this chapter presents a multitude of nonlinearities. Conventional controllers are becoming increasingly inappropriate to control the systems with an imprecise model where many nonlinearities are manifested. Therefore, advanced techniques such as neural networks and fuzzy algorithms are deeply involved in the control of such systems. Neural networks, initially proposed by McCulloch and Pitts, Rosenblatt, Widrow, had several

limitations that restricted the domain of applications. An important change took place in the 1980s when Hopfield's theory regarding recurrent neural networks, the model of self-organization developed by Kohonen, and cellular neural networks (Chua) relaunched this research field. The development of some efficient algorithms dedicated specifically to the architecture of neural networks, and the application of these networks in control, represents an interesting area of research in the contemporary world of science.

Fuzzy systems, in conjunction with neural networks, hold an important place in advanced techniques of control. These systems have origins in fuzzy set theory initiated by L. Zadeh. One essential feature of fuzzy systems is the approximate reasoning in which the variables are described in a qualitative manner. Due to the capability of fuzzy systems to deal with imprecise information, they are strongly recommended in order to express knowledge in the form of linguistic rules. In this way, the human operator's knowledge, which is linguistic or numerical, is used to generate the set of fuzzy if-then rules as a basis for a fuzzy controller. A main drawback of fuzzy systems is the difficulty to design them on the basis of a systematic methodology. To overcome this drawback, the learning procedures from neural networks are applied successfully in order to tune the parameters of membership functions.

The merging of these two fields has led to the emergence of neuro-fuzzy systems, which have been applied with promising results in the field of control-engineering. In order to improve dynamic and static performances of the systems characterized by nonlinearities and uncertainties, neuro-fuzzy controllers are used.

The present contribution is organized as follows. An introduction of the electrohydraulic systems with an emphasis on the control of such devices is realized in Section 13.2. Section 13.3 is devoted to the MM of electrohydraulic axes, and the subsequent sections treat the control of electrohydraulic axes through conventional methods (Section 13.4), fuzzy systems (Section 13.5), neural networks (Section 13.6), and neuro-fuzzy techniques (Section 13.7). Conclusions are given in Section 13.8.

13.2 Generalities Concerning ROBI_3, a Cartesian Robot with Three Electrohydraulic Axes

The automated installation, which uses electrohydraulic axes, whose mathematical model is described in Section 13.3, is a Cartesian robot named ROBI_3. ROBI_3 has three identical electrohydraulic axes and is built from aluminium profiles [21]. The slides are actuated by hydraulic servoactuators (Rexroth). Slades move on the linear guideways with balls and two recirculation paths. The hydraulic supply installation is placed under the robot table and has a cooling and controlling installation with air. The mechanical structure of the robot is depicted in Figure 13.1. The three axes of ROBI_3 are identically controlled by the controlling software named TORCH, which runs in Windows. The 32-bit dSPACE controlling hardware endowed with 10 A/D and D/A interfaces is plugged into the PC and serves as the interface between the PC and each of the axes.

An electrohydraulic axis consists of a servovalve and a hydraulic cylinder and has a nonlinear structure. The control system of one axis consists of:

1. The controller represented by a personal computer endowed with a process card
2. The electrohydraulic converter
3. The actuator (a linear hydraulic servomotor (LHM))
4. The mechanical process to be controlled, characterized by the slade position
5. The position transducer

The control system of robot ROBI_3 is illustrated in Figure 13.2, where the presence of three electrohydraulic axes, as well as the structure of one axes, identical to the others, can be observed.

The corresponding mathematical model for one axis, on the basis of which the control of the robot is achieved, is described in Section 13.3. Through numerical simulations of the three axes of the robot, the necessary mechanical structure interface data is obtained.

Preliminary experiments with driven and controlled variable: position and velocity are made in order to achieve experience (Figures 13.2 and 13.3). The diagram of a closed-loop position control with

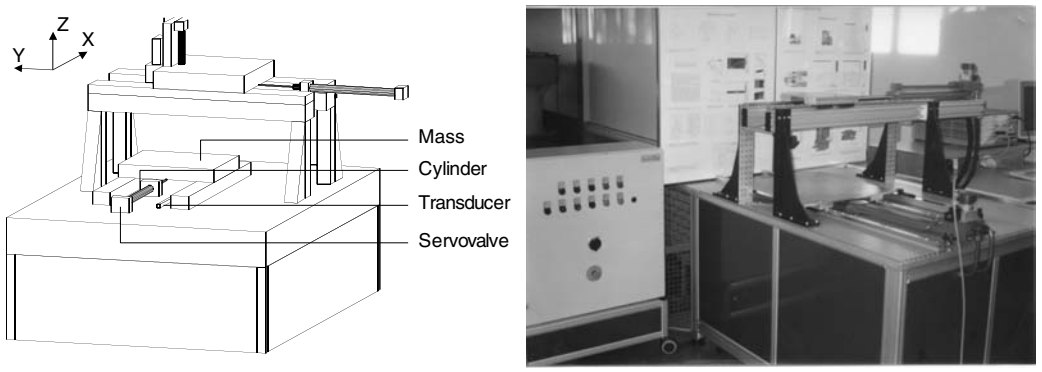


FIGURE 13.1 ROBI_3, a Cartesian robot with three axes [21]. (a) Design. (b) Practical allignment.

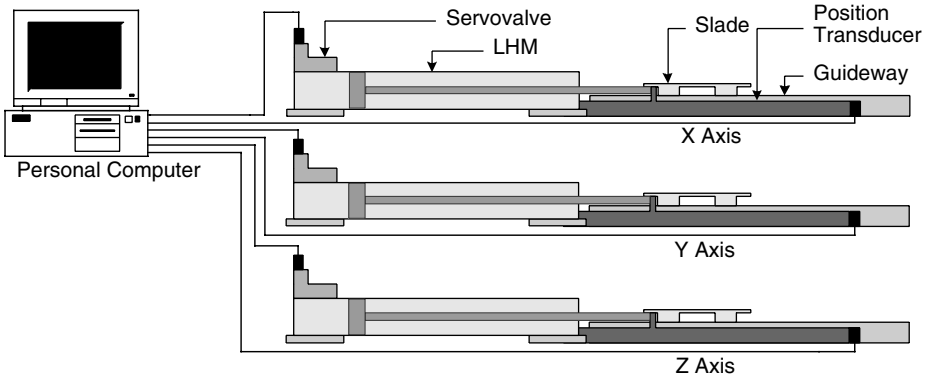


FIGURE 13.2 The control system of the robot [21,24].

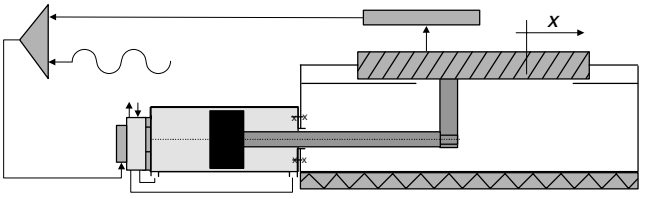


FIGURE 13.3 Diagram of a closed-loop position control with direct measurement.

direct measurement (driven by means of a servovalve) is shown in Figure 13.3, and the closed-loop position control with indirect measurement at spindle (actuated by means of a servovalve) is shown in Figure 13.4.

A volumetric Q-rate regulation with constant pressure ($Q \approx$ const; $p \cong$ const) is shown in Figure 13.5. However, this classic model, useful for application, was used only for preliminary results in simulation. One of the main reasons to use this is because we need a well-known mathematical model with well-studied behavior in order to test the controllers (neural and neuro-fuzzy, namely).

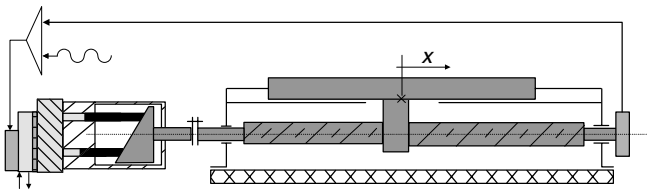


FIGURE 13.4 Closed loop-position control with indirect measurement.

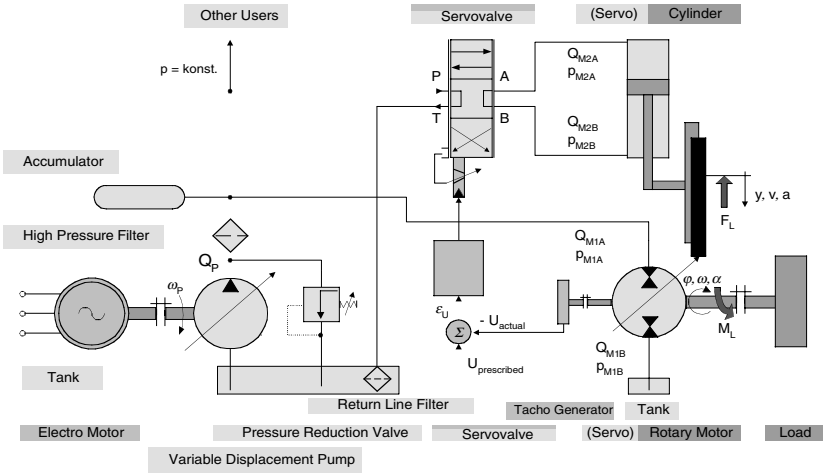


FIGURE 13.5 Volumetric Q-rate regulation with constant pressure.

13.3 Mathematical Model and Simulation of Electrohydraulic Axes

Section 13.3 deals with the analytical findings of the mathematical model (MM) of an electrohydraulic axis, a component part of robot ROBI_3. This method of analysis is advantageous because it offers the possibility to use this MM for other electrohydraulic axes as well, regardless of the different number of stages, and also allows the testing of dynamic performances of the axis at the design level.

In this section, the following were realized:

1. Static models of electrohydraulic system components (servovalve, hydraulic linear motor).
2. Parameters involved in MM, based on constructive and flowing regime characteristics.
3. Nonlinear MM of the proposed electrohydraulic system.
4. Structural scheme of the hydraulic axis in order to simulate its behavior (SIMULINK).
5. Investigations regarding MM, which certify the stability of the system and the fact that the modelled process is a rapid one.

Values of parameters that describe the MM are set based on hydraulic characteristics and on constructive parameters of the considered system.

13.3.1 The Extended Mathematical Model

The studied system consists of a servovalve and an asymmetric motor. In most cases, the control of an electrohydraulic axis is directed towards position control, velocity control, pressure control, or force control. The position control of the axis is studied. The position control loop for the proposed installation is illustrated in Figure 13.6, together with the denomination of some data.

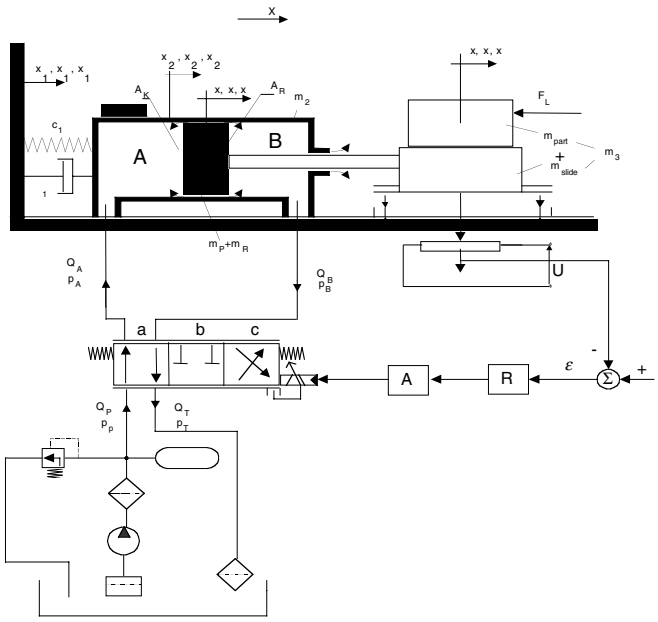


FIGURE 13.6 Control scheme of the servosystem.

13.3.2 Nonlinear Mathematical Model of the Servovalve

As previously mentioned, the servovalve used at the proposed electrohydraulic axis has four paths and three positions. The servovalve has four active control edges at the second hydraulic stage. Three stages could be distinguished by this servovalve: the electromechanical, the mechanohydraulical, and the hydromechanical one. Regarded as a system, a servovalve is complex, with various types of nonlinearities being manifested. Different static and dynamic nonlinearities such as dead zone, jump in origin, saturation, Coulombian and Newtonian frictions with hysteresis, and asymmetry appear in each of these three levels and also in the actuator of the electrohydraulic motor. These were taken into account in the modeling of the servovalve and of the cylinder behavior.

For the studied servovalve, the circulation of the fluid is considered directed from the pump to the admission chamber A (Q_A) and from the discharge chamber B to the reservoir (Q_B). Figure 13.6 presents the control loop where the transducer is placed on the feedback path and the controller R and the amplifier A are on the direct path. Electrical signal (±10 V or ±300 mA) is converted into the displacement x_v of the valve, and in this way in a flow Q, which is transmitted to the linear hydraulic motor.

From the point of view of control characteristics, the number of active edges serves as a method of classification of slide valves [3]. A servovalve with four active control edges was considered. The lightly simplified mathematical model, which describes the functionality of the servovalve, consists of the following equations:

$$u(t) = L \cdot \frac{di(t)}{dt} + R \cdot i(t) \tag{13.1}$$

where L [H], the inductance of electrical level; R [W], the resistance of electrical level; $u(t)$ [V], the control voltage; $i(t)$ [A], the control intensity.

$$m \cdot x_v(t) + d \cdot x_v(t) + c \cdot x_v(t) = \sum F \tag{13.2}$$

where m [kg], the mass of valve; d [N/(m/s)], the linearized gradient of viscous friction for the piston of the valve; c[N/m], the coefficient of hydraulic elasticity; x_v [m], the spool displacement; $\sum F$ [N], the

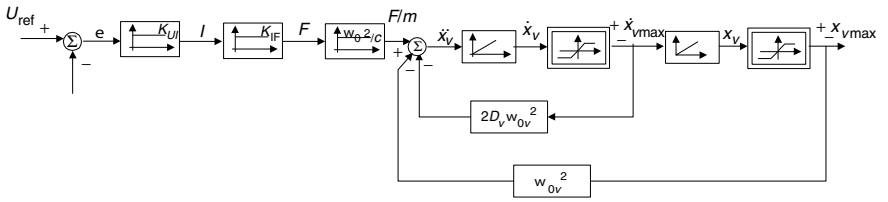


FIGURE 13.7 Nonlinear MM of the first stages of the servovalve.

resulting force, which actuates on the valve spool.

$$\ddot{x}_v(t) + 2 \cdot D_v \cdot \omega_{0v} \cdot \dot{x}_v(t) + \omega_{0v}^2 \cdot x_v(t) = \frac{\sum F}{m}$$

$$\ddot{x}_v(t) = k^* - 2 \cdot D_v \cdot \omega_{0v} \cdot \dot{x}_v(t) - \omega_{0v}^2 \cdot x_v(t) \quad \text{where } k^* = \frac{\sum F}{m} \tag{13.3}$$

The displacement x_v, obtained based upon the mentioned equations, is implemented in SIMULINK using the scheme from Figure 13.7. This module of nonlinear MM includes two stages of electrohydraulic axis: the electrohydraulic and the hydromechanical one.

The corresponding equations for the four flows that go through the servovalve are

$$\begin{aligned}
Q_{PA} &= \alpha_D \cdot \pi \cdot D_v \cdot \sqrt{\frac{2}{\rho}} \cdot (x_0 + x_v(t)) \cdot \sqrt{p_P - p_A(t)}, \quad x_v \in [-x_0, x_{\max}] \\
Q_{AT} &= \alpha_D \cdot \pi \cdot D_v \cdot \sqrt{\frac{2}{\rho}} \cdot (x_0 - x_v(t)) \cdot \sqrt{p_A(t) - p_T}, \quad x_v \in [-x_{\max}, x_0] \\
Q_{PB} &= \alpha_D \cdot \pi \cdot D_v \cdot \sqrt{\frac{2}{\rho}} \cdot (x_0 - x_v(t)) \cdot \sqrt{p_P - p_B(t)}, \quad x_v \in [-x_{\max}, x_0] \\
Q_{BT} &= \alpha_D \cdot \pi \cdot D_v \cdot \sqrt{\frac{2}{\rho}} \cdot (x_0 + x_v(t)) \cdot \sqrt{p_B(t) - p_T}, \quad x_v \in [-x_0, x_{\max}]
\end{aligned} \tag{13.4}$$

where Q_{PA} [m^3/s], the flow to the hydraulic motor, from pump to the chamber A of the motor; Q_{AT} [m^3/s], the flow from the chamber A to reservoir; Q_{PB} [m^3/s], the flow from pump to the chamber B of the motor; Q_{BT} [m^3/s], the flow from the chamber B to reservoir; α_D [-], the discharge coefficient; D_v [m], the spool's diameter; x_0 [m], the dimension of the lap of the spool; x_v [m], the spool's displacement; p_A [N/m^2], the fluid pressure in chamber A; p_B [N/m^2], the fluid pressure in chamber B [kg/m^3] fluids density. The flows, which are transmitted to LHM and evacuated from LHM, are Q_A and Q_B, which are computed as following:

$$Q_A = Q_{PA} - Q_{AT}, \qquad Q_B = Q_{BT} - Q_{PB} \tag{13.5}$$

The lap of the spool is considered to be zero ($x_0 = 0$), and, therefore, the static characteristic is linear around the origin and also in the rest. With $Q_0 = \alpha_D \cdot \pi \cdot D_v \cdot \sqrt{2/\rho}$, the flow equations become

$$\begin{aligned}
Q_{PA} &= Q_0 \cdot x_v \cdot \sqrt{p_P - p_A}, & Q_{AT} &= Q_0 \cdot (-x_v) \cdot \sqrt{p_A - p_T} \\
Q_{PB} &= Q_0 \cdot (-x_v) \cdot \sqrt{p_P - p_B}, & Q_{BT} &= Q_0 \cdot x_v \cdot \sqrt{p_B - p_T}
\end{aligned} \tag{13.6}$$

13.3.3 Nonlinear Mathematical Model of Linear Hydraulic Motor

The differential equations, based on which the MM of the linear hydraulic motor (LHM) was achieved, are

i. Equation of the dynamic equilibrium of the forces reduced to the motor's rod
ii. Equation of movement and flow continuity

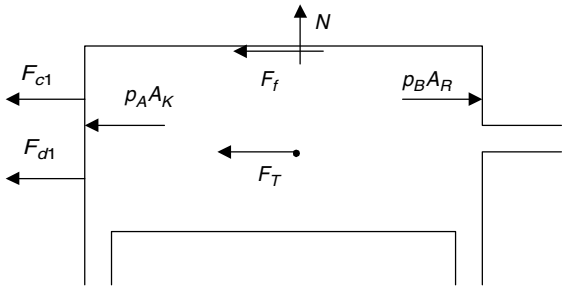


FIGURE 13.8 D'Alembert's principle applied to the cylinder.

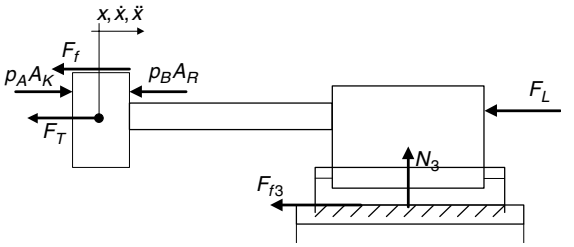


FIGURE 13.9 D'Alembert's principle applied to the rod, piston, and load.

Modeling the LHM, several simplifications (most of them concerning the Coulombian and Newtonian friction) were used. The forces that actuate on the LHM cylinder are depicted in Figure 13.8:

Applying D'Alembert's principle, the equation of dynamic equilibrium of forces for the cylinder of LHM is

$$x_2 = \frac{p_B \cdot A_R - p_A \cdot A_k + c_1 \cdot (x_1 - x_2) + d_1 \cdot (x_1 - x_2) + c_{\text{fu}} \cdot N \cdot \text{sgn}(x - x_2) + d_z \cdot (x - x_2)}{m_2 \cdot s} \tag{13.7}$$

where c_1 [N/m], the elasticity; d_1 [N/(m/s)], the linearized coefficient of the viscous Newtonian friction in the connection actuators' cylinder and wall; c_{fu} [-], the coefficient of the dry Coulombian friction in the cylinder and rod seals; d_z [N/(m/s)], the coefficient of Newtonian friction in the piston and rod seals; m_2 [kg], the cylinder mass; p_A [N/m^2], the fluid pressure in the admission chamber A of the actuator; p_B [N/m^2], the fluid pressure in the discharge chamber B of the actuator; A_K [m^2], the piston active area in chamber A; A_R [m^2], the piston active area in chamber B; N [N], the normal force, which determines the friction force between piston and cylinder; x [m], the piston displacement; x_1 [m], the wall displacement; and x_2 [m], the cylinder displacement.

The forces acting on the rod, piston, and working element are illustrated in Figure 13.9

The velocity corresponding to the rod, piston, and mass m_3 (slade, guideway, and loading are considered stiff fastened) is inferred from the equilibrium equation:

$$\dot{x} = \frac{p_A \cdot A_K - p_B \cdot A_R - F_L - c_{\text{fu}} \cdot N \cdot \text{sgn}(x - x_2) - d_z \cdot (x - x_2) - c_{\text{fu3}} \cdot N_3 \cdot \text{sgn}(x - x_1) - d_3 \cdot (x - x_1)}{(m_p + m_T + m_3) \cdot s} \tag{13.8}$$

where m_p [kg], the piston mass; m_T [kg], the rod mass; m_3 [kg], load mass (reduced at the rod of piston); c_{fu3} [-], the coefficient of Coulombian friction between guide ways and slade; d_3 [N/(m/s)], the coefficient of Newtonian friction between guideways and slade; N_3 [N], the normal force which appears between loading and table; F_L [N], the loading force.

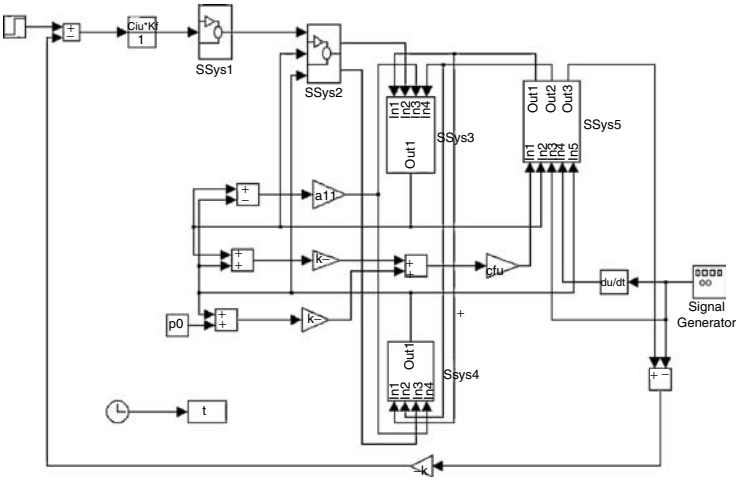


FIGURE 13.10 Electrohydraulic axis implemented in SIMULINK.

From the equations of continuity the pressures p_A and p_B are inferred:

$$p_A = [Q_A - A_K \cdot (x - x_2) - a_{11} \cdot (p_A - p_B)] \cdot \frac{E_{\text{ers}}}{V_{0K} + A_K \cdot (x - x_2)} \tag{13.9}$$

$$p_B = [A_R \cdot (x - x_2) + a_{11} \cdot (p_A - p_B) - a_{12} \cdot (p_B - p_0) - Q_B] \cdot \frac{E_{\text{ers}}}{(V_{0R} + A_R \cdot h) - A_R \cdot (x - x_2)} \tag{13.10}$$

where a_{11}, a_{12} [(N/m^2)/(m^3/s)], the gradients of leakages; $V_{0K,0R}$ [m^3], the initial average volume of chambers A and B, respectively; E_{ers} [N/m^2], the equivalent bulk modulus of oil; $p_{A,B}$ [N/m^2], the fluid pressure in chambers A and B, respectively; and h [m], the stroke of the piston-rod.

The LHM operation is based on the equations described above, namely (13.7)–(13.10). The MM proposed in this section is implemented in SIMULINK 2.1/ MATLAB 5.1 and has the structure presented in Figure 13.10.

The signal generator icon from the above figure generates the displacement of the wall x_1, which has a sinusoidal form with the frequency 0.5 Hz and amplitude 0.0001 m.

The subsystems Ssys1 and Ssys2 have as outputs the valve displacement x_v, and the flows Q_A and Q_B, respectively. Ssys3 is the block that implemented Equation 13.9, while the subsystem Ssys4 modelled Equation 13.10. The equations that describe the displacement of the wall cylinder and the LHM piston are modelled by subsystem Ssys5. The reference signal is a step one whose values are in the range 0–10 V.

13.4 Conventional Controllers Used to Control the Electrohydraulic Axis

This section is organized as follows: the first part presents the bibliographic research concerning the traditional directions of the control system, and the second one contains the testing of several classic control structures (PID and control algorithms with Luenberger observer) through simulations of the electrohydraulic axis endowed with these controllers.

The testing of the MM is performed with SIMULINK and has a goal of the achievement of reference experimental results in order to perform a comparative study of classical controllers and advanced control structures applied to the electrohydraulic axis.

13.4.1 PID, PI, PD with Filtering

The conventional control structures used in this chapter are PI (proportional-integral), PID (proportional-integral-derivative), and PD (proportional-derivative) with filtering coefficient.

The transfer function of the PI controller has the following expression:

$$H_{\mathrm{PI}} = \frac{U(s)}{\varepsilon(s)} = K_R \cdot \left(1 + \frac{1}{T_i \cdot s}\right) = K_R \cdot \frac{T_i \cdot s + 1}{T_i \cdot s} \tag{13.11}$$

K_R is the proportional factor, and T_i is the time constant of the integrative component.

The transfer function of the PID controller is described by the following equation:

$$H_{\mathrm{PID}} = \frac{U(s)}{\varepsilon(s)} = K_R \cdot \left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s\right) = K_R \cdot \frac{T_i \cdot T_d \cdot s^2 + T_i \cdot s + 1}{T_i \cdot s} \tag{13.12}$$

K_R and T_i have the same significance as previously mentioned, and T_d is the time constant of derivative component.

The transfer function corresponding to PD with filtering has the expression:

$$H_{\mathrm{PDF}}(s) = \frac{U(s)}{\varepsilon(s)} = K_R \cdot \frac{1 + T_d \cdot s}{1 + \alpha \cdot T_d \cdot s} \tag{13.13}$$

where the coefficient α could have values in the range 0.1–0.125.

Generally speaking, PID controllers are commonly used in industrial control systems and, therefore, are well established. Nevertheless, the results obtained using a PID controller for complex control loops are not very satisfactory because it could be costly and time consuming to retune such regulators. PI controller is enough in situations where derivative action is not frequently used.

13.4.2 Observer

The theory of observers, started with the work of Luenberger and Ackermann, is fairly complete and comprehensive. For the proposed axis, an $(n - m - 1)$ order structure of the observer is adopted, where $n = 5$ represents the order of the system and $m = 1$ is the number of outputs [25]. The model of the servodrive is described by five state variables: two of them for the second-order model of the servovalve and the other three for the third-order servoactuator. The use of a linear observer as a parallel model reconstitutes the state-variables of the installation and delivers them to the controller. Two possibilities could be followed: with partial and with global reconstruction. The solution chosen was with partial reconstruction [12]. The complete system consists of the installation with nonlinearities, a parallel second-order model for the servovalve, a third-order linear servoactuator, a correction matrix for the observer, and a controller with five loops for the five state variables [23,25,26].

The block diagram of an electrohydraulic axis controlled with a third-order observer is shown in Figure 13.11, where **A**, **b**, $\mathbf{c}^{\mathrm{T}}$ are the characteristic matrices of the linear system (electrohydraulic axis), **k** is the correction matrix, and **R** represents the matrix corresponding to the controller.

The simulation diagram of the electrohydraulic axis, controlled by a third-order observer, as it is depicted in SIMULINK, is illustrated in Figure 13.12.

The algorithm used to compute the matrix **k** and **r** consists of the following steps:

i. Achievement of the MM for servovalve and servoactuator
ii. Setting the state-variables of the process
iii. Obtaining the controller upon the dynamics of the closed-loop system
iv. Computing the correction matrix by using the desired poles for the observer [12]

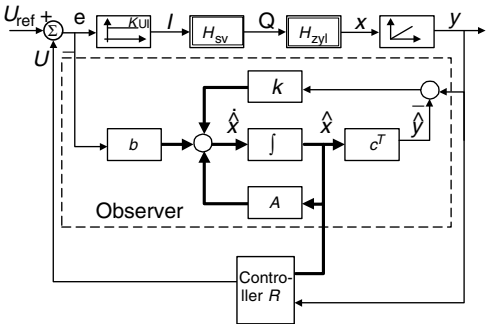


FIGURE 13.11 Block diagram of the control loop using the observer.

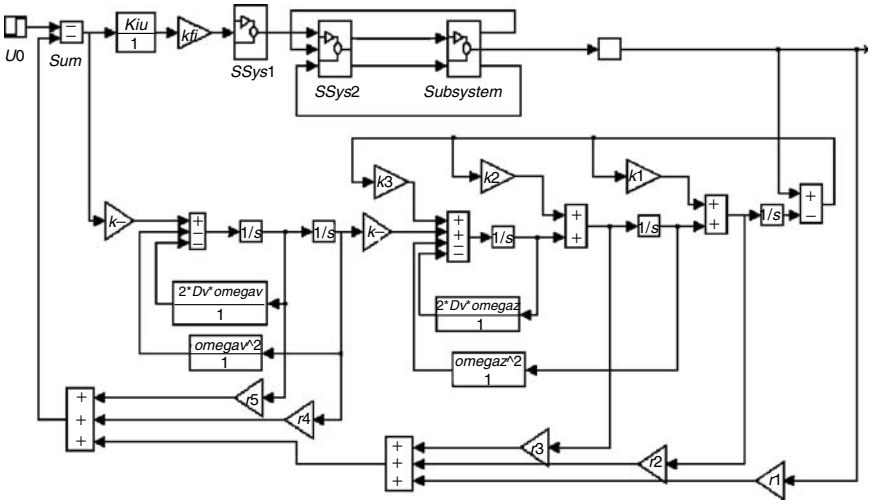


FIGURE 13.12 Control structure with observer for the electrohydraulic axis with SIMULINK.

13.4.2.1 The Linear Mathematical Models

The linear mathematical models (LMM) of the servovalve and of the servoactuator are obtained by using the appropriate transfer functions H_{SV}, H_Z:

$$H_{SV}(s) \hat{=} \frac{Q_V(s)}{I_V(s)} = \frac{k_V \omega_V^2}{s^2 + 2D_V \omega_V s + \omega_V^2} \tag{13.14}$$

$$H_Z(s) \hat{=} \frac{Y_Z(s)}{Q_V(s)} = \frac{k_Z \omega_Z^2}{(s^2 + 2D_Z \omega_Z s + \omega_Z^2)s} \tag{13.15}$$

with the following meanings: Q_V, the servovalve's flow; I_V, the current intensity, Y_Z, the rod's position.

13.4.2.2 LMM in the State of Space

The used variables are: x_1, the rod position; x_2, the rod velocity; x_3, the rod acceleration; x_4, the spool position; and x_5, the spool velocity.

$$\begin{aligned} x_1 &= x_2(t), \quad x_2 = x_3(t), \quad x_4 = x_5(t) \\ x_3 &= -\omega_Z^2 x_2(t) - 2D_Z\omega_Z x_3(t) + k_Z\omega_Z^2 x_4(t) \\ x_5 &= -\omega_V^2 x_4(t) - 2D_V\omega_V x_5(t) + k_V\omega_V^2 u(t) \end{aligned} \tag{13.16}$$

Thus the MM of the axis in state-space form becomes

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{b}u(t) \\ \mathbf{y}(t) &= \mathbf{c}^T\mathbf{x}(t) \end{aligned} \tag{13.17}$$

where

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\omega_Z^2 & -2D_Z\omega_Z & k_Z & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\omega_V^2 & -2D_V\omega_V \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ k_V \end{pmatrix}, \quad \mathbf{c}^T = (1\ 0\ 0\ 0\ 0) \tag{13.18}$$

13.4.2.3 Controller Design

The characteristic polynomial is obtained from $\det[s\mathbf{I} - (\mathbf{A}_C - \mathbf{b}_C\mathbf{r}^T)] = 0$, where $\mathbf{A}_C$ and $\mathbf{b}_C$ are the controllable forms of the matrices $\mathbf{A}$ and $\mathbf{b}$. If $\mathbf{A} \neq \mathbf{A}_C$, the use of transformation matrix $\mathbf{T}$ is advisable, in order to obtain $\mathbf{A}_C$ and $\mathbf{b}_C$. Thus $\mathbf{A}_C = \mathbf{TAT}^{-1}$, and $\mathbf{b}_C = \mathbf{Tb}$. The matrix $\mathbf{F} = \mathbf{A}_C - \mathbf{b}_C\mathbf{r}^T$ has the form

$$\mathbf{F} = \begin{pmatrix} 0 & 1 & . & . & 0 \\ 0 & 0 & . & . & 0 \\ . & . & . & . & . \\ 0 & 0 & . & . & 1 \\ -a_0 - r_1 & -a_1 - r_2 & . & . & -a_{n-1} - r_n \end{pmatrix} \tag{13.19}$$

The characteristic polynomial of the matrix $\mathbf{F}$ is

$$s^n + (a_{n-1} + r_n)s^{n-1} + \cdots + (a_1 + r_2)s + (a_0 + r_1) \tag{13.20}$$

The poles chosen for the closed-loop determine the polynomial

$$s^n + p_{n-1}s^{n-1} + p_{n-2}s^{n-2} + \cdots + p_1 s + p_0 \tag{13.21}$$

The polynomials (13.20) and (13.21) are identical; therefore, the coefficients of matrix $\mathbf{r}_R^T$ are

$$r_v = p_{v-1} - a_{v-1}, \quad v = 1, \ldots, n$$

If

$$\mathbf{A} = \mathbf{Ac}, \quad \mathbf{r}^T = \mathbf{r}_R^T$$

otherwise

$$\mathbf{r}^T = \mathbf{r}_R^T \mathbf{T} \tag{13.22}$$

13.4.2.4 Correction Matrix Design

The matrix **F** is $\mathbf{F} = \mathbf{A}^* - \mathbf{K}\mathbf{c}^T$, where $\mathbf{A}^*$ is the matrix of the observer. For **F** the chosen poles are $s_1, s_2, \ldots, s_n$, and

$$\det[s\mathbf{I}\text{-}\mathbf{F}] = (s - s_1)(s - s_2)\cdots(s - s_n) \tag{13.23}$$

$$\det[s\mathbf{I}\text{-}\mathbf{F}] = s^n + f_{n-1}s^{n-1} + \cdots + f_1 s + f_0 \tag{13.24}$$

From these two equations, the coefficients $k_1, k_2, \ldots, k_n$ are obtained.

In this case, $\mathbf{c}^T = (1\ 0\ 0)$ and the matrix of the third observer is

$$\mathbf{A}^* = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -\omega_Z^2 & -2D_Z\omega_Z \end{pmatrix} \tag{13.25}$$

The correction matrix influences the transient behavior; the further the poles of **F** from the poles of $\mathbf{A}^*$ the quicker the response.

13.4.3 Simulation Results of Electrohydraulic Axis with Conventional Controllers

Based on the above algorithm in order to determine the correction matrix and the controller matrix, the SIMULINK implementation of the observer involves the following values:

$$r_1 = 19.95854, \quad r_2 = 0.069481, \quad r_3 = -7.06024 \times 10^{-4},$$

$$r_4 = -3.158688 \times 10^{2}, \quad r_5 = -3.451209 \times 10^{-1}$$

for the controller, and

$k_1 = 1.67 \times 10^{-2}$, $k_2 = 3.7028 \times 10^{4}$, $k_3 = -6.969698 \times 10^{6}$ for the correction matrix.

When the reference signal is a step signal with $U = 10$ V, the simulation results are shown in Figures 13.13 and 13.14.

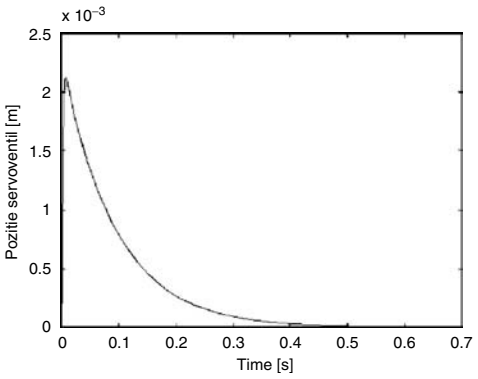


FIGURE 13.13 Position of servovalve for MM with observer.

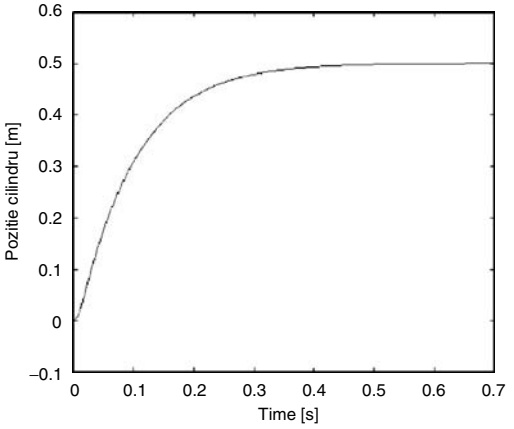


FIGURE 13.14 Cylinder position.

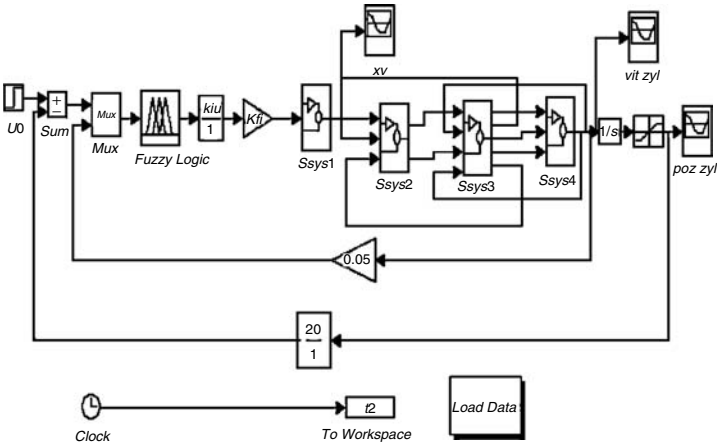


FIGURE 13.15 Electrohydraulic axis control with a fuzzy controller with two inputs.

13.5 Control of Electrohydraulic Axis with Fuzzy Controllers

Section 13.5 of this contribution is devoted to the presentation and the testing of nontraditional controllers based on fuzzy sets, which model the behavior of a human operator in the control process. The simulation results of an electrohydraulic axis with SUGENO and MAMDANI controllers are depicted. For the same number of inference rules extracted from the knowledge base, simulations proved that dynamic performances are improved for a fuzzy controller with two inputs.

The scheme achieved with SIMULINK to control the electrohydraulic axis with two inputs fuzzy controller and a MAMDANI or SUGENO inference is depicted in Figure 13.15.

The results presented concern the simulation of the hydraulic axis endowed with a fuzzy controller, which is based on MAMDANI inference [59]. "Fuzzy Logic" toolbox gives the user the possibility to create MAMDANI or SUGENO fuzzy systems using graphic interfaces. FIS (Fuzzy Inference System) Editor, Membership Function Editor, and Inference Rules Editor are several of the tools available in SIMULINK. For instance, the corresponding FIS editor and the Membership Function Editor of each input for the proposed fuzzy controller with MAMDANI inference and two inputs are illustrated in Figures 13.16 and 13.17.

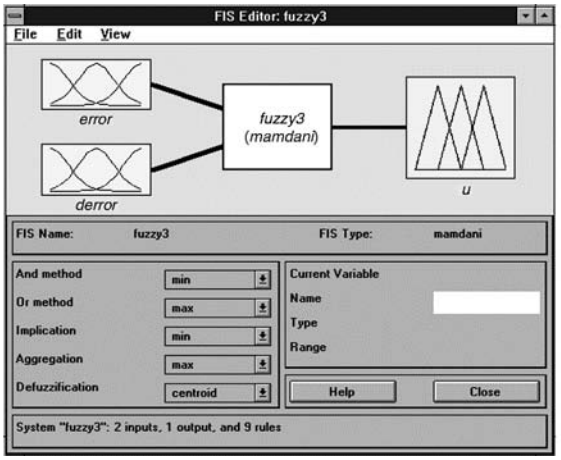


FIGURE 13.16 FIS Editor for fuzzy system based on MAMDANI inference.

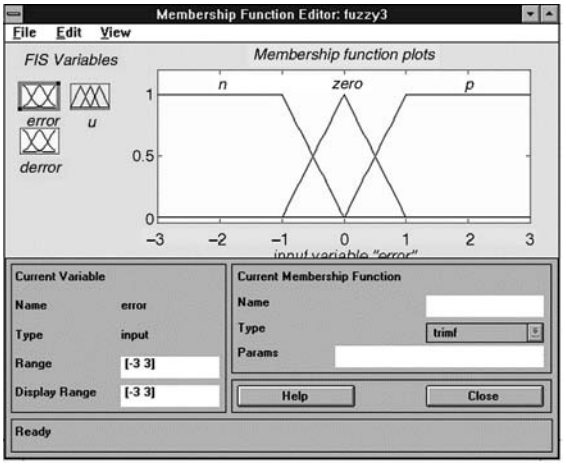


FIGURE 13.17 Membership function associated to the inputs.

For this fuzzy controller there were chosen nine inference rules, which could be visualized using the Inference Rules Editor of SIMULINK. Several simulation results of electrohydraulic axes obtained with the proposed fuzzy controller are shown in Figures 13.18 and 13.19 and depict graphically the position and the velocity of the cylinder.

13.6 Neural Techniques Used to Control the Electrohydraulic Axis

Section 13.6 has as its goals: to emphasize MATLAB's possibilities of using its resources in order to design control systems based on advanced control techniques such as neural networks; to test through simulation these neural algorithms; and to verify performances of the neural control architecture applied to the studied electrohydraulic axis.

There are two main research directions involved in the neural control. One of these implies the developing of one controller going from a neural network, and the other one embeds several controllers

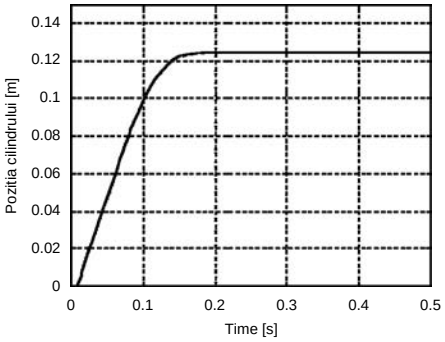


FIGURE 13.18 LHM position.

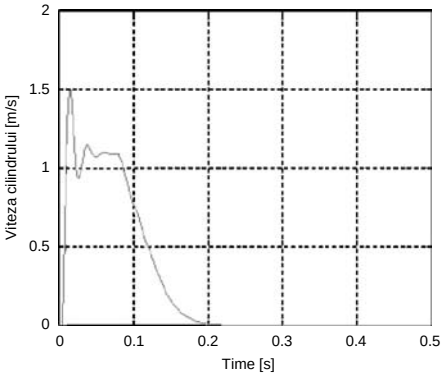


FIGURE 13.19 LHM velocity.

inside a neural network [50]. This section deals with the control of an electrohydraulic axis using a neural controller that has a widely spread structure, namely, multilayer perceptron (MLP).

13.6.1 Neural Control Techniques

13.6.1.1 Learning Based on Mimic

Inspired from biological systems, learning by mimic is applied to control systems. A supervised neural network can mimic the behavior of another system. A first method to develop a neural controller is to replicate a human controller. The neural controller tries to behave like the human operator. Neural training means learning the correspondence between the information received by the human operator and the control input (Figure 13.20).

13.6.1.2 Inverse Learning

The purpose of inverse control is to control a system by using its inverse dynamic. In this case, the neural network receives the output of the system as input and has the input of the system as output. The system works in open loop and has to be in the region where the controller will operate. Inverse learning (Figure 13.21) is an indirect approach to minimize the network output error instead of the overall system error.

13.6.1.3 Specialized Inverse Learning

According to Psaltis, who proposed in 1988 a *specialized inverse learning*, the neural network should be trained online in order to minimize the control error $e_y = r - y$ (see Figure 13.22).

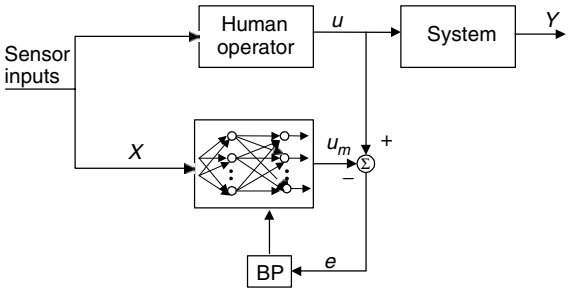


FIGURE 13.20 Diagram for learning based on mimic.

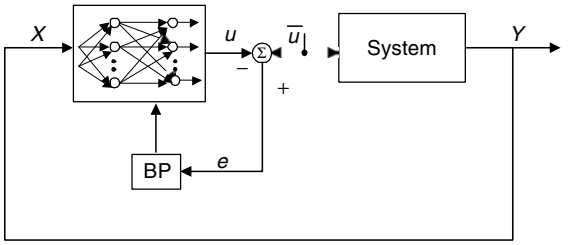


FIGURE 13.21 Training phase at inverse learning.

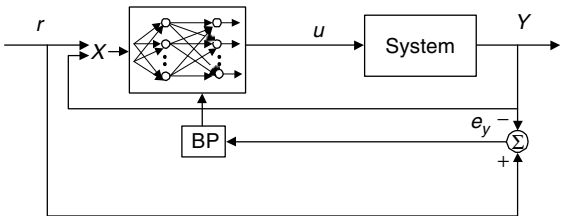


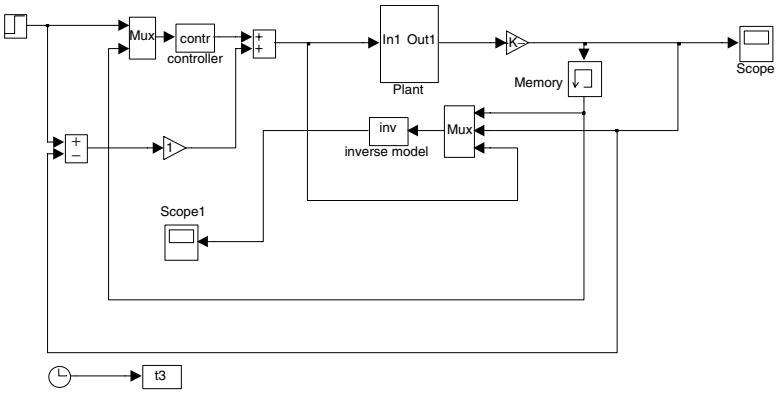
FIGURE 13.22 Specialized inverse control architecture (after [50]).

The neural controller used to control the position of an electrohydraulic axis is a *feed-forward* multilayer neural network, whose learning algorithm is *back-propagation*. In order to adapt the weights which preserve the learned information, two steps are gone through with: a *forward* propagation procedure of the useful signal and a *backward* propagation of the error. The control structure is implemented in SIMULINK as it is shown in Figure 13.23. The neural control of the electrohydraulic axis and the achievement of controller parameters are performed online.

A neural network with four layers, having two neurons on the first layer, a neuron on the last layer, and five neurons on each hidden layer, is proposed. The graphic characteristic corresponding to the axis position and obtained using the neural network described above is illustrated in Figure 13.24.

13.7 Neuro-Fuzzy Techniques Used to Control the Electrohydraulic Axis

This chapter deals with several computer-aided design techniques of hybrid control algorithms. This paper concentrates on these types of algorithms, because the performances achieved through simulation of an electrohydraulic axis with a neuro-fuzzy controller are comparable or superior to those yielded by other control algorithms. Taking into account the novelty of neuro-fuzzy algorithms and the absence in



FIGUGE 13.23 The control structure for proposed controllers.

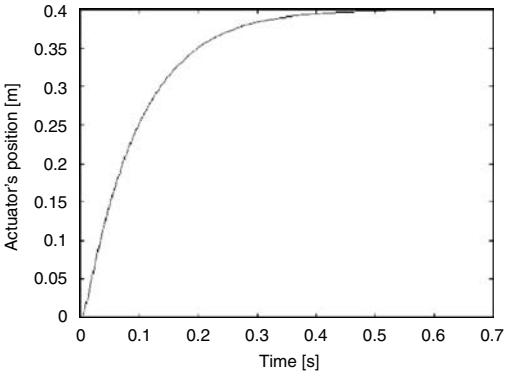


FIGURE 13.24 The axis position for $U = 8$ V input voltage.

SIMULINK of a toolbox devoted to them, the research was oriented to the achievement of a library of C++ programs, which can afford the use of SIMULINK in the design of such controllers. Thus, online adaptation procedures of fuzzy controller parameters are implemented. The comparative study of different classic and advanced algorithms is performed on the basis of integral squared error computed on the transitory horizon.

Because of the capability of fuzzy systems to treat imprecise information, they are strongly recommended in order to express knowledge in the form of linguistic rules. In this way, the human operator's knowledge, which is linguistic or numerical, is used to generate the set of fuzzy if-then rules as a basis for a fuzzy controller. A main drawback of fuzzy systems is the difficulty to design them based on a systematic methodology. To overcome this drawback, the learning procedures from neural networks are successfully applied in order to tune the parameters of membership functions. The merger of neural networks and fuzzy logic has led to the existence of neuro-fuzzy controllers. It can be asserted that neuro-fuzzy controllers embed essential features of both fuzzy systems and neural networks.

The proposed neuro-fuzzy controller has a structure based on the Takagi-Sugeno method and it is depicted in Figure 13.25.

A learning procedure in fact represents a parameter estimation problem. The learning procedure for the proposed neuro-fuzzy controller is gradient-descendent. The method applied to design such a controller is called inverse learning in which an online technique is used to model the inverse dynamics of the plant. The obtained neuro-fuzzy model—the inverse dynamics of the plant—is used to generate control actions.

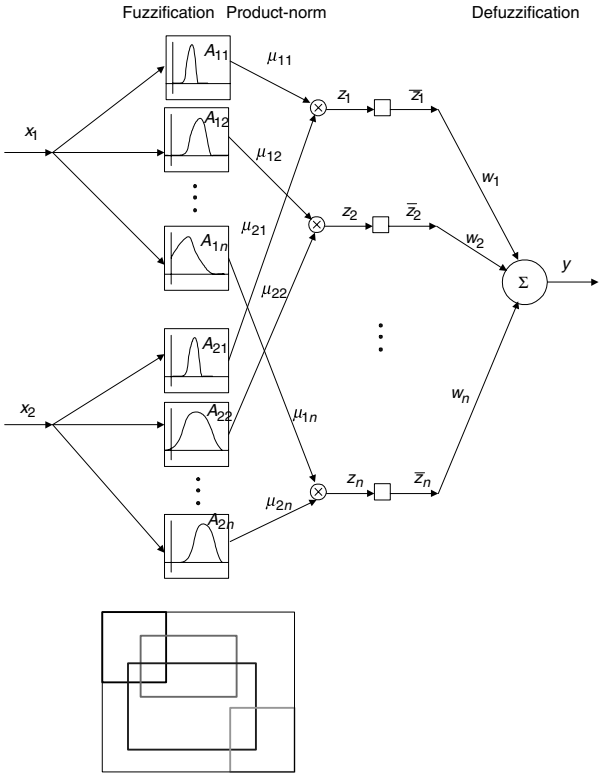


FIGURE 13.25 Structure of the neuro-fuzzy controller.

The neuro-fuzzy controller is a multilayer connectionist system, a multi-input and single-output fuzzy logic system. The network has three layers: one input layer with $n \times m$ units, one hidden layer with n units, and one output layer with one unit [15]. The partition used for this model is a scatter partition [33].

Figure 13.25 presents a particular case where the fuzzy controller has only two inputs and one output. In a general case, the fuzzy controller has m inputs and one output.

The fuzzy rule base contains a set of n linguistic rules in the form:

R_i: If x_1 is A_{1i} and x_2 is A_{2i}
and…
and x_m is A_{mi}
then y is w_i, $i = 1, 2, \ldots, n$

where i is the index of the rule; A_{ji} is a fuzzy set for the jth linguistic variable and the ith rule; and w_i is a number that represents the consequent part.

The membership functions assigned to each input are Gaussian functions. The centers of the membership functions are chosen such that these functions are uniformly distributed over the universe of discourse:

$$\mu_{ji} = e^{-(x_j - a_{ji})^2 / 2b_{ji}^2} \tag{13.26}$$

The fuzzy inference involved in this neuro-fuzzy controller is the product operator T-norm defined as an *and* conjunction. The firing strength of every rule is

$$z_i = \mu_{1i} \cdot \mu_{2i} \cdot \ldots \cdot \mu_{mi}, \quad i = 1, \ldots, n \tag{13.27}$$

The output is a crisp value obtained as a result of the evaluation of a center of gravity:

$$y = \frac{\sum_{i=1}^{n} z_i \cdot w_i}{\sum_{i=1}^{n} z_i} = \frac{z_1}{\sum_{i=1}^{n} z_i} \cdot w_1 + \frac{z_2}{\sum_{i=1}^{n} z_i} \cdot w_2 + \cdots + \frac{z_n}{\sum_{i=1}^{n} z_i} \cdot w_n = \sum_{i=1}^{n} z_i \cdot w_i \tag{13.28}$$

The parameters to be estimated are obtained by finding the minimum of the following cost function:

$$J(k) = \frac{1}{2} \cdot (y(k) - y_d(k))^2 \tag{13.29}$$

where $y_d(k)$ is the desired output and $y(k)$ is the obtained response at time k.

To minimize this cost function, the stochastic approximation method is used. The learning procedure means the estimation of parameters and is based on the least-mean square algorithm. The parameters to be estimated are

$$p = (a_{11}, \ldots, a_{nm}, b_{11}, \ldots, b_{nm}, w_1, \ldots, w_n) \tag{13.30}$$

The equations to adapt the parameters are the following:

$$\begin{aligned} a_{ji}(t+1) &= a_{ji}(t) - \lambda_a \frac{z_i}{\sum_{l=1}^{n} z_1} \cdot (y - y_d) \cdot (w_i - y) \cdot \frac{x_i - a_{ji}(t)}{b_{ji}^2} \\ b_{ji}(t+1) &= b_{ji}(t) - \lambda_b \frac{z_i}{\sum_{l=1}^{n} z_1} \cdot (y - y_d) \cdot (w_i - y) \cdot \frac{(x_j - a_{ji}(t))^2}{b_{ji}^3} \\ w_i(t+1) &= w_i(t) - \lambda_w \frac{z_i}{\sum_{l=1}^{n} z_1} \cdot (y - y_d) \end{aligned} \tag{13.31}$$

where the learning factors λ_a, λ_b, λ_w are predefined.

In the learning process, parameters that could be modified are (a_{ji}, b_{ji}) which describe Gaussian functions, and w_i, the conclusion values. If the structure of the membership function is established, the only values that could be modified are w_i.

13.7.1 Control Structure

In order to design the neuro-fuzzy controller proposed above, the inverse learning method is applied. The control of an electrohydraulic axis involves the use of an online technique to model the inverse dynamics of the plant. The block diagram for online inverse learning is presented in Figure 13.26.

This scheme is in open loop and it is also found by the controller output error method (COEM) [1] to online tune or adapt the parameters of a fuzzy controller. This method does not require the plant output error to be propagated at the input. There is another constraint, namely the controller has to be capable of stabilizing the plant before the commencement of tuning. To avoid this requirement, a modified COEM (MCOEM) [2] is used. The diagram block in this case is depicted in Figure 13.27.

A proportional feedback controller P is introduced and in this situation the plant input is the sum of $u'(k)$ and $u_p(k)$. The consequent singletons are initialized to zero and the controller P is chosen in such a way that it stabilizes the plant. The structure and the parameters of inverse model and of neuro-fuzzy controller are identical.

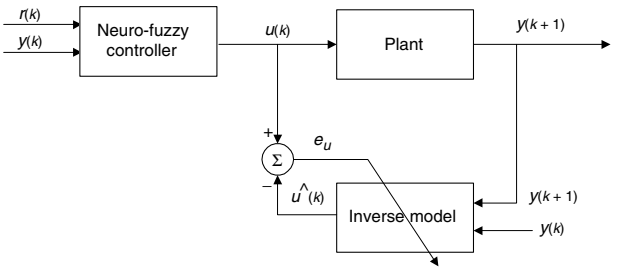


FIGURE 13.26 Diagram of control based on inverse learning.

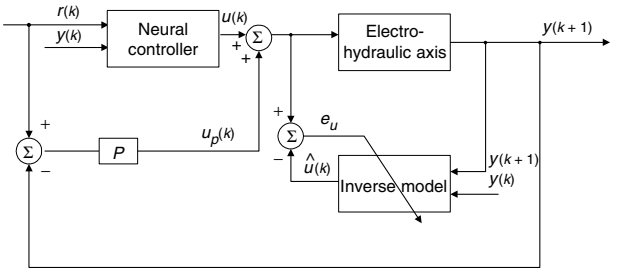


FIGURE 13.27 Block diagram for inverse learning with proportional controller.

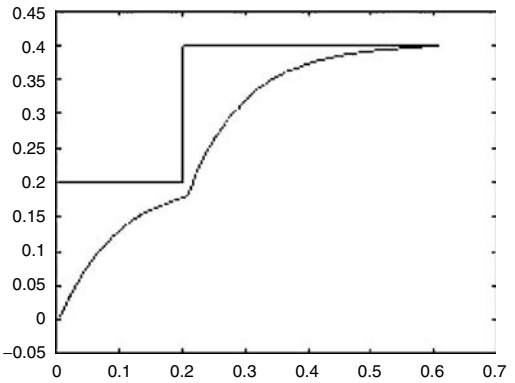


FIGURE 13.28 The position control with neuro-fuzzy controller.

There are two phases in the design of such a controller: the control and the adaptation. In the control phase, the plant output and the reference signal determine a control command $u(k)$. The plant input becomes $u(k)$, the sum of the $u(k)$ and $u_p(k)$. In the adaptation phase, the inverse model, which has as inputs $y(k+1)$ and $y(k)$, produces the signal $\hat{u}(k)$ as an output. This signal is used to compute the error $e_u(k)$, which determines the value of the cost function $J(k)$ that has to be minimized.

$$J(k) = \frac{1}{2} \cdot e_u^2(k) = \frac{1}{2} \cdot (u(k) - \hat{u}(k))^2 \tag{13.32}$$

This procedure was used at the control of the electrohydraulic axis position, where the controller parameters are determined online. The actuator position obtained when the reference signal is changed from $U = 4$ V to $U = 8$ V is depicted in Figure 13.28.

TABLE 13.1 Results Obtained by Applying IAE and ISE

Regulator	IAE	ISE
PID (chapter 4)	0.8042	3.4754
PI (chapter 4)	0.8006	3.4618
PD (chapter 4)	0.7928	3.4537
Neural (chapter 6)	0.8027	3.4622
Neuro-fuzzy (chapter 7)	0.7911	3.4501

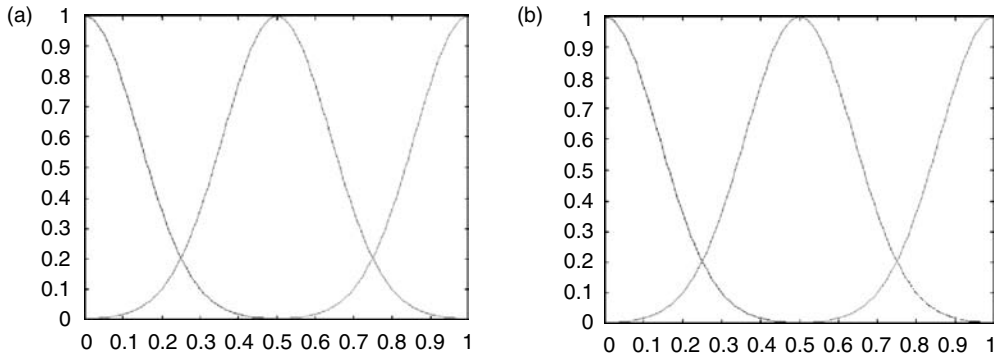


FIGURE 13.29 (a) Membership functions before learning for the variable x_1. (b) Membership functions before learning for the variable x_2.

In order to achieve a comparison of the modern control algorithms (included in this thesis) to the conventional structures, two spread integral criteria, namely, the integral of absolute error (IAE) performance index and the integral of squared error (ISE), are used. The results obtained applying these criteria are included in Table 13.1.

According to previous results, it can be inferred that the described neuro-fuzzy controller exhibits superior performances compared to those obtained with the neural controller based on MLP, or with the classic controllers (PID, PI, PD with filtering) presented in this chapter. The simulation results emphasize the neuro-fuzzy controller, arguing that it represents a very useful tool for practical applications with many nonlinearities.

Optimized results were obtained through variation of data sets and number of iterations. In order to test the performance of the proposed neuro-fuzzy controller, one nonlinear function given by an analytical equation was approximated. The membership functions of input variables x_1 and x_2 before learning are shown in Figure 13.29a,b. The surface obtained after simulation is depicted in Figure 13.30c. One may observe the accuracy of the reconstruction after 300 learning iterations by comparison with the surface to be obtained.

Sets of intermediary results obtained with different simulation data sets are presented below. Different data sets of simulations were used in order to achieve optimized results. Some of them are presented in Figures 13.30–13.34 without comment.

In order to obtain good performances from the model, 10 membership functions are used for each input variable. The learning factors λ_a, λ_b, λ_w were chosen as 0.01. The control algorithm is capable of handling the change in operating range. The results of the electrohydraulic axis simulation with the proposed neuro-fuzzy controller are obtained for various inputs. Those in time domain, results presented in Figure 13.35a,b, correspond to input voltages of 8 and 10 V.

1st set: gw = 0.1; ga = 0.05; gb = 0.05; nepoc = 100; nesant = 100; niter = 200; threshold error = 0.001; Vmax = 0.04.

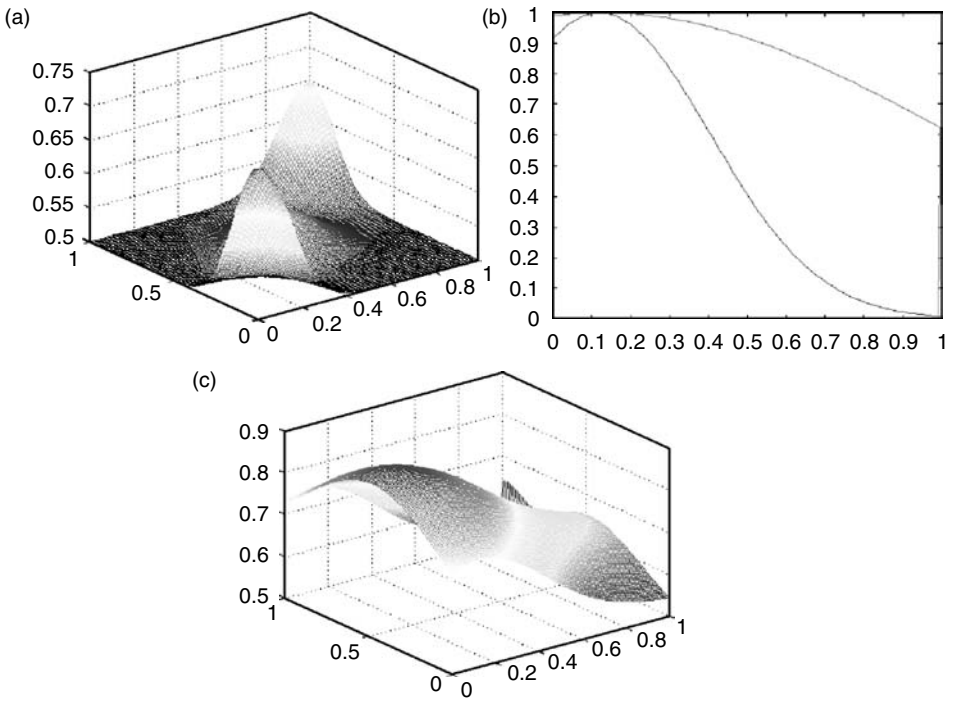


FIGURE 13.30 (a) The surface obtained after the first iteration, (b) membership functions after learning for the variable x_2, (c) the surface obtained after simulation.

2nd set: gw = 0.1; ga = 0.07; gb = 0.05; nepoc = 100; nesant = 21; niter = 200; threshold error = 0.001; $V_{max} = 0.0475$.

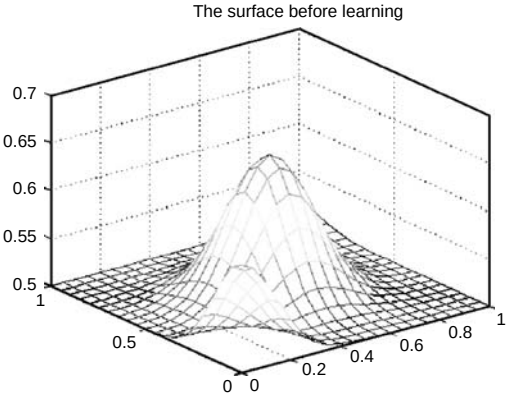


FIGURE 13.31 The surface obtained after the first iteration.

3rd set: gw = 0.5; ga = 0.07; gb = 0.03; nepoc = 200; nesant = 21; niter = 200; threshold error = 0.001; Vmax = 0.047515.

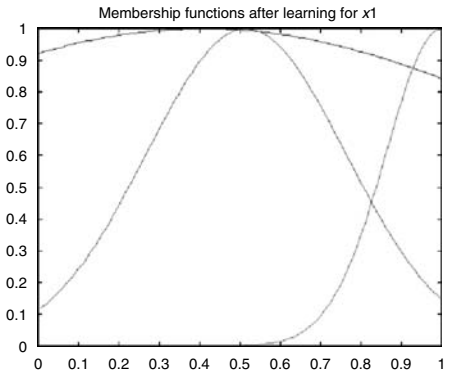


FIGURE 13.32 Membership functions after learning for the variable x_2.

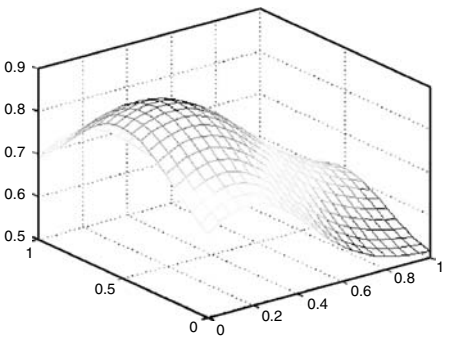


FIGURE 13.33 The surface obtained after learning.

13.8 Software Considerations

The MM of electrohydraulic axis studied in this thesis is supported by a physical installation existing in the mechatronics laboratory of UAS-Konstanz (see Figure 13.1b). Two variants of nonlinear MM are set forth in Section 13.3 and add in static and dynamic nonlinearities that arise in the function of electrohydraulic axis [23,58].

The MM of hydraulic drive presented in the structure of ROBI_3 was implemented in SIMULINK in order to study the dynamic behavior of the axis [26,27]. The extended variant of MM hydraulic axis was done taking into account the relative motion of the constituent parts of this servodrive.

The neural and neuro-fuzzy controller (Takagi-Sugeno) was developed in Borland C++ and implemented in SIMULINK for controlling the electrohydraulic axis. SIMULINK offers the user a FUZZY LOGIC library that allows the designing and modeling of SUGENO or MAMDANI fuzzy inference systems. The lack of a dedicated software to design neuro-fuzzy controllers persuaded the implementation of such a controller in C++ and afterwards the use of it in SIMULINK [26–28].

The support for simulation, SIMULINK 2.1 and MATLAB 5.2 (under Windows), offers solutions to implement our controllers as modules and corresponding icon in a specialized toolbox. In our experiments, we used the facility offered by S-functions and C MEX in conjunction with Borland C++ 5.0 to compile them. We have chosen the C S-function because of the speed necessary to process the information in our block that implement the controller. The block that implements the controller has two inputs (even three inputs are available, though the adaptation process is more complicated) and one output.

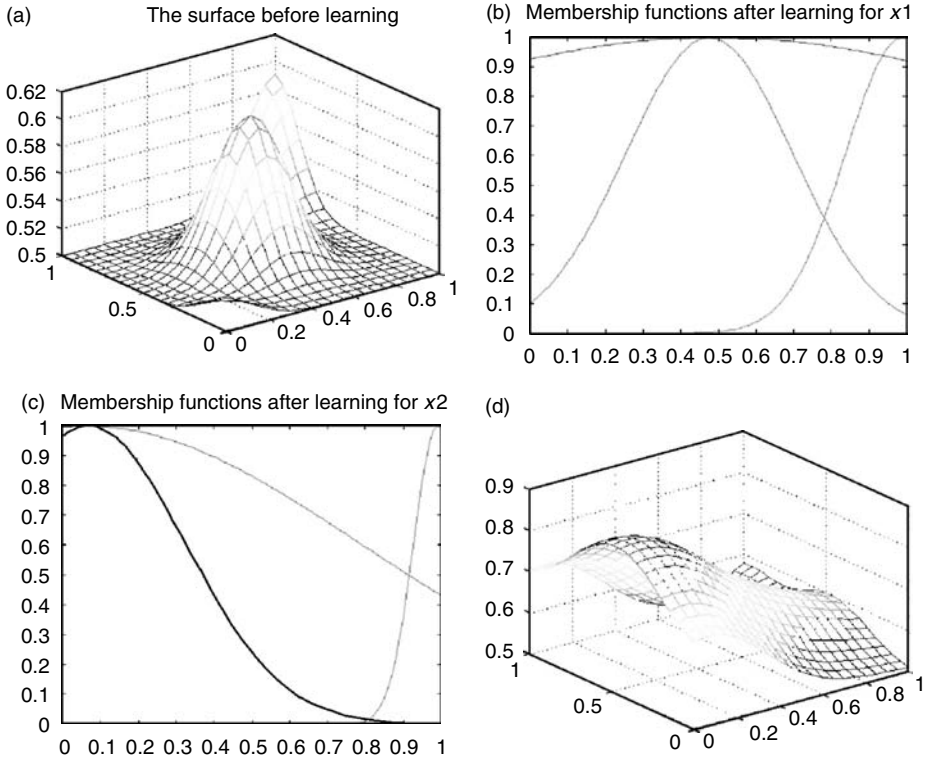


FIGURE 13.34 (a) The surface obtained after first iteration, (b) membership functions after learning for the variable x_1, (c) membership functions after learning for the variable x_2, (d) the surface after learning.

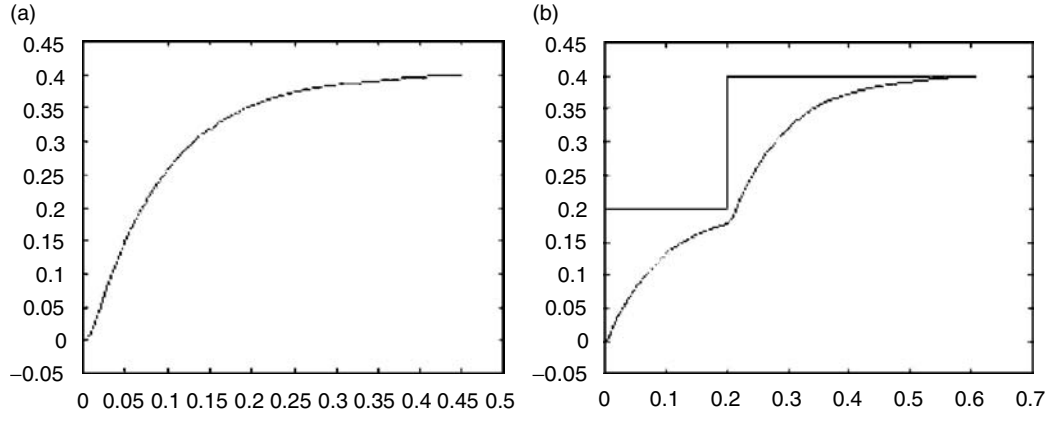


FIGURE 13.35 (a) The position control with neuro-fuzzy controller ($U = 8$ V), (b) the position control with neuro-fuzzy controller ($U = 10$ V).

The adapting parameters (weights, centers, and spread for Gaussian function) must be persistent. Declaring global, static or using the workspace in order to store the are useful techniques to accomplish the task.

The newest version of Simulink offers the possibility to write wrapper S-function, to use the callbacks functions, and as an alternative ADA or Fortran programming language.

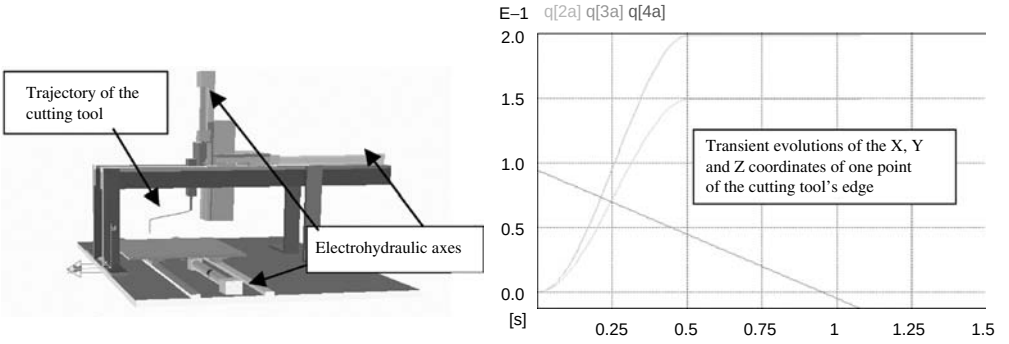


FIGURE 13.36 3-Axes Cartesian robot modeled, controlled, and simulated in SDS [21,24].

13.9 Conclusions

The research achieved as part of this chapter has, as an essential purpose, the development of improved control structure based on advanced techniques (neural and fuzzy), in relation to the conventional one. The studied electrohydraulic axis is a component of the Cartesian robot ROBI_3 implemented in the mechatronics laboratory of UAS-Constance. 3D-simulations with direct and inverse dynamics and implemented controllers by using the SDS-Modelling and Simulation software of the real installation were performed [21,23,25,28]. Model and simulation results are presented in Figure 13.36.

As an overview, Section 13.1 deals with the introduction of this chapter. In Section 13.2, the most important aspects of electrohydraulic system control and of nonlinearities that arise with this type of installation are pointed out. The robot, ROBI_3, is presented from both a component and control perspective.

The mathematical model (MM) of ROBI_3's hydraulic axis is described in Section 13.3. The nonlinear MM is achieved based upon technical data of different components of installation and also taking into account theoretic assessments of electrohydraulic installation functionality. Simulation results of nonlinear MM placed into a position loop are obtained with simulation environment SIMULINK/MATLAB. As a general remark, it should be mentioned that all simulations included in this chapter are achieved by using MATLAB/SIMULINK, while the advanced control algorithms (neural and neuro-fuzzy) are developed in Borland C++ 5.0.

Section 13.4 contains a short overview of the theory devoted to conventional controllers PID, PD, PI, and observer, followed by simulation results of the electrohydraulic axis endowed with the above control structures.

Following the scientific goal of this contribution, Section 13.5 reviews fuzzy system theory and presents simulations of electrohydraulic axis with fuzzy controllers. Fuzzy system theory has contributed greatly to system modeling, and the development of a theoretical frame appropriate to implement the qualitative reasoning specific to human beings. This kind of reasoning is very useful to model complex systems, which are characterized by nonlinearities or imprecise information. Simulation results of hydraulic axis are obtained using SUGENO and MAMDANI fuzzy controllers.

A short introduction of neural networks theory, the most widespread neural structures and also neural control techniques are presented in the beginning of Section 13.6. Neural networks work quantitatively, numerically. If fuzzy logic has an inference based on uncertainty, then neural networks learn by training, at the end of which the network will approximate a desired function. The analysis of trained NN involves many challenges, and as a result, the rules are usually not extracted from trained NN. Simulation results included in Section 13.6 are obtained with a multilayer NN.

Neuro-fuzzy systems preserve the characteristics of NN and also of fuzzy systems, and have been used successfully in control in recent years. Section 13.7 is devoted to the neuro-fuzzy system theory, to the presentation of neuro-fuzzy controller implemented in Borland C++ and applied in SIMULINK to

electrohydraulic axis, to the simulation results achieved in this case, and to the comparative study of conventional and modern controllers.

Section 13.8 contains a concise presentation of this chapter, the main contributions to the subject area presented, as well as a listing of perspective areas of interest in order to pursue further research in this direction.

Without intending to confine the parameters of this chapter, following is a listing of possible research directions and development perspectives that may be followed in future research endeavors:

- Applying various controllers implemented in SIMULINK not only to control the electrohydraulic axis discussed, but also for systems with very complex structure which are involved in large hydraulic installations, offering the user a neuro-fuzzy controller's library;
- The hardware implementation of described neuro-fuzzy controller;
- Continued research in the development of an optimal controller, systemically based (through the further study of stability utilizing linear matrix inequalities—LMI);
- The integration of presented controllers in software packages dedicated to hydraulic and pneumatic fields (for instance in HYPAS[23], DSH, etc.);
- The development of controller design in order to promote those controllers, which allow a better symbiosis between classical and advanced methods (neuro-fuzzy, genetic algorithms);
- The extension of preoccupations and extrapolation of research results regarding control of velocity, acceleration, pressure, flow, force, moment, and power.

References

1. Andersen, H.C., Lotfi, A., Tsoi, A.C. A new approach to adaptive fuzzy control: the controller output error method, *IEEE Trans. on Systems, Man, and Cybernetics,* SMC-27-B(4), August 1997.
2. Abonyi, J., Nagy, L., Szeifert, F. Indirect adaptive Sugeno fuzzy control, *Proceedings in Artificial Intelligence,* FNS'98, München, Germany, march 19–20.
3. Backé, W. *Systematik der hydraulischen Widerstandsschaltungen in Ventilen und Regelkreisen.* Krauskopf-Verlag, Mainz, 1974.
4. Costa Branco, P.J., Dente, J.A. *Inverse-Model Compensation Using Fuzzy Modeling and Fuzzy Learning Schemes.* Intelligent Engineering Systems through Artificial Neural Networks, Smart Engineering Systems: Fuzzy Logic and Evolutionary Programming, Ed. C.H. Dagli, M. Akay et al., Vol. 6, ASME Press, New York, pp. 237–242, 1996.
5. Brown, M., Harris, C. *Neuro-Fuzzy Adaptive Modelling and Control.* Prentice-Hall, Englewood Cliffs, NJ, 1994.
6. Catana, I., Vasiliu, D., Vasiliu, N. *Servomecanisme electrohidraulice. Constructie, functionare, modelare, simulare si proiectare asistata de calculator.* U.P.B. Bucuresti, 1995.
7. Cybenko, G. *Mathematical Problems in Neural Computing.* Signal Processing Scattering and Operator Theory and Numerical Processing, M.A. Kashoek, J.H. van Schupper, A.C. Ram, 1989. Vol. 3. pp. 47–64.
8. Driankov, D., Hellendoorn, H., Reinfrank, M. *An Introduction to Fuzzy Control.* Springer-Verlag, Berlin, 1993.
9. Dubois, D., Prade, H., Ughetto, L. Checking the coherence and redundancy of fuzzy knowledge bases, *IEEE Trans. on Fuzzy Systems,* 5(5):398–417, 1997.
10. Dumitrache, I. sa. *Automatizari Electronice.* Editura Didactica si Pedagogica, Bucuresti, 1993.
11. Dumitrache, I., Catana, I., Militaru, A. *Fuzzy Controller for Hydraulic Servosystems.* IFAC International Workshop on Trends in H&P Components & Systems, Chicago, IL, 1994.
12. Föllinger, O. *Regelungstechnick.* Dr. A. Hütig Verlag, Heidelberg, Germany, 1978.
13. Friedrich, A. *Logik und Fuzzy-Logik.* Expert-Verlag, 1997.
14. Ghaoui, L. El. Reduced-order multimodel control using linear matrix inequalities: sufficient conditions, *Proc. Od ACC 1993,* pp. 633–634, 1993.

15. Godjevac, J., Steele, N. Adaptive neuro-fuzzy controller for navigation of mobile robot, *International Symposium on Neuro-Fuzzy Systems AT'96,* Conf. Report, EPFL-Lausanne, 1996.
16. Gupta, M.M. *Fuzzy Logic and Neural Networks,* Proc. of the 2nd International Conference on Fuzzy Logic & Neural networks, Iizuka, Japan, 17–22 July, pp. 187–188, 1992.
17. Healey, M. *Principles of Automatic Control.* The English Universities Press Ltd., 1975.
18. Haykin, S. *Neural Networks*, MacMillan College Publishing Company, New York, 1994.
19. Ionescu, Fl. *Computer aided design of hydraulic and electrohydraulic drive installations*, Proc. of the9th Triennial World IFAC Congress, Budapesta, Ungaria, Pergamon Press, Vol. 1, pp. 569–574, 1984.
20. Ionescu, Fl., Stoffel, B. *Contribution to the Automatic Generation of Mathematical Models for the Computer Assisted Analysis and Synthesis of Hydraulic Drive Systems.* Proc. of the 2nd Intern. Conf. on Fluid Power, Tampere, Finland, 19–21 March, pp. 469–482, 1991.
21. Ionescu, Fl., Haszler, Fl. *TORCH: A Control Software for Electrohydraulic Cartesians Robots.* Proceed. of the 6th Intern. IMEKO Symposium on Measurement and Control in Robotics, ICMR'96, Bruxelles, Belgium, 9–11 May, pp. 484–489, 1996.
22. Ionescu, Fl. *Non-Linear Problems in the Hydraulic Drive Systems.* 2nd World Congress of Nonlinear Analysts, Athena, Greece, 10–17 July 1996, Pergamon Press, Vol. 30, part 3, pp. 1447–1461.
23. Ionescu, Fl., Vlad, C.I. Tools of HYPAS for the control of electrohydraulic drive installations, *Proc. of 7th Symposium on Computer Aided Control Systems Design,* Gent, Belgia, 1997, pp. 311–316.
24. Ionescu, Fl., Borangiu, Th., Vlad, C.I. High integrated CAD strategies for control design of electro-hydraulic systems, *Proc. 3rd IFAC Conference SSC,* Bucharest, pp. 390–395, 1997.
25. Ionescu, Fl., Vlad, C.I. *Hypas tools for the control of electro-hydraulic drive installations Journal* ***a,*** Vol. 38, No. 3, Belgium, pp. 38–41, 1997.
26. Ionescu, Fl., Vlad, C.I. Sugeno and hypas fuzzy-control solutions for electro-hydraulic drive installations, *Proceedings EUFIT'97,* Aachen, Germany, 8–11 Sept., Vol. 2, pp. 1238–1242, 1997.
27. Ionescu, Fl., Vlad, C.I., Arotaritei, D. Fuzzy and neuro-fuzzy HYPAS controllers implemented for an electro-hydraulic axis, *International ICSC Symposium on Engineering of Intelligent Systems EIS'98,* Teneriffe, Spain, Feb. 11–13, 1998.
28. Ionescu, Fl., Arotaritei, D., Vlad, C.I. *Modelling of Nonlinearities, Signal Reconstruction and Predictive Solutions Applied in Mechatronics Systems by Using Neuro-Fuzzy Systems,* Internal Report, Department of Mechatronics, FH-University of Applied Sciences-Konstanz, 1998.
29. Ionescu,V., Varga, A. *Teoria Sistemelor. Sinteza robusta. Metode numerice de calcul.* Editura ALL, Bucuresti, 1995.
30. Isermann, R. *Digitale Regelsysteme*, Springer-Verlag, Berlin, 1987.
31. Isermann, R. *Zur Anwendung der Fuzzy-Logik in der Regelungstechnik.* Automatisierungstechnische Praxis (atp) Fuzzy-Control, 38, Oldenbourg Verlag, Germany, 1996.
32. Isermann, R. On fuzzy logic application for automatic control, supervision, and fault diagnosis, *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans,* Vol. 28, No. 2, March 1998, pp. 221–235.
33. Jang, J.-S.R., Sun, C.-T., Mizutani, E. Neuro-fuzzy and soft computing, *A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
34. Joh, J., Hong, S.K., Nam, Y., Chung, W.J. On the systematic design of Takagi-Sugeno fuzzy control systems, *International ICSC Symposium on Engineering of Intelligent Systems EIS'98,* Tenerife, Feb. 1998.
35. Kandel, E.R. *Nerves Cell and Behavior,* Principles of Neural Sciences, 3rd ed., pp. 18–36, 1992.
36. Knappe, H. Comparison of conventional and fuzzy-control of non-linear systems, In: Kruse, R., *Fuzzy Systems in Computer Science*, Verlag Vieweg, Wiesbaden, Germany, 1994.
37. Kokotovic, P.V. *Lectures Notes in Control and Information Sciences.* Springer-Verlag, Berlin, 1991.
38. Kosko, B. *Neural Networks and Fuzzy Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1992.
39. Kovacic, Z., Balenovic, M., Bogdan, S. Sensitivity-based self-learning fuzzy logic for a servo-system, *IEEE Control Systems,* June, 1998.

40. Lippmman, R. An introduction to computing with neural nets, *IEEE ASSPMagazine,* April 1987, pp. 4–22.
41. MATLAB 5.2. MathWorks Corp, USA. 1998.
42. Miller, Th., Sutton, R., Werbos, P.J. *Neural Networks for Control.* MIT Press, 1990.
43. Nauck, D., Klawonn, F., Kruse, R. *Neuronale Netze und Fuzzy-Systeme.* Grundlagen des Konnektionismus, Neuronaler Fuzzy-Systeme und der Kopplung mit wissensbasierten Methoden. Vieweg, 1994, Germany.
44. Nesterov, Y., Nemirovski, A. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM, Philadelphia, 1994.
45. Pedrycz, W. *Fuzzy Control and Fuzzy Systems*, Wiley, New York, 2nd ed., 1993.
46. Piechnik, M., Feuser, A. *Simulation mit Komfort–HYVOS 4.0 und MOSIHS 1.0,* Ö & P, 38, 1994.
47. Postlethwaite, B.E. A model-based fuzzy controller, *Trans IChemE,* Vol. 72, Part A, Jan. 1994.
48. Postlethwaite, B.E. Building a model-based fuzzy controller, *Fuzzy Sets and Systems,* 79 (1996), Elsevier.
49. Rehfeldt, K., Shöne, A., Büngener, N. *Einsatz von Fuzzy-Reglern zur Drehzahlregelung einer Hydraulikpumpe, Ölhydraulik und Pneumatic,* 36, Nr. 6, pp. 397–402, 1992.
50. Ronco, E., Gawthrop, P.J. *Neural Networks for Modelling and Control.* Technical Report: csc97008, Centre for System and Control, Dept. of Mechanical Engineering, Univ. of Glasgow, 10 Nov. 1997.
51. Simulink, Dynamic System Simulation for MATLAB, Writing S-functions, The Math Works Inc., 1998.
52. Sontag, E.D. Mathematical control theory, *Deterministic Finite Dimensional Systems.* Springer-Verlag, Berlin, 1990.
53. Takagi, T., Sugeno, M. Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems, Man, and Cybernetics,* Vol SMC-15, No. 1, pp. 116–132, 1985.
54. Tanaka, K., Sugeno, M. Stability analysis and design of fuzzy control systems, *Fuzzy Sets and Systems,* Vol. 45, pp. 135–156, 1992.
55. Teodorescu, H.N. *Sisteme Fuzzy si Aplicatii.* Institutul Politehnic Iasi, Romania, 1989.
56. Tertisco, M., Penescu, C., Ionescu, G., Ceanga, E. *Identificarea Experimentala a Proceselor Automatizate.* Editura Tehnica, Bucuresti, 1971.
57. Viersma, T. J. *Analysis, Synthesis and Design of Hydraulic Servosystems and Pipelines.* Elsevier, Amsterdam-New York, 1980.
58. Vlad, C.I. Contributions to the Direct Computer Control of Electrohydraulic Axes for Industrial Robots. Technical University "Politehnica," Bucharest, Romania, 1998.
59. Wang, L., Liu, G.P., Harris, C.J., Brown, M. *Advanced Adaptive Control*, Pergamon, 1997.
60. Werbos, B. *Overview* of *Design and Capabilities.* In Neural Networks for Control, pp. 59–65, MIT Press, MA, 1990.
61. Westcott, J.H. The minimum-moment-of-error-squared criterion: a new performance criterion for servo mechanisms, *Proc. of IEE.,* Measurements Section, pp. 471–480, 1954.
62. Yager, R., Zadeh, L. *Fuzzy Sets, Neural Networks and Soft Computing*, 1994.
63. Zadeh, Lotfi. *Fuzzy sets, Information & Control*, No. 8, pp. 338–353, 1965.
64. Zadeh, L., King-Sun Fu, Tanaka, K., Shimura, M. *Fuzzy Sets and their Applications to Cognitive and Decision Processes.* Academic Press, 1975.
65. Zimmermann, H.-J. *Fuzzy Sets Theory—and Its Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

14

Design Optimization of Mechatronic Systems

Tomas Brezina
Ctirad Kratochvil
Cestmir Ondrusek
Technical University of Brno

14.1 Introduction

Electromechanical systems form an integral part of mechanical and mechatronic systems. Their optimization is a necessary condition for a product to be competitive. In engineering practice, a large number of optimization and identification problems exist that could not be solved without the use of computers [5]. The present level of technological development is characterized by increasing the performance of machines with the production costs kept at a satisfactory level. The demands on the reliability and safety of operation of the designed machines are also considerable.

From practical experience we know that the dynamic properties of electromechanical systems have a considerable influence on their reliability and safety. On the other hand, the tendency to push the price of a machine down often leads to unfavorable dynamic properties that result in increased vibrations and noise during operation. Also, electrical properties dramatically deteriorate as the amount of active materials in a machine is reduced. The increased load leads to, among other things, excessive heat formation, which, in turn, has a negative effect on insulation, shortening the service life of a machine.

14.2 Optimization Methods

14.2.1 Principles of Optimization

The properties of electromechanical systems can be described mathematically using physical quantities. The degree of these properties is then described using mathematically formulated objective (preference) functions. Structural parameters ranging between limit values given as satisfying secondary conditions are the independent variables of these functions. The particular form of the functions depends on the type of machine and its mathematical description. The solutions of a mathematically formulated optimization

problem together with optimization methods allow a considerable number of different design variants of a machine to be calculated in a relatively short time. They also make it possible to perform these calculations at production planning stages for a prototype to possess the qualities given by a chosen criteria function. In this way, the design of a machine is not only analyzed but also modified and reconstructed in terms of its electromechanical properties with the aim to improve these properties as much as possible (or optimize them).

From a physical point of view, these are actually problems that, to a certain degree, are inverse to those of calculus. A problem of calculus assumes a fixed, mathematically described model of a real machine to be used for deriving its resulting properties. In problems of calculus, we define properties and try to find out which parameters of a chosen mathematical model possess those properties. In problems of parametric optimization, we look for those parameters that, by a chosen preference function, provide the best properties. It is clear that problems of synthesis and optimization are much more sophisticated than those of calculus.

14.2.2 Parametric Optimization

As the aim is to find the values of certain structural parameters of a machine, we shall deal with this notion in more detail. By a parametric optimization of electromechanical systems, we mean the process of finding those parameters of a mathematical description of the system (arranged in a vector $\bar{p}$) from a set P of admissible parameters at which a suitably selected objective (preference) function $\psi(p)$ of these parameters reaches its extreme.

The objective function $\psi(p)$ quantifies the degree of the properties of an electromagnetic system that has to be made extreme (the parameters with the best degree of this property have to be chosen). When defining an admissible set P, we are guided by the structural possibilities of changes in individual parameters (variables), or we can introduce secondary criteria of the type "the degree of properties may not exceed given critical limits." The possibility of taking into consideration the structural changes of parameters leads to the so-called trivial (natural) constraints of the type $p_i^d \le p_i \le p_i^h$, where p_i^d is the lower and p_i^h the upper bound of the ith optimization variable. The introduction of secondary criteria leads to the definition of limiting functions q_i of optimization variables for which we have $q_i^d \le q_i(p) \le q_i^h$, where q_i^d is the lower and q_i^h is the upper bound of the relevant function.

Thus, from the mathematical point of view, parametric optimization of electromechanical systems is formulated as the problem of finding a point p in the admissible set P, at which the preference function ψ reaches its global extreme value (maximum or minimum) with regard to P. The admissible set is generally described by m inequalities defined by functions $q_j(p)$, where $j = 1, 2, \ldots, m$. If $P = R^s$, where s is the number of variables to be optimized, we say that the optimization is unconditional. In all other cases we say that the optimization is conditional.

To solve the problem of optimizing the selected properties of a system, the following has to be done:

- A mathematical description has to be formulated.
- It has to be analyzed at the starting point.
- The desired form of the objective function ψ has to be specified.
- The optimization variables have to be selected.
- The desired form of the constraining functions q_j has to be specified.
- A suitable optimization method has to be selected.
- The resulting mathematically formulated optimization problem has to be solved.
- Using the mathematical model, the results have to be transformed back into the dynamic model (for dynamic problems only).

14.2.3 General Aspects of the Optimization Process

If the aim of an optimization process is to optimize several properties that simultaneously affect the system (such as minimizing the size values while respecting the electrical properties), we obtain a multi-criteria objective function. The objective function then takes the form of a weighted sum of single-criterion

functions. Each of these functions generally assumes its local minima at different points of the optimization parameter spaces. This is the reason why a multi-criteria function can have a large number of shallow local minima or is insensitive to changes in the optimization parameters. Due to this fact, the selection of an optimization method is of great importance. The result is averaged in the sense that several criteria may participate simultaneously in a reduction of the multi-criteria function, while some other criteria may increase.

A more suitable method may be to select a single-criterion objective function, including all criteria in the constraints. Only the most significant criterion is chosen for the objective function to be specified in the subsequent process. All other criteria included in the constraints are kept within specified limits without being optimized. Thus, the results of an optimization process are dependent on the degree of reduction of the admissible set given by the inequality-type constraints.

Generally, we specify the constraints in a form similar to the objective function

$$q_i(p) = f_i(p) - f_i^h, \quad i = 1, 2, \ldots, m^* \tag{14.1}$$

Here f_i are suitable functions of a vector variable and f_i^h their maximum admissible values.

The selection of optimization variables is given by the sensitivity of the objective function to changes of relevant optimization variables. This sensitivity is described by the gradient vector of the objective function.

$$\text{grad } \psi(p) = \left[\frac{\delta\psi(p)}{\delta p_1}, \ldots, \frac{\delta\psi(p)}{\delta p_s}\right]^T \tag{14.2}$$

14.2.4 Types of Optimization Methods

14.2.4.1 Standard Optimization Methods

Most practical problems lead to nonlinear (transcendental) systems of equations. These may only be solved using numerical optimization methods. According to the order of the derivatives used in the application of a method, numerical methods of finding local minima of functions of several variables may be divided into:

1. Zero-order methods (comparative)
   - Methods of co-ordinate comparison
   - Simplex methods
   - Stochastic methods
2. First-order methods (gradient and quasi-gradient)
   - Methods of associated directions
   - Variable-metric methods
3. Second-order method (Newton method)

Stochastic Methods

These methods consist of calculating the values of the objective function at a large number of selected points. The points are selected by such criteria that each point in the space has an equal probability of being selected. The best points are then determined by comparing the function values. From the outlined strategy, it follows that these methods lead to computing the function values at a large number of points, which may protract the calculation. On the other hand, we can more easily reach the global optimum of the function to be optimized. These methods also comprise the evolution methods since the first solution population is generated completely by random. The difference only consists in the strategy of selecting better solutions.

14.2.4.2 Evolutional Optimization Methods

Since some problems are difficult to solve by standard numeric optimization methods, even if they converge to an acceptable optimum in a reasonable time, new approaches had to be developed. Therefore, a number of new methods have been designed based on the laws of natural genetics copying them in various degrees. These methods employ random generation of input parameters and so can be thought of as stochastic methods. In general, stochastic optimizing algorithms (including virtually all the evolutional algorithms) optimize using multi-parameter function with "wild" behavior, that is, with many minima or with an unknown gradient. Stochastic optimization methods are necessarily slower than heuristic approaches, which take advantage of the fact that they know the type and details of the function to be optimized. Unless the conditions for the global optimum are previously known, we can never be sure whether we have reached the global optimum to be able to terminate the optimization process. However, stochastic optimization methods also bring numerous benefits. They are generally very well specified and thus applicable virtually to any problem, and they can get out of the trap of a local minimum. The evolutional process of searching the space of potential solutions requires an equilibrium of two objectives:

- To find the nearest (mostly local) minimum as quickly as possible
- To search the space of all potential solutions in the optimum manner

The methods differ in their orientation towards these two objectives and they can be roughly ordered in a sequence starting with methods tending to local minima to methods searching a large number of potential solutions:

1. Stochastic "hill climbing" algorithms
2. Tabu search algorithms
3. Simulated annealing algorithms
4. Genetic algorithms

Hill Climbing Algorithm

This is the simplest optimization algorithm being a variant of the gradient method "without gradient" where the direction of the steepest climb is determined by searching the neighborhood. This algorithm also has all the drawbacks of gradient methods, in that it is very likely to end up in a local extreme without reaching the global minimum. Here the starting solution is generated at random. For the currently designed solution, a certain neighborhood is generated using a finite set of transformations and the best minimum is chosen from this neighborhood. The local solution obtained in this way is then used as the center of a new neighborhood in which the optimization is repeated. This process is iterated a specified number of times. In the course of this process the subsequent best solutions are recorded to be finally used as the resulting minimum. The basic drawback of this algorithm is that, after a number of iterations, it may revert to a local minimum that has already been passed in a previous step (the problem of looping). This problem can be avoided by running the algorithm several times with different randomly generated initial values to eventually choose the best result achieved.

Tabu Search Algorithm

At the end of the 1980s, Professor Fred Glover designed a new approach to solving the problem of finding the global minimum, which he called *tabu search*. At present, this method is among those used most frequently to solve combinatorial problems and problems of finding the global minimum. Based on the hill-climbing algorithm, it tries to eliminate the problem of looping. The hill-climbing algorithm is equipped with a so-called short-time memory, which, for a short previous interval of the algorithm history, remembers the inverse transformations to locally optimal solution transformations used to obtain the new centers in iterations. These inverse transformations are prohibited (tabu) when the new neighborhood is created for a given current solution. In this way, the looping caused by falling into the trap of a local minimum may substantially be reduced. A hill-climbing algorithm modified in this way systematically searches the entire area in which the global minimum of a function is to be found.

Simulated Annealing Algorithm

Apart from the stochastic methods and methods based on natural evolution, there is another possibility of simulating the evolution of systems based on the physical evolution of macroscopic systems. The annealing of a solid body in order to remove the internal stress is a simple example of this kind of evolution. For a physical interpretation of this process, consider a body that is heated until it reaches a high temperature. The temperature is then gradually lowered. The atoms of a body heated to a high temperature can easily overcome the local energetic barriers to reach equilibrium states. When the temperature is lowered, atoms are fixed in this state and the cooled off body is without internal stress.

This principle was used to design the method of simulated annealing. First, an initial temperature $T_{\max}$ is set, whose value is important for the method to be efficient. The simulated annealing algorithm then searches the space of all potential solutions in a strongly stochastic way, also accepting the states that correspond to solutions worse than the current one. This property of simulated annealing is a characteristic feature of this method and provides a way of escaping from a local minimum trap, thus allowing the search of another area of the entire solution space. However, as the annealing temperature T is lowered, the probability of accepting worse states as well is diminishing. For small temperature values then, only solutions better than the current one are considered.

Genetic Algorithm (GA)

Genetic algorithms (GAs) are most frequently used to optimize the parameters of an unknown system whose mathematical description is either too complicated or unknown [5]. When applying a GA, it is mostly sufficient to know a function assigning a price to each individual in the population. This may be the error of the solution for randomly selected parameters during GA. Since a GA is looking for a maximum, the error, which, on the contrary, is being minimized, must be transformed into looking for a maximum. This may be done in several different ways: by subtracting the error from the maximum error occurring, by calculating the inverted value of the error, or by using another transforming function that approaches zero as the error approaches one. Increased attention should be paid to setting up the program implementing the pricing function since it consumes the most computing time compared with the other GA components.

Apart from general optimization problems, GAs are mostly applied to neural networks. Here the tendency is to employ GAs at two different levels. First, for finding suitable weights for a neural network and second, when optimizing the structure of a neural network, that is, when selecting the algorithm, the number of input neurons in the hidden layers, the number of hidden layers, etc. Using a genetic algorithm to optimize the parameters of another genetic algorithm (the size of the population, the number of crossbreedings, the extent of mutations, the frequency of mutations) is a very revolutionary idea (optimization of the computation time where the computation time is a pricing function of the GA). As far as applications of GAs to problems encountered in research of electric machines are concerned, GAs have been used to identify the parameters of the substitution diagram of an induction motor.

By way of conclusion, it may be added that genetic algorithms perform surprisingly well when all other algorithms fail, such as for incomplete problems where the computation time is an exponential or factorial function of the number of variables. There is no point in using GAs to optimize relatively simple functions or functions for which special algorithms exist for their description. Considering the necessity to calculate the function values for tens or hundreds of genetic chains in a population and the necessity to evaluate hundreds or even thousands of populations during a single run of the program, GAs are rather time-consuming.

Despite the positive results achieved by using GAs, it is clear that nature must use even more intricate and, at the same time, not very sophisticated methods. The GAs described above only correspond to very primitive examples observed in nature, particularly those related to asexual reproduction with a single chromosome. Since nature has taken billions of years to test its algorithms, it is highly efficient to further learn from it. It is interesting that it needs no mathematics to solve complicated problems of optimization. Nevertheless there are other optimization methods suitable to solve the problems of the design [2–4].

14.2.5 Selection of a Suitable Optimization Method

The standard gradient method is still one of the most frequently used methods. Gradient methods or even the standard non-gradient methods (such as the simplex method) are not suitable if the finding of the global minimum is required of a function with many local minima. Mostly, these methods only reach an insignificant minimum close to the starting point (the initial solution) in which they are trapped. This deficiency is mostly removed by repeatedly selecting at random the initial solution of an optimization problem and taking the best result for the solution. The stochastic character of this process can only be seen in the random selection of the initial solution. The subsequent optimization algorithm then proceeds without any randomness. Then the evolutional optimization methods are thought of as stochastic ones despite their employing of a certain strategy when choosing the getter points. The following are the main differences between a genetic algorithm and the more frequently used gradient method:

- GA performs no gradient computation, which might be difficult and time consuming particularly for large systems.
- GA works with randomly generated solutions and may converge more quickly to the global minimum.

To optimize the draft design of an induction motor, an optimization method was employed using a genetic algorithm. This method is described in more detail in the following chapter.

14.3 Optimum Design of Induction Motor

14.3.1 IM Design Introduction

Actual design of an induction motor (IM) usually depends on the requirements of individual customers, who specifically define parameters which a designed machine should accomplish. In this way, with the same machine output, we can obtain different implementations that meet individual conditions more or less. It is possible to require a good quality of one parameter only with the deterioration of other parameters. We are going to deal with a design of motors of (0,6–200) kW outputs. Motors are designed for permanent load and with the project assignment the following input values are required:

Machine output P_n [kW], voltage U_{1n} [V], winding connection Y/D, number of poles 2**p** or rotation speed n [min^{-1}], grid frequency f [Hz], efficiency η[%], power factor cos φ, insulation class, IP implementation, and the shape of the machine.

We consider squirrel-cage motors in closed implementation with framework and cooling ribs. There is a cast aluminum rotor cage. For the design, data such as conductors and slots dimension or magnetic characteristics deducted from tables and graphs that are given by the standard or by the manufacturer's measurement, are needed. The actual design is a compromise between individual design parameters, so that a resulting machine would have the best possible operating characteristics with a perfect heat and material utilization. The actual motor design is described in the following section.

14.3.2 Classical IM Design Evaluation

An induction motor design, when carried out manually, represents hundreds of calculations, which can last tens of hours even with an experienced constructor. As computers made their way into practically all branches of design and analysis, a series of programs which co-operate with a designer in an interactive fashion and speed up a calculation were created.

In spite of indisputable advantages of this design process, we have to realize that there is a remarkable quantity of various design implementations of the given motor which, more or less, achieve the required motor operating characteristics. This approximates the global minimum of an objective function, which evaluates the design quality.

Thus, the idea to create a program for searching the whole state space of all possible solutions and selecting such a variant, which is the most appropriate to an evaluating objective function (the required

TABLE 14.1 Generated Parameters List and Setup of Their Limits

Parameter Name	Symbol	Dimension	High Limit	Low Limit
Stator outside diameter	D_e	mm	User optional	User optional
Stator inside diameter	D	mm	User optional	User optional
Ideal iron length	l_i	mm	User optional	User optional
Air gap induction	$B\delta$	T	0.5	1.0
Stator slot filling	k_{dr1}	—	0.6	0.75 (0.8)
Air gap size	δ	mm	0.2	0.4
Stator current density	σ_1	A mm^{-2}	3.0	15.0
Rotor rod current density	σ_t	A mm^{-2}	2.0	6.0
Rotor ring current density	σ_k	A mm^{-2}	2.0	4.0
Teeth magnetic induction	B_z	T	1.6	2.0
Slot number per pole and stator phase	q_1	—	2.0	5.0

motor characteristics) using some of the optimization methods, was developed. The stochastic evolutionary method genetic algorithm was selected, because it searches the whole state space of all possible solution in a best way.

14.3.3 Description of a Solved Problem

14.3.3.1 Generated Parameters

The values given in Table 14.1 are recommended only, and, for the motors of outputs below 200 kW, they are mostly limiting values. Varying the parameter values or substituting one parameter for another is possible only by intervention in the program source text. Diameter limits D_e and D from the input file are considered only in the case that a motor design without regard to standardized axis height is required. In the case that standardized axis height is entered, these limits are calculated. Limits of an ideal rotor length are appropriate to enter as narrow as possible for faster convergence to a limit. But this is not a required condition. Generally, the lower the range of individual parameters, the faster the convergence to a global minimum, and the number of local minimums is lower.

14.3.3.2 Objective (Criterion) Function

It is not just the form of the objective function, but also the selection of optimized parameters that is important for good optimization results. Selected parameters must sufficiently describe a quality solution to the given problem. In the case of an induction motor design optimization task, the following parameters were selected:

Motor volume	V [dm^3]
Motor temperature rise	ϑ_n [K]
Motor nominal power factor	$\cos\varphi_n$ [–]
Motor nominal efficiency	η_n [–]
Torque overload capacity	m_{pn} [–]

These parameters are most important for the quality of the design and describe the design sufficiently.

The total error is based on the relation given in Equation 14.1, a sum of individual partial errors of each controlled parameter. If we put more emphasis on some parameter, we increase a corresponding weight coefficient, thus achieving its improvement in the final design. At the same time, values of other parameters will decrease. Finding an optimal setup of gain coefficients is one of the most important and difficult problems. The term "optimal setup" means that the designed motor has the highest power factor, efficiency, and torque overload capacity values at the minimal volume and simultaneously does not exceed a permitted temperature rise for a selected insulation class. We have the relationship

$$\begin{aligned}\varepsilon(\mathrm{GR}_i) = {} & \mathrm{abs}(kV \cdot V) + \mathrm{abs}(k\vartheta(0.89\,\vartheta_d - \vartheta_n)) + \mathrm{abs}(k_{\cos\varphi}(1 - \cos\varphi_n)) \\ & + \mathrm{abs}(k\eta(1 - \eta_n)) + \mathrm{abs}(k_{mp}(m_p + 1 - m_{pn}))\end{aligned} \tag{14.3}$$

TABLE 14.2 Input Values of 5.5 kW, 380 V Motor

Quantity Name	Symbol	Dimension	Value
Nominal power output	P_n	W	5500
Nominal voltage	U_{1n}	V	380
Required power factor	$\cos\varphi$	—	0.81
Required efficiency	η	—	0.86
Grid frequency	f	Hz	50
Motor axis height	H	mm	132
Number of pole pairs	p	—	3
Temperature class of insulation	TT	—	F
Torque overload capacity	m_p	—	2

where kV is the volume weight coefficient, $k\vartheta$ is the temperature factor weight coefficient, $k_{\cos\varphi}$ is the power factor weight coefficient, $k\eta$ is the efficiency weight coefficient, k_{mp} is the torque overload capacity weight coefficient.

14.3.4 Achieved Results

14.3.4.1 5.5 kW, 380 V Motor Design Description

During program development and tuning, an optimization was performed on the motor described in Table 14.2. In this section we describe the results and problems encountered in the optimization process. The symbols and quantities, which are not explained in detail, were either used in the previous text or are listed at the list of used quantities at the beginning of the document. The motor input parameters are given in Table 14.2.

14.3.4.2 Other Results

It follows from the physical principle that optimized quantities are closely related. Increase of a gain of one quantity results in a disadvantage of the quantities. Based on the performed optimizations, it can be concluded that two kinds of motors exist, depending on the content of iron and copper:

1. Motor with prevailing iron content, high stator current density, good power factor, for a price of worse efficiency of the motor and with slightly worse torque overload capacity than the second motor.
2. Motor with high copper content and conversely low stator current density, good efficiency, and worse power factor value. Torque overload capacity is good.

The type of motor is determined based on the setting of gain coefficients. A sum of temperature and power factor errors on one side impacts the sum of volume and efficiency errors on the other side. The torque overload capacity can be good for both kinds of motors.

Results of individual optimizations are listed in Table 14.3, ordered by volume value from smallest to largest. Different varieties of motors were obtained, depending on values of gain coefficients. It is difficult to determine which solutions are good or bad, because the selections depend on actual customers' requirements. The solution that gives the best value of optimized quantity is marked in bold. Solution numbers 1, 5, 8, 23, and 25 can be considered successful from this perspective. The motor described above (solution no. 2) serves for depicting of the task. Previously-mentioned relations between individual quantities can be observed in Table 14.3, which lists the optimization results without limiting the generated parameter, thus using the requirements in Table 14.1.

Next, a motor optimization was performed with just one optimized parameter, when the gain of the other parameters was set equal to zero.

1. *Volume optimization.* In this case, the algorithm selected as the best solution motors with minimal dimensions, when parameters D_e, D, and l_i were converging to minimum preset limits.
2. *Temperature optimization.* The algorithm reached first reached a local minimum with temperature at the maximum based on the required insulation class, and mostly stayed on this value.

TABLE 14.3 Motor $P = 5.5$ kW, $U = 380$ V Solutions List, without Generated Parameters Limited

Number	V [dm^3]	ϑ [K]	$\cos\varphi$ [–]	η [–]	m_p [–]	Directory
1	**3.96**	88.1	0.798	0.834	1.72	Motor1
2	4.20	86.9	0.818	0.843	1.90	Motor2
3	4.31	74.9	0.787	0.865	1.77	Motor3
4	4.32	88.8	0.836	0.817	1.78	Motor4
5	4.33	75.1	0.690	**0.973**	1.07	Motor5
6	4.50	89.0	0.836	0.834	1.79	Motor6
7	4.51	86.8	0.818	0.818	1.93	Motor7
8	4.54	**90.0**	0.884	0.812	1.98	Motor8
9	4.56	84.6	0.857	0.816	1.74	Motor9
10	4.58	86.5	0.836	0.817	1.77	Motor10
11	4.63	68.2	0.792	0.858	2.10	Motor11
12	4.69	88.4	0.862	0.808	1.80	Motor12
13	4.70	73.4	0.845	0.830	2.25	Motor13
14	4.73	61.0	0.799	0.871	1.90	Motor14
15	4.78	78.1	0.853	0.858	1.67	Motor15
16	4.78	71.0	0.767	0.870	1.80	Motor16
17	4.81	70.6	0.703	0.934	1.28	Motor17
18	4.97	54.5	0.804	0.883	1.90	Motor18
19	5.08	55.5	0.762	0.877	2.20	Motor19
20	5.12	88.2	0.879	0.806	2.05	Motor20
21	5.96	44.0	0.784	0.870	2.55	Motor21
22	6.35	42.4	0.803	0.882	2.69	Motor22
23	6.40	87.5	**0.887**	0.853	2.27	Motor23
24	6.57	59.0	0.747	0.956	1.16	Motor24
25	7.05	42.3	0.793	0.865	**3.00**	Motor25
26	7.39	52.9	0.714	0.986	1.03	Motor26

3. *Power factor optimization*. An effort to achieve the first type of motor (see above discussion) with low copper content, high current density σ_1, worse value of efficiency. Torque overload capacity was good.
4. *Efficiency optimization*. The designed motor corresponded to the second type of motor (see above discussion) with prevailing copper content, low current density σ_1, and good efficiency, however, with worse power factor values. Torque overload capacity was good.
5. *Torque overload capacity optimization*. The motor is designed with high number of slots for pole and phase, resulting in gradual spread of conductors on the perimeter. The motor can have prevailing iron or copper content depending on a local solution, to which it converged. It can have good values of power factor and efficiency for a price of machine volume increase.

14.4 The Use of a Neuron Network for the Identification of the Parameters of a Mechanical Dynamic System

The basic step used to solve the dynamic tasks by means of any type of modeling is to create a set of important quantities that include both the quantities describing structure, conditions, and the interactions of technical objects and the quantities that characterize the consequences (i.e., their demonstration and behavior).

The methods of creating the mathematical models in drive systems, in general an interactive process, utilize

- The applications of well-known physical principles that describe the phenomena in drive systems (e.g., the second Newton's principle, Kirchhoff's laws, etc.)
- The applications of the methods based on artificial intelligence algorithms (e.g., genetic algorithms [1] and artificial neuron networks [6,7])

The theories on which the methods of artificial intelligence are based replace the "standard" analytical and numerical methods when

- These are the only theories which can solve the problem.
- They exhibit better properties from the point of view of the problem solution (e.g., a better conditionality considering the changes of input values).
- They allow the problems to be solved more effectively.

The last case is typical when we want to approach the real operational conditions as close as possible. From various methods of artificial intelligence, the stochastic evolution algorithms and the artificial neuron networks are being increasingly utilized in the field of the modelling of drive interactive systems. In the following section, two methods are shown that are applied to the problems of the analysis of dynamic properties in drive systems.

The solution of dynamics by means of the algorithms of artificial intelligence represents a solution of the following partial problems:

- Specifying the set of important (relevant) quantities
- Selecting the theory which is suitable to solve the problems
- Arranging the relations among relevant quantities so that they allow the selected algorithm of artificial intelligence to be used
- Generating the training data, and the selection of the method of teaching, for example, in neuron networks
- Testing the quality of the results reached and their evaluation

14.4.1 Practical Application

Many identification methods are known and very often verified quite well in practical terms. The limit factors that make these procedures more difficult (e.g., the assumptions about system linearity, stationarity, and normality of the phenomena which occur in the systems, etc.) are also known. Hence, we have used an untraditional approach to the problem *f* identifying the dynamic properties in mechanical systems for which the use of neuron systems seems to be promising, and at the same time available for engineers' thinking.

14.4.1.1 A Practical Application—Gearbox

The first case that was analyzed is a vehicle gearbox. Inputs (engine load), outputs (frequency-amplitude spectra of torsional oscillations), and the gearbox structure were known. The ilete information was related to the selected parameters of stiffness and damping in the drive. Due to variable operational conditions, the magnitude of damping may vary enough to significantly affect output characteristics. If many experimental results are evaluated, some typical failures can be identified and their possible occurrence can be anticipated from output files. We have used the data that were measured in the real system. The frequency-amplitude spectrum of torsional oscillations was measured at the gearbox shaft, which is seated on four bearings with five speed gears (Figure 14.1). Originally, the measurement was carried out to determine the resonance of frequency systems with the goal to reduce noise. The following parameters were set in the system:

Stiffness $K \in \{0.3, 12.0, \infty\}$
Damping $B \in \{-, 0, 0.3, 12.0\}$

The measurements were done with testing frequencies $f = 512$ Hz and $f = 1024$ Hz. To record significant oscillation harmonics, the excitation frequency was flexible in both cases:

1. From 2.5 to 14.0 Hz in steps of 0.5 Hz
2. From 14.0 to 40.0 Hz in steps of 1.0 Hz

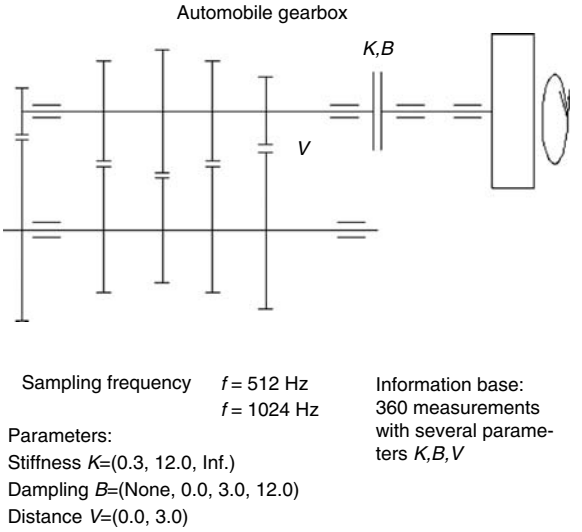


FIGURE 14.1 Five speed gear.

TABLE 14.4 Expected Natural and Excitation Frequencies for the Gearbox

Table of Expected Frequencies		[Hz]
Low frequencies		up to 5.0
Operational frequency (OF) (speed)	OF	14.16
	2 × OF	28.32
	3 × OF	42.48
Interharmonic frequency (IHF)	0.5 × OF	7.08
	1.5 × OF	21.25
	2.5 × OF	35.42
Natural frequency (NF)	I.NF	43.91
	0.5 × I.NF	21.96
	2 × I.NF	87.82
	II.NF	322.1
	0.5 × II.NF	161.1
	2.0 × II.NF	644.2
Combination frequency (CF)	2 × OF + 0.5 × I.NF	50.28
	OF + I.NF	58.07
	2 × OF + I.NF	72.23
	2 × OF + 2 × I.NF	116.1
Tooth frequency (TF) {TF = z.OF}	1st speed gear	184.1
	2nd speed gear	325.7
	3rd speed gear	424.9
	4th speed gear	580.6

The spectrum always included 512 spectral lines. The measurement was repeated 360 times for different variations of the parameters K and B (the system adjustment). The values of natural and excitation frequencies, which are expected for the gearbox to be analyzed, are shown in Table 14.4.

14.4.1.2 Task Definition

The task was originally defined in the following way: to estimate the corresponding magnitudes of parameters K and B (this means, to recognize the adjustment of parameters used in the mechanical dynamic system) by means of the artificial neuron network on the basis of the frequency-amplitude

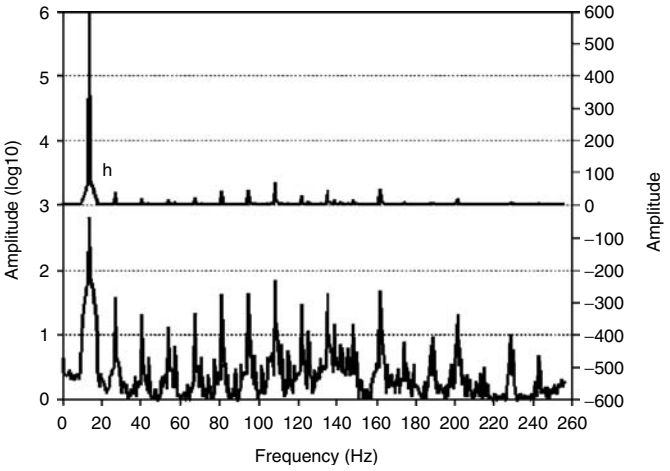


FIGURE 14.2 Stimulus vectors normalization (top, before normalization; bottom, after normalization).

spectrum of torsional shaft oscillations in the given system. A multilayer perceptron with three layers (i.e., input, hidden, and output) is used as the configuration of the neuron network. Select the signal neuron functions as the linear, hidden, and output layers of the logistic function, $f(x) = 1/(1 + e^{-x})$.

According to this definition, this is to identify the system parameters on the basis of the measured frequency-amplitude spectra. However, the parameters are taken from discrete sets (and very low), and the task could be redefined as the "standard" task of the spectrum classification according to seven attributes (each attribute corresponded to one of the possible values of the parameters K and B). The application of neuron networks to solve such a problem is more successful when compared to the solution of the original task.

The amplitudes of spectral lines were expressed in logarithm scale, and a reduction of spectral dynamics with an increase of their informative quality has been achieved. Considering the nonlinear nature of the activation neuron functions used, which extends beyond the saturation range for the input interval $\langle 0.5, 0.95 \rangle$, the network cannot respond well to stimulus vectors with a high range of the values in the individual components. This is illustrated in Figure 14.2. The input network layer was configured to 512 input neurons. The amplitude logarithmic value of one spectral line was entered into each input. The individual neurons in the output layer correspond to the classification attributes. Because there are seven attributes, seven neurons were configured in the output layer.

The only-hidden layer was set as the arithmetic mean of the number of input and output neurons. Two hundred sixty neurons were configured to the only-hidden layer, as illustrated in Figure 14.3. Each item corresponded to one measurement of the frequency spectrum (a stimulus vector) with a corresponding attribute vector (a vector of the required responses). The specific variation of the parameters was expressed by the required network response to two corresponding output neurons equal to 1, and the remaining output neurons equal to 0.

From the original 360 items, 36 items were randomly separated (10% of total) for the future tests. We ensured that the network tests would be carried out with the items that have not passed the training network process (the network was not trained to these situations). This is necessary to verify the generalization model properties. The training set was formed by the remaining 324 items. The sequential strategy of teaching was used, i.e., the items from the training set were used in the teaching process with the fixed sequence (cyclic passages through the training set). Taking into consideration the size of the neuron network to be configured, the method of feedback moment propagation has been selected as the teaching method that exploits only the information up to the first-order inclusively (the values of a special function—a teacher and his gradient), and it has not used Hessian or its estimate, which, in this case, would be very demanding.

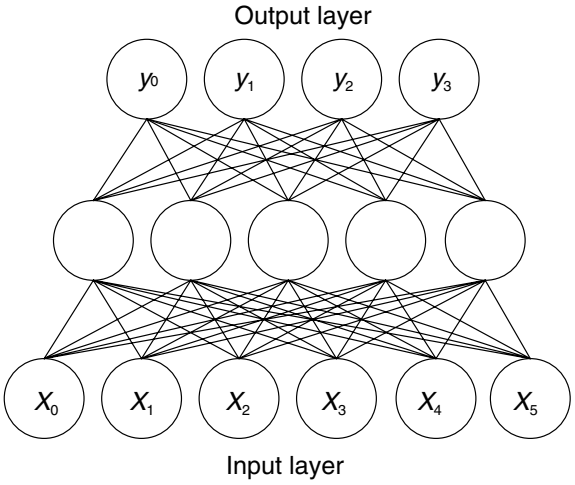


FIGURE 14.3 Network configuration (hidden layer contains neurons without labels).

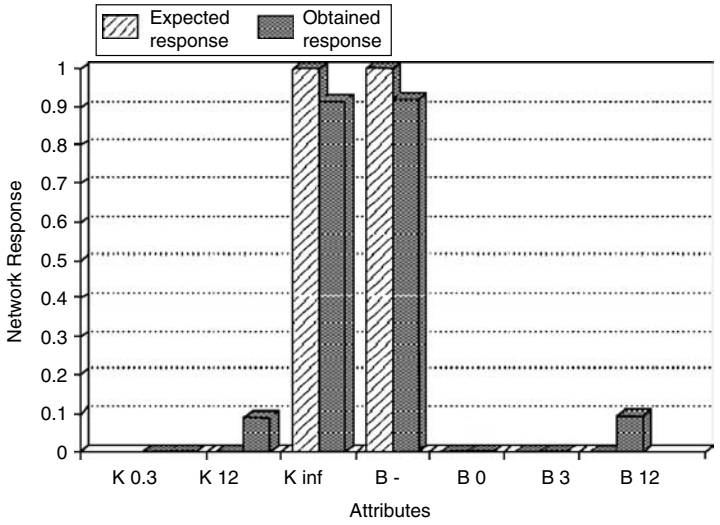


FIGURE 14.4 Successful response of neural net.

14.4.1.3 Results

The neuron model of the mechanical system has manifested a high rate of success during the verification by test sets. The network was taught with random selections of the test items. During testing of the individual models, the responses of the network were successful in 85–95% of all cases (see Figures 14.4 and 14.5). Moreover, the estimate of the values K and B that correspond to the parameters is available within a couple of seconds for the frequency spectrum in the active mode. However, it is possible that a model with higher quality will be achieved if special optimizing techniques are used in the future.

In summary, the neuron model of the mechanical system described above can be assessed as usable in practical terms.

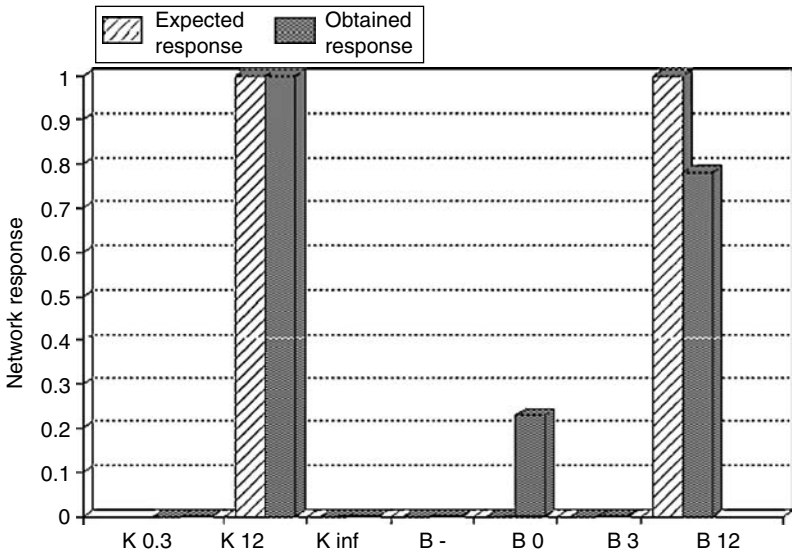


FIGURE 14.5 Failure response of neural net.

References

1. Goldberg, D., *Genetic Algorithms in Searching, Optimisation and Machine Learning,* Reading, MA: Addison-Wesley, 1989.
2. Glover, F., Lagunai, M., Marti, R., Fundamentals of scatter search and path relinking, *Control and Cybernetics,* pp. 653–684, 2000.
3. Glover, F., *Scatter Search and Star-Paths—Beyond the Genetic Metaphor,* New York: Springer-Verlag, pp. 125–137, September 1995.
4. Glover, F., Kelly, J.P., Langunai, M., Genetic algorithm and tabu search—hybrids for optimization, *Computers and Operations Research,* pp. 111–134, January 1995.
5. Lee, J., Hajela, P., Parallel genetic algorithm implementation in multidisciplinary rotor blade design, *Journal of Aircraft,* Vol. 33, No.5, pp. 962–969, September–October 1996.
6. Hagan, M.T., Demuth, H., Beale, M., *Neural Network Design,* Boston: PWS Publishing, 1996.
7. Kosko, B., *Neural Networks and Fuzzy Systems,* Englewood Cliffs, NJ: Prentice-Hall, 1992.
8. Ye, X., Loh, N., Dynamic system identification using recurrent radial basis function network, *Neural Networks Theory, Technology, and Applications,* New York: IEEE Technology Update Series, 1996.

15

Motion Control

Rahul Kulkarni
National Instruments, Inc

15.1 Introduction to Motion Control

15.1.1 Definition

Motion control involves precisely controlling the position, velocity, and torque of a rotational or linear electromechanical device. Examples of such devices include hydraulic pumps, linear actuators, and electric motors.

15.1.2 History

In 1831, Michael Faraday discovered electromagnetic induction—electricity being generated in a wire by means of the electromagnetic effect of a current in another wire, and magneto-electric induction—electricity being generated in a wire by means of the electromagnetic effect of a magnet. These results led to the creation of the modern electric motor.

The industrial revolution put the motor to use in a variety of applications. A single motor was typically used by multiple machines connected to a drive train using belts for transmission and gears for speed reduction. A clutch was used to engage and disengage different machines on the drive train.

Complex machines were designed for mass production using a motor connected to amechanical system based on gears and cams. Hundreds of parts were produced on the same line as long as no parameter in the design changed.

As world moves from mass production to mass customization, machine designers want the same machine to handle a variety of different products at different times in the day, with minimal

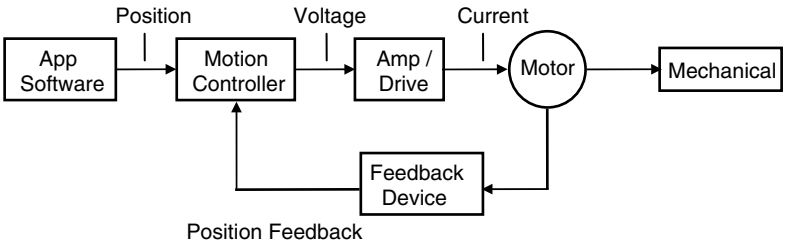


FIGURE 15.1 Components of a motion control system.

changeover time. Servo-based motion control systems enabled tight control over position and move profiles. Thus most machines today are complex electromechanical systems involving servo-based control.

15.2 Components of a Typical Motion Control System

A typical motion control system consists of a variety of components including user interface, motion controller, drive, motor, feedback device, and motion I/O. These components work together seamlessly to control the position, velocity, or torque of an electromechanical device. Figure 15.1 shows the different components of a motion control system.

15.2.1 User Interface

The user interface component accepts commands from the engineer or operator and sends them to the motion controller for execution. These commands include parameters such as target position, and motion control profiles for smooth movement. The user interface typically includes a graphical interface with switches and knobs to resemble the control panel of a machine in operation, and elements such as graphs or XY plots to represent the current position of the object in motion.

15.2.2 Motion Controller

A motion controller is at the center of a typical motion system, which consists of supervisory control, trajectory generation, and a control loop. The controller converts high-level user commands into command signals that drives use to move actuators. The motion controller also monitors the system for error conditions, faults, and asynchronous events that can cause the system to change speed, direction, or start/stop the actuators.

15.2.3 Drive

In a motion control system, the drive is the component that translates the command signals from the controller to current that produces torque or rotation in the motor. Drives can be classified on the basis of the type of motor they actuate—brushed servo, brushless servo, or stepper, or on the basis of the level of computation carried out in the drive—analog or digital. We will learn more about different types of drives in Sections 15.4 and 15.5.

15.2.4 Motor

Motors turn electrical energy into mechanical energy and produce the torque required to move to the desired target position. Motors are designed to provide torque to some mechanical elements including linear slides, robotic arms, and special actuators. There are three main types of motors—brushed DC

TABLE 15.1 Types of Motors

	Pros	Cons	Applications
Stepper Motors	Inexpensive, generally run open loop, good low-end torque, clean rooms	Noisy and resonant, poor high-speed torque, not for hot environments, not for variable loads	Positioning, micro movement
Brushed DC Servo Motors	Inexpensive, moderate speed, good high-end torque, simple drives	Maintenance required, no clean rooms, brush sparking causes EMI and danger in explosive environments	Velocity control, high-speed position control
Brushless Servo Motors	Maintenance-free, long lifetime, no sparking, high speeds, clean rooms, quiet, run cool	Expensive and complicated drives	Robotics, pick-and-place, high-torque applications

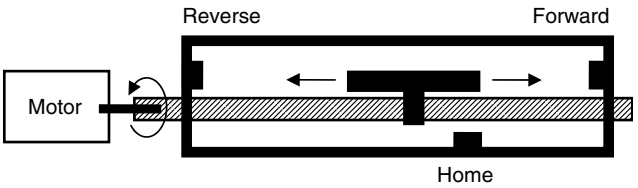


FIGURE 15.2 Limit and home switches in a motion control system.

servo motors, brushless servo motors, and stepper motors. Table 15.1 summarizes the pros, cons, and typical application areas for each type of motor.

15.2.5 Feedback Device

In most motion control systems, amethod for measuring the characteristic being controlled (i.e., position, velocity, etc.) needs to be implemented. This is known as feedback. Some of the different kinds of feedback used in motion control are relative position, absolute position, velocity, and analog voltage feedback. For relative position feedback, the most common device used is the incremental encoder. This device increments as it turns and can be used to keep track of the total distance traveled from some point of reference. The most common absolute position feedback devices are absolute encoders and resolvers. An absolute encoder is similar to an incremental encoder except that it does not need to have a reference position and can know what position the shaft is in on start-up. A resolver is also another absolute position device that is commonly used for both position and velocity feedback. For velocity feedback, a tachometer is commonly used. Some applications require feedback other than the type that has been mentioned above. In these cases, analog feedback describing some characteristic of the system such as temperature or pressure can be useful in controlling motors. A position feedback device is not required for some motion control applications, such as controlling stepper motors, but is vital for servo motors.

15.2.6 Motion I/O

A motion control system also includes digital I/O such as limit switches, home switches, position compare outputs, and position capture inputs.

Limit switches provide information about the end of travel to help you avoid damaging your system. When a motion system hits a limit switch, it typically stops moving. Home switches, on the other hand, indicate the system home position to help you define a reference point. This is important for applications such as pick-and-place (Figure 15.2).

Triggers such as position compare outputs or position capture inputs help when synchronizing moves with other external devices and sensors such as data acquisition and vision.With position compare outputs, you can set up a trigger that fires at a defined position. This type of action is very useful in operations such as scanning, where you might want to trigger a system to take measurements at a series of defined positions. Position capture inputs, on the other hand, cause the motion controller to immediately capture a position on the occurrence of an external event such as a sensor being encountered while moving. This is useful for resynchronizing the position of an axis (registration moves for "cut on the fly" applications) or recording positions for analysis (scanning defects on a printed circuit board (PCB)).

15.3 Functions of a Motion Controller

If we look at the functions of a motion controller, they can be broadly split up into three components—trajectory generation that involves path planning on the basis of the profile desired by the user, supervisory control that includes system initialization, event handling, fault detection, and command response, and the control loop that includes closing the position and velocity loops on the basis of feedback. Figure 15.3 shows the parts and processes of a typical motion controller.

These three components of a motion controller work in synchronization with each other. The supervisory control is the interface that handles user-defined requests and commands the trajectory generator to start calculating set points. The trajectory generator calculates a set point per iteration and writes it to the control loop. The control loop takes this set point and using a control algorithm ensures that the actuator follows this set point accurately. In the case, the control loop can run faster than the trajectory generator; the control loop uses an interpolation algorithm that creates intermediate set points per iteration of the control loop.

15.3.1 Supervisory Control

The supervisory control is the main loop of the motion control system. This loop intercepts commands from the user and signals the trajectory generator to start/stop moves. The supervisory control loop also monitors all I/O needed to perform initialization tasks, such as finding the reference or origin. Supervisory controlmonitors I/O for fault conditions and user-defined events and accordingly commands the trajectory generator to take action.

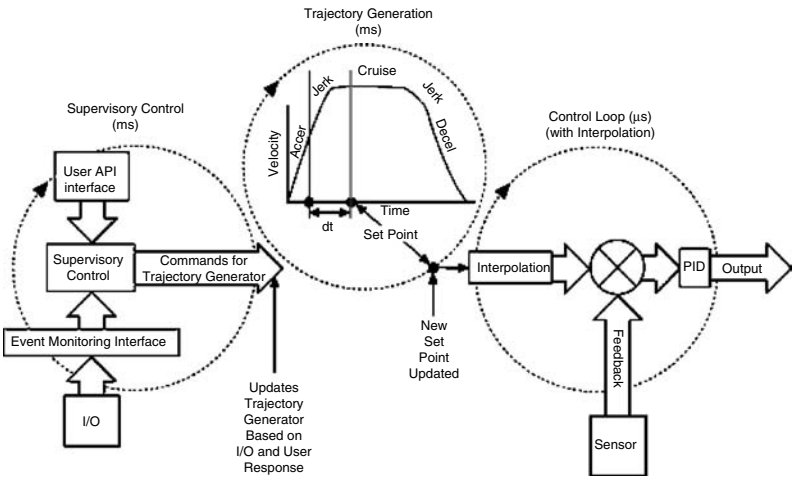


FIGURE 15.3 Motion controller architecture.

15.3.1.1 Axis Initialization (Reference Moves)

A reference move resets the axes of a motion script to a known location or state. Reference moves are generally used at the beginning of a motion script to prepare the motion system for subsequent motion. The following are valid types of reference moves—finding the home switch, finding the encoder index, finding the forward limit switch, finding the reverse limit switch, and finding the center for a system.

Finding the Home Switch

Home reference moves perform a course reference move by traversing the entirety of the motion path in search of a physical home switch. A home switch consists of a logical ON state bordered by a front edge and a back edge. Home reference moves can operate when there is a physical home switch present or if there is a physical end-of-travel present, such as on a stage.

Finding the Encoder Index

Index reference moves perform a find reference move by moving the axis until the index state on the encoder is reached. The index state is a specific position on the 360° encoder that is passed during each revolution of the motor. An index reference move is usually performed after a home reference move for fine reference adjustment. The encoder index signal is accurate to one encoder count and provides amuch more repeatable reference than using just a home switch edge.

Finding the Forward Limit and Reverse Limit Switches

A forward limit reference move initiates a search sequence to find the forward limit. A reverse limit reference move initiates a search sequence to find the reverse limit.When the search operation is initiated, the axis starts moving in the direction of the limit. When the limit in that direction is encountered, the search is completed. A fine-tuning operation is performed when the limit is reached to minimize the effects of motion system windup, backlash, and/or home sensor hysteresis. This type of reference move is used when a home switch is not available on a system.

Finding the Center

Center reference moves initiate a search sequence positioning the axis in the middle of the forward and reverse limits.When the search direction is forward, the axis starts moving in the forward direction.When the limit in that direction is encountered, the direction is reversed to find the reverse limit. After both limits are reached, a center position can be calculated from the recorded positions. The axis then proceeds to the calculated center.

After finding the reference position for an axis the internal position register for that axis is typically updated by:

Resetting the Position Counter Resetting the position counter for an axis causes the axis to start counting its position at a specified starting number. This starting number is typically zero, but resetting to a nonzero number can be useful when you want to begin the moves relative to an absolute zero, rather than the current position. For example, a steering system for a vehicle might come online with the wheels turned 7000 counts to the right of absolute zero. In this example, absolute zero points the wheels straight ahead. To ensure the motion system recognizes that the wheels are not starting off pointed straight ahead, reset the position to the absolute count of 7000.

Offsetting the Axis Position This is achieved by moving the axis by an amount equal to a predetermined offset. Offsetting the position preserves the value of performing the reference moves, but adds another move to create distance from the home switch or index. This is usually done to prevent the axis from sitting on a switch or encoder index.

15.3.1.2 Electronic Gearing

Electronic gearing allows one slave motor to be driven in proportion to a master motor or feedback sensor, such as an encoder or torque (analog) sensor. As the slave follows the master position at a constant ratio, the effect is similar to that of two axes mechanically geared. For example, every turn of the master axis may cause a slave axis to turn twice.

Electronic gearing has several advantages over mechanical gears. The most notable is flexibility because you can change gear ratios on-the-fly. The other major advantage to electronic gearing is that you can superimpose amove over a geared axis. The superimposedmove is added to the geared profile of the slave axis, which allows the slave axis to be synchronized on-the-fly (Figure 15.4).

The gear ratio is used to determine how far the slave axis must move in proportion to the master when gearing is enabled. The gear ratio can be absolute or relative.

$$\text{Slave axis move} = \text{Master axis position} \times \text{Gear ratio}$$

For example, if you have a gearing ratio of 2:1 (slave:master), the slave moves 20 counts when the master device moves 10 counts.

Absolute gearing behaves similarly to relative gearing in that when gearing is enabled, the slave axis follows the master axis movement as it is defined by the gear ratio. The difference between relative and absolute gearing is that the reference position calculated for the master axis is updated only when gearing is enabled. This difference is apparent when the gear ratio is updated on-the-fly (Figure 15.5).

For example, if the gear ratio is 2:1, the current master position is 1010, the current slave position is 3020, and the gear ratio is changed to 3:1; the slave axis jumps from 3020 to 3030 but the master position remains the same (Figure 15.6).

Changing a gear ratio on-the-fly during absolute gearing allows you to quickly synchronize the slave axis with the master axis.

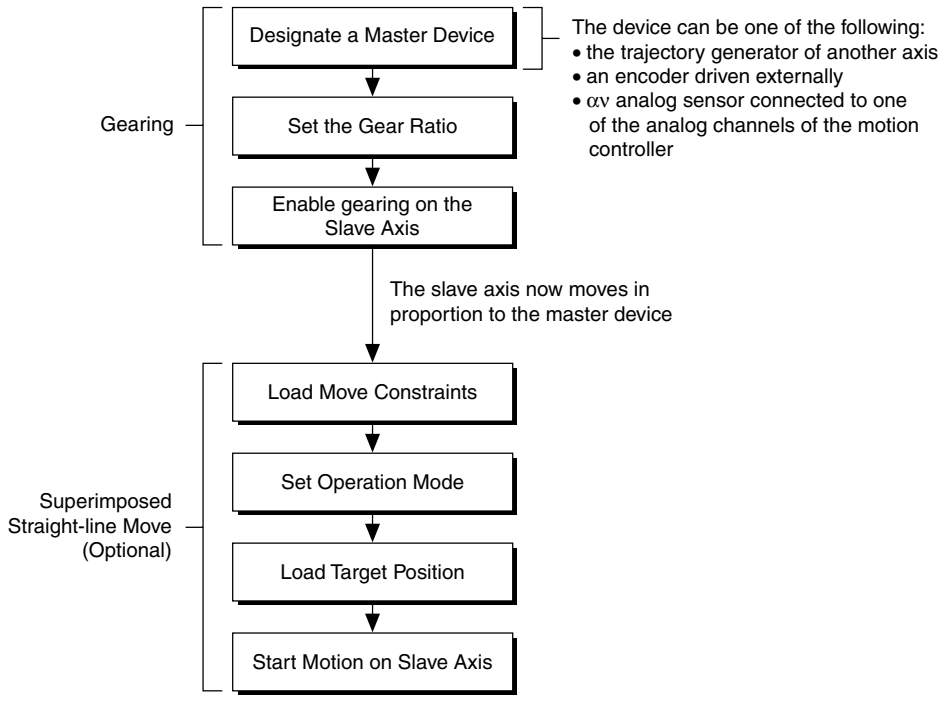


FIGURE 15.4 Electronic gearing algorithm.

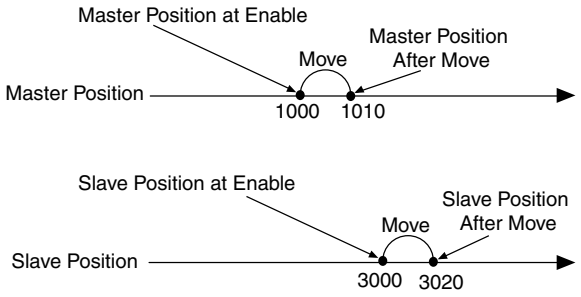


FIGURE 15.5 Relative gearing at enable.

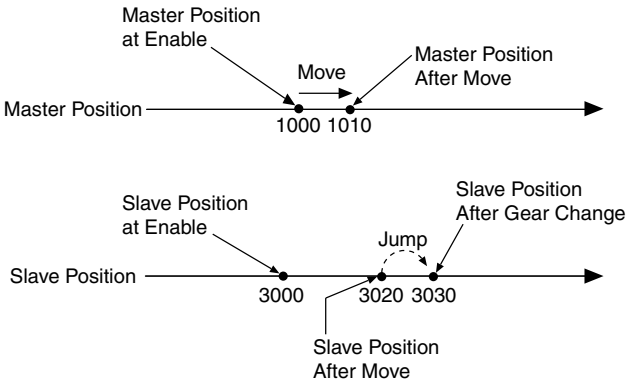


FIGURE 15.6 Absolute gearing at gear ratio change.

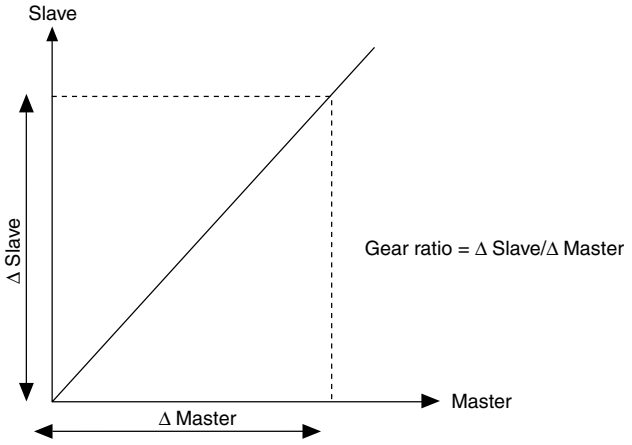


FIGURE 15.7 Master/slave ratio in gearing.

15.3.1.3 Electronic Camming (Includes Registration Moves)

Electronic camming operates similarly to electronic gearing in that the move distance of an axis is proportional to the move distance of its master device. Camming differs from gearing in how the master/slave ratio is handled by the motion controller. Gearing is used in applications where a constant gear value creates a linear slave position profile, as shown in Figure 15.7.

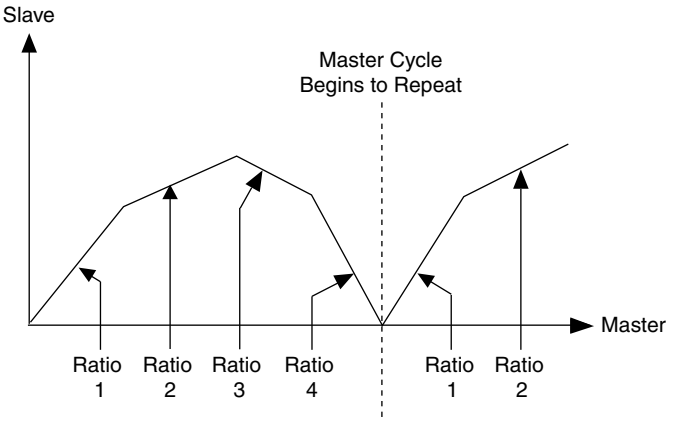


FIGURE 15.8 Multiple camming gear ratios.

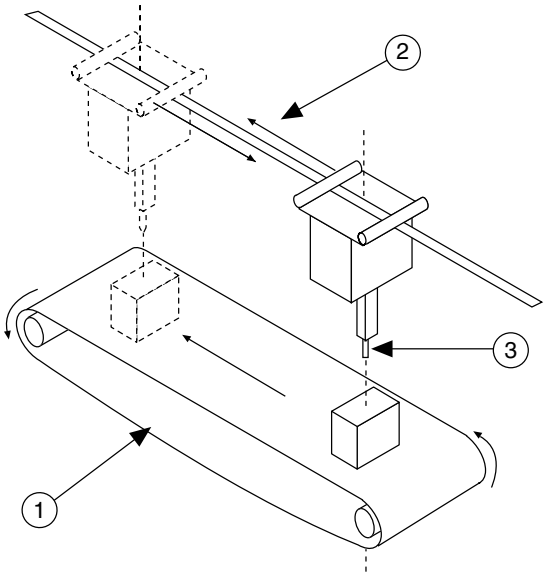


FIGURE 15.9 Welding application showing (1) conveyor belt, (2) movement of the welder as it follows the object and then returns to the initial position, and (3) welding point.

Camming creates a more flexible profile by using more master/slave ratios. These ratios are handled automatically by the motion controller, allowing precise switching of the gear ratios, as shown in Figure 15.8. Camming is used in applications where the slave axis follows a nonlinear profile from a master device.

Figure 15.9 shows that a welding point moves to the first position, and then welds the material for a couple of seconds. Because the conveyor belt keeps moving at a constant rate, the welding point must follow the material at the same speed as the conveyor belt during the weld process. When the welding process is finished for one item, the welding point must quickly return to its initial position, and the process repeats.

In this application, the master device is the position encoder attached to the conveyor belt, and the slave axis is the actuator that moves the welding point. The slave axis repeatedly performs a two-segment movement—first, it follows the material with the same speed as the conveyor belt, and next, it returns to

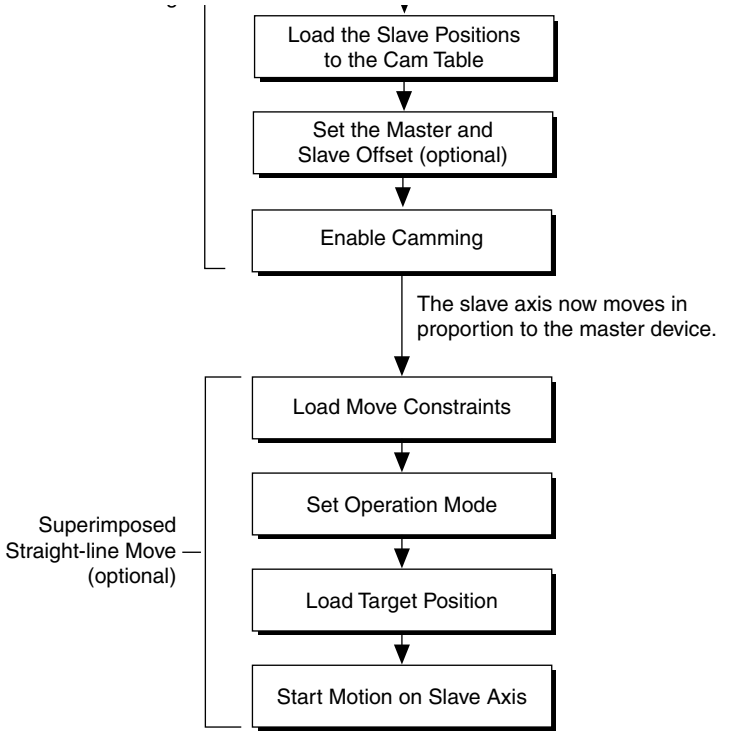


FIGURE 15.10 Camming algorithm.

the initial position as the next material approaches. Each segment of the move is represented with a gear ratio that dictates how fast and which direction the welding point is moving relative to the conveyor belt. This application requires the slave axis to switch from one ratio to the other at the correct master position, otherwise the welding process is not repeatable. If this application used gearing instead of camming, the latency, or delay, of loading a new gear ratio might cause an accumulation of position errors.

Similar to gearing, in a camming application, a slave axis can perform any move when camming is enabled. The move profile is superimposed over the camming profile (Figure 15.10).

15.3.1.4 Exception Handling (E-Stop and Limit Switches)

A motion controller needs to be able to detect fault conditions and handle these exceptions in order of severity. Examples of exceptions include scenarios where the actuator hit a forward or reverse limit switch or if the E-stop switch was pressed.

Limit switches can be either hardware or software limits. The hardware limit inputs are typically connected to end-of-travel limit switches or sensors. An enabled limit input causes a full torque halt stop on the axis when the input becomes active. Active limit inputs also prohibit attempts to start motion that would cause additional travel in the direction of the limit. Similarly, software limits are often used to restrict the range of travel further and avoid ever hitting the hardware limit switches. An enabled software limit causes the axis to smoothly decelerate to a stop when the limit position is reached or exceeded.

15.3.1.5 I/O Synchronization (Captures, Compares, Time-Sampled Data Logging)

You may need to synchronize motion control with data acquisition or image acquisition devices within the same systemusing position compares, position captures and time-sampled data logging. Let us understand first what each of these terms mean.

In scanning applications, you typically inspect an object such as a wafer or an LCD display for defects using a camera. The wafer or LCDdisplay rests on an XY stage that moves in two dimensions. The objective

of the scan is to cover as much space on the wafer as possible in the shortest amount of time and trigger the camera to take an image at specific points along the scan. In such applications, you need to send a digital signal (called a breakpoint) to the trigger line of a camera to command it to take an image based on a specific position reached by the XY stage. This functionality is called position compares (breakpoints), which can be classified into absolute breakpoints, relative position breakpoints, modulo breakpoints, periodically occurring breakpoints, and buffered breakpoints.

In some cases, you may need to synchronize position with some measurement occurring externally to the motion controller. You can think of this conceptually as a conveyor belt application running at a constant velocity. Each time a manufactured item moves down the conveyor belt and breaks a light beam across the belt, a stamp is placed on the item, a set distance from the point where the beam is broken. This allows all items to be stamped in the same relative position without necessarily being equally spaced along the conveyor belt. You need to command amove of a specified length relative to the position at which a trigger input is received to perform such a stamping operation. This functionality is called high-speed position captures (triggers), which could be buffered or nonbuffered.

In vision-guided motion applications, you may need to log position and velocity data periodically for off-line analysis and correlation with vision data obtained concurrently. This functionality is known as time-sampled data logging. Data is collected at hardware/firmware timed period and stored in the motion controller's internal memory. This data can be retrieved from the motion controller asynchronously.

Position Compares (Breakpoints)

You can configure a motion controller to generate a trigger signal or toggle the state of an I/O line when the motor reaches a specified position. This specified position is called a breakpoint and is similar to a switch that is triggered after the motor passes a certain position. To use breakpoints, you must have a board with closed-loop encoder feedback, because the controller needs the encoder position to determine where the breakpoint occurs. You can specify breakpoints for both servo motors and stepper motors. You can use breakpoints as a trigger for another piece of hardware because it sends a signal to an I/O line. For example, when a motor reaches a certain position, you can use a breakpoint signal to initiate an image capture or a data acquisition operation. The five ways in which breakpoints can be configured are as follows:

Absolute Mode: With absolute position breakpoints, you can trigger external activities as the motors reach specified positions. For example, if you need to use an image acquisition device to capture an image from a certain position while the device under test is in continuous motion, the motion controller must be able to trigger the image acquisition device as it reaches those positions. The current position is continuously compared against the specified breakpoint position by the encoder circuitry to produce a latency of less than 100 ns. After a breakpoint triggers, you must re-enable it for the breakpoint to work again. In certain cases, such as buffered and periodic breakpoints, the motion controller automatically re-enables the breakpoints.

Relative Mode: Relative position breakpoints trigger events on the basis of a change in position relative to the position at which the breakpoint was enabled. Instead of keeping track of absolute positions and the current position, you can use relative breakpoints to specify the breakpoint relative to the position where the breakpoint is enabled. For example, if you are creating a motion control system to control the 2D movement of a microscope, you might use relative position breakpoints to move the microscope a specific distance in a direction, and then hit a breakpoint that triggers a camera snap. The relative breakpoint is useful in this example because the current position is not important. The application must move the axis a specific number of counts from wherever it is, and then generate a breakpoint.

Periodic Mode: You can program the motion controller to generate multiple breakpoints at fixed and exact intervals, regardless of the direction of travel or velocity. Periodic breakpoints require that you specify an initial breakpoint and an ongoing repeat period.When enabled, the periodic breakpoints begin when the initial breakpoint occurs. From then on, a new breakpoint occurs each time the axis

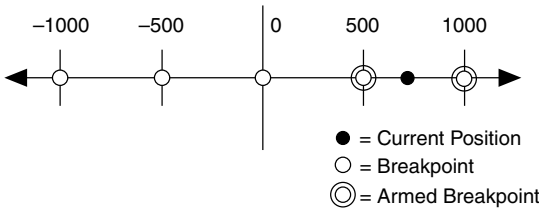


FIGURE 15.11 Periodic breakpoint every 1000 counts/steps.

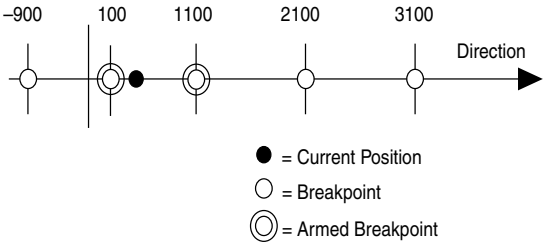


FIGURE 15.12 Breakpoint modulus of 500.

moves a distance equal to the repeat period, with no re-enabling required. For example, if an axis is enabled at position zero, the initial breakpoint is set for position 100, and the breakpoint period is set at 1000, then the axis behaves as shown in Figure 15.11.

Modulo Mode: Modulo breakpoints use a breakpoint window that defines an area around the current position. The two breakpoints around the current position are always enabled. The breakpoint modulus creates a repeat period for the breakpoints, and the breakpoint position is the offset from absolute zero. For example, to create a breakpoint every 500 counts, set the repeat period to 500 and the breakpoint position to 0. If the breakpoint is enabled when the axis is at 710, the breakpoints at 1000 and 500 are both armed, as shown in Figure 15.12.

Buffered Mode: Buffered breakpoints are a way of specifying absolute or relative breakpoint positions in a buffer that can be preloaded into the motion controller memory. The motion controller then automatically arms breakpoint positions from the buffer as and when the breakpoints triggers. The, arming of breakpoints occurs on a firmware-timed basis, which provides higher bandwidth.

Position Captures (Triggers)

Some motion control applications require that you execute amove and record the position where external triggers occur. For example, you might be aligning two fiber-optic cables, in which case, the maximum optical power needs to correspond with the alignment position. To align the fibers, the external device that is recording the optical power must trigger the motion controller so that you can synchronize and analyze positions and optical power measurements. This functionality is known as high-speed capture or trigger inputs. The implementation for high-speed capture is divided into the buffered and nonbuffered high-speed capture methods:

Buffered High-Speed Captures: With buffered high-speed capture, you can create a buffer that holds captured positions that you can read asynchronously from the motion controller. The motion controller automatically arms the next high-speed capture, and writes the captured high-speed data into its onboard buffer. The enabling of high-speed capture occurs on a firmware-timed basis, which provides better frequency than the nonbuffered high-speed capture method.

Nonbuffered High-Speed Captures: With nonbuffered, high-speed capture, you can configure a single high-speed capture event. For multiple high-speed captures, you must re-enable the high-speed capture each time before it triggers.

Time-Sampled Data Logging
Some motion controllers can acquire a hardware-timed buffer of position and velocity data for time-sampled data logging. After you command the motion controller to acquire position and velocity data, a separate acquire data task is created in the motion controller, which reads time-sampled position and velocity data into a FIFO buffer on the motion controller. You can read data in from this buffer number of samples that consist of position data, in counts or steps, and velocity data, in counts/s or asynchronously from the user interface on the host computer. Typically, the FIFO buffer holds a certain steps/s.

15.3.2 Trajectory Generation

The trajectory generator on a motion controller is responsible for the path planning on the basis of geometry specified by the user. Different kinds of geometries are typically supported: straight lines, circular arcs, helical arcs, and spherical arcs. In addition to the predefined geometries, the user can specify a custom geometry via a buffer of points that the motion controller uses to interpolate through using a cubic spline algorithm. This is known as a contoured move.

The user can also specify the velocity profiles that he wants used when traversing desired geometries. This can be specified using move constraints that include maximum velocity, maximum acceleration, and maximum deceleration and jerk (rate of change of acceleration/deceleration). These parameters are used to generate a trapezoidal or s-curve profile. The trajectory generator on the motion controller ensures that these move constraints are never violated while making a move. In addition to this, the user can specify custom profiles by providing position, velocity, and time data (Position Velocity Time, PVT). The trajectory generator also typically supports blending velocity profiles. In this case, the users can concatenate multiple profiles specifying the point of concatenation.

The output of the trajectory generator are set points that are fed to the control loop to act on the basis of the path that needs to be followed.

To be highly reactive and allow for on-the-fly changes, the trajectory generator on the motion controller creates these set points in real time. This allows users to pre-empt the current trajectory by changing end positions, velocity, and acceleration. It also allows for stopping and restarting a move that is currently in progress.

15.3.2.1 Straight-Line Moves

A straight-line move executes the shortest move between two points. Straight-line moves can be position based or velocity based.

Position Based
Position-based straight-line moves use the specified target position to generate the move trajectory. For example, if the motor is currently at position zero, and the target position is 100, a position-based move creates a trajectory that moves 100 counts (steps).

The controller requires the following information to move to another position in a straight line:

- Start position—current position, normally held over from a previous move or initialized to zero
- End position—also known as the target position, or where you want to move to
- Move constraints—maximum velocity, maximum acceleration, maximum deceleration, and maximum jerk

The motion controller uses the given information to create a trajectory that never exceeds the move constraints and that moves an axis or axes to the end position you specify. The controller generates the trajectory in real time, so you can change any of the parameters while the axes are moving. Figure 15.13 illustrates a typical algorithm for a position-based straight-line move.

Velocity Based
Some motion applications require moves that travel in a straight line for a specific amount of time at a given speed. This type of move is known as velocity profiling or jogging. You can use a motion control

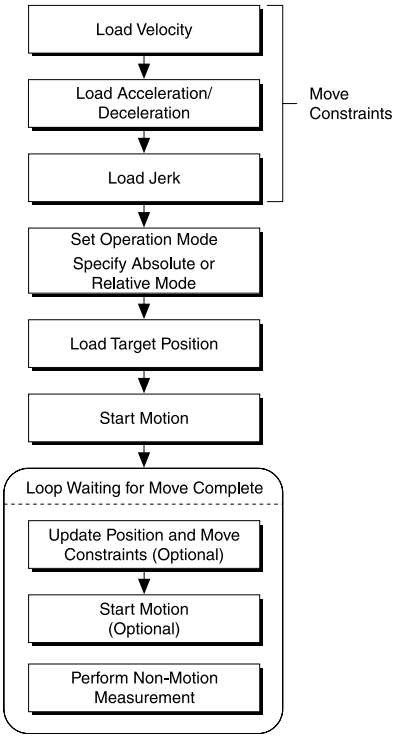


FIGURE 15.13 Position-based straight-line move algorithm.

application to move a motor at a given speed for a specific time, and then change the speed without stopping the axis. The sign of the loaded velocity specifies the direction of motion. Figure 15.14 illustrates an algorithm for a typical velocity-based straight-line move.

15.3.2.2 Arc Moves

An arc move causes a coordinate space of axes to move on a circular, spherical, or helical path. You can move two-dimensional vector spaces in a circle only on a 2D plane. You can move a three-dimensional vector space on a spherical or helical path. Each arc generated by the motion controller passes through a cubic spline algorithm that ensures the smoothest arc. A cubic spline algorithm generates multiple points between every two points of the arc, ensuring smooth motion, minimum jerk, and maximum accuracy at all times.

Circular Arcs

A circular arc defines an arc in the XY plane of a 2D or 3D coordinate space. The arc is specified by a radius, start angle, and travel angle. In addition, like all coordinate space moves, the arc uses the values of move constraints—maximum velocity, maximum acceleration, and maximum deceleration.

To move axes in a circular arc, the motion controller needs the following information:

- Radius—specifies the distance from the center of the arc to its edge.
- Start Angle—orients the arc on its plane using the starting point as an axis to spin around. Because the starting point for a new arc is fixed on the basis of the current position, moving its center around the starting point alters the orientation of a new arc.
- Travel Angle—indicates how far the arc travels in a 360° circle. For example, a travel angle of 90° executes a quarter circle, a travel angle of 360° creates a full circle, and a travel angle of 720° creates two full circles (Figure 15.15).

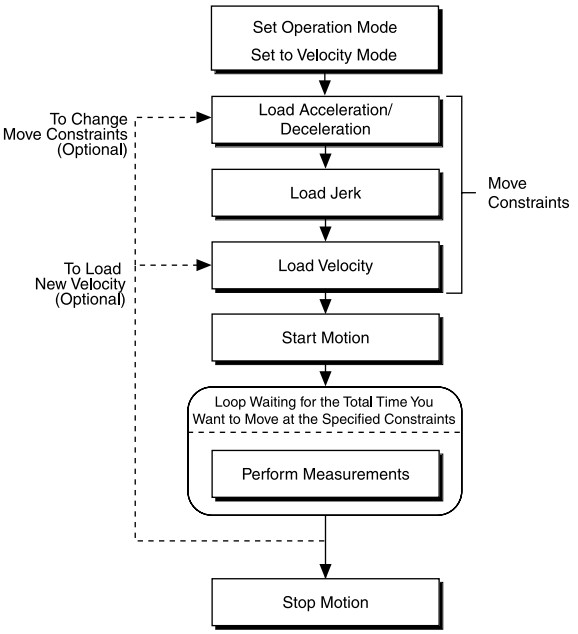


FIGURE 15.14 Velocity-based straight-line move algorithm.

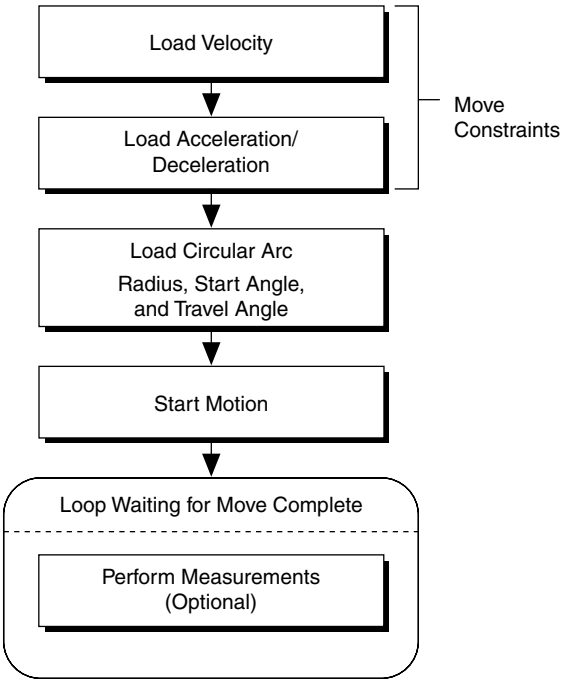


FIGURE 15.15 Circular arc move algorithm.

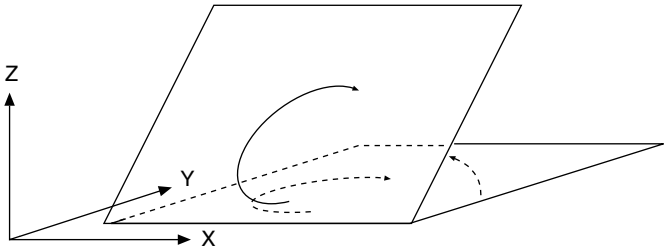


FIGURE 15.16 Changing pitch by rotating the X-axis.

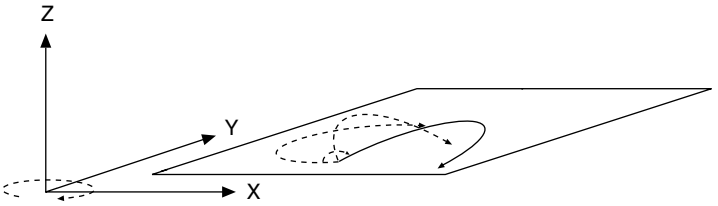


FIGURE 15.17 Changing yaw by rotating the Z-axis.

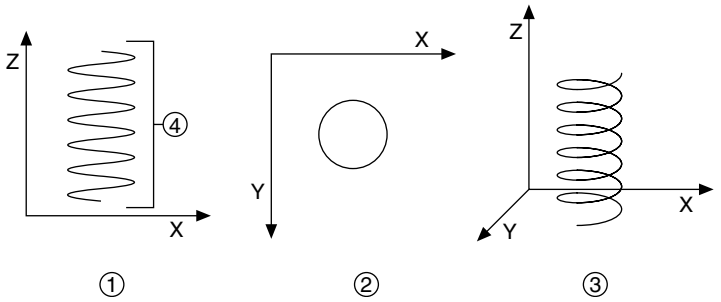


FIGURE 15.18 Helical Arc showing (1) side view, (2) top view, (3) isometric view, and (4) linear travel.

Spherical Arcs

A 3D spherical arc defines a 2D circular arc in the XY plane of a coordinate system that is transformed by rotation in pitch and yaw from the normal 3D coordinate space (XYZ), as shown in Figures 15.16 and 15.17.

In the transformed X′Y′Z′ space, the 3D arc is reduced to a simpler 2D arc. The 3D arc is defined as a 2D circular arc in the X′Y′ plane of a transformed vector space X′Y′Z′. This transformed vector space, X′Y′Z′, is defined in orientation only, with no absolute position offset. Its orientation is relative to the XYZ vector space, and is defined in terms of pitch and yaw angles.When rotating through the pitch angle, the Y and Y′ axes stay aligned with each other while the X′Z′ plane rotates around them. When rotating through the yaw angle, the Y′ axis never leaves the original XY plane, as the newly-defined X′Y′Z′ vector space rotates around the original Z-axis. The radius, start angle, and travel angle parameters also apply to a spherical arc that defines the arc in two dimensions.

Helical Arc

Ahelical arc defines an arc in a 3Dcoordinate space that consists of a circle in the XY plane and synchronized linear travel in the Z-axis. The arc is specified by a radius, start angle, travel angle, and Z-axis linear travel. Linear travel is the linear distance traversed by the helical arc on the Z-axis, as shown in Figure 15.18.

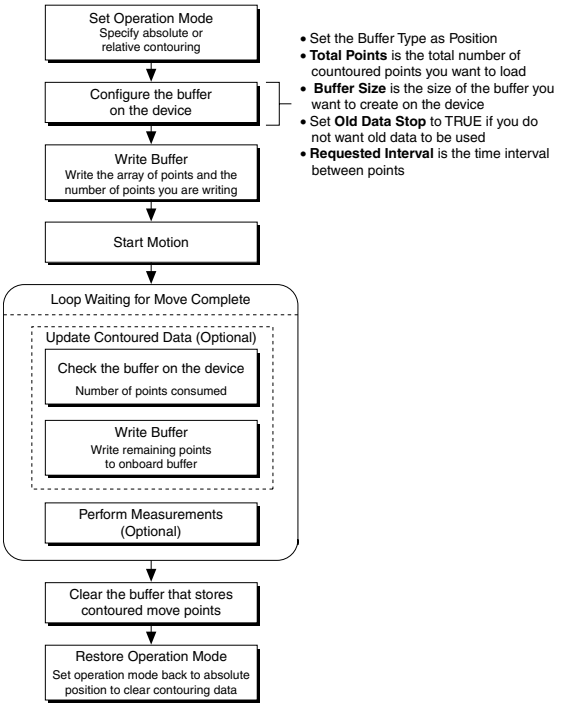


FIGURE 15.19 Contoured move algorithm.

15.3.2.3 Contoured Moves

A contoured move moves an axis or a coordinate space of axes in a pattern that you define. The trajectory generator on the motion controller is not used during a contoured move. The controller takes position data in the form of an array, and splines the data before outputting it to the control loop.

Contoured moves are useful when you want to generate a trajectory that cannot be constructed from straight lines and arcs. To ensure that the motion is smooth with minimum jerk, the motion controller creates intermediate points using a cubic spline algorithm. The move constraints commonly used to limit other types of moves, such as maximum velocity, maximum acceleration, maximum deceleration, and maximum jerk, have no effect on contoured moves. These need to be embedded in the series of contoured points provided (by spacing the points) (Figure 15.19).

15.3.2.4 Position Velocity Time Profiles

Position velocity time moves give you maximum control over the velocity of the move at certain times throughout the move. PVT moves specify a series of move segments by giving several move points for each axis. In addition, PVT moves specify both the time, in ms, each move segment takes and the velocity at which all axes are traveling at the end of each segment.

The following table demonstrates typical PVT inputs:

X	Y	Z	V	T
4317	4317	4317	1000	750
12245	11245	13245	500	1500
15173	15173	15173	500	2500

T: Time begins at 0 for each PVT move, and is cumulative through the end of the move. In this example, there are three segments of duration 750, 750, and 1000.

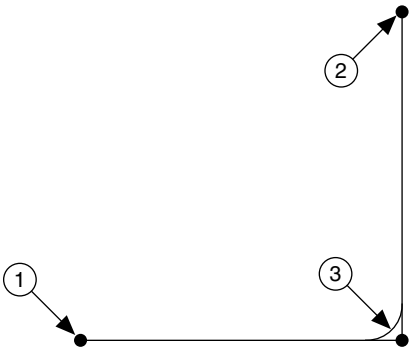


FIGURE 15.20 Two blended straight-linemoves showing (1) starting position, (2) end point, and (3) corner rounded by blending.

V: PVT moves assume a beginning velocity of zero. Therefore, you must bring each axis to a complete stop in the previous move before beginning a PVT move.

X, Y, Z: PVT moves start a position counter for each axis at 0, independent of where you are in relation to the original reference move. The numbers in the spreadsheet or text file are absolute in relation to the new counter. In this example, there are three move segments of lengths 4317, 7928, and 2928, respectively.

15.3.2.5 Blending

Blending, also called velocity blending, superimposes the velocity profiles of two moves to maintain continuous motion. Blending is useful when continuous motion between concatenated move segments is important. Examples of some applications that can use blending are scanning, welding, inspection, and fluid dispensing. Blending must occur on velocity profiles of two move segments, so the end positions of each move segment may or may not be reached. For example, if you are blending two straight-line moves that form a 90° angle, the blended move must round the corner to make the move continuous. In this case, the move never reaches the exact position where the two straight lines meet, but instead follows the rounded corner, as shown in Figure 15.20.

Motion controllers can perform blending between two straight-line moves, between two arc moves, or between straight-line and arc moves. Blending typically is not supported for reference and contoured moves.

Superimposing Two Moves

Superimposing two moves is the most common use of blending. In this case, the motion controller tries to maintain continuous motion by superimposing the two move segments such that the second move segment starts its profile while the first move is decelerating, as shown in Figure 15.21.

The velocity during the superimposition depends on the cruising velocity, deceleration, and jerk of the first move segment, and the jerk, acceleration, and cruising velocity of the second move segment.

Blending Moves after the First Move Is Complete

Blending moves after the first move is complete; this causes the first move segment to come to a complete stop before starting the profile of the second segment, as shown in Figure 15.22. This type of blending is useful if you want to start two move segments, one after the other, with no delay between them.

Blend after Delay

You can blend two moves after a delay at the end of the first move. Blending in this manner is useful if you want to start two move segments after a deterministic delay. The two move segments can be either straight-line moves or arc moves.

Because blending occurs on velocity profiles, the effect of reaching the end positions of the move segments and the maximum velocity depends on the velocity, acceleration, deceleration, and jerk loaded

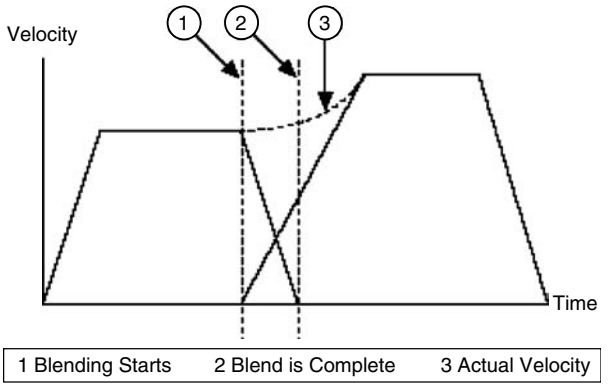


FIGURE 15.21 Superimposing two moves.

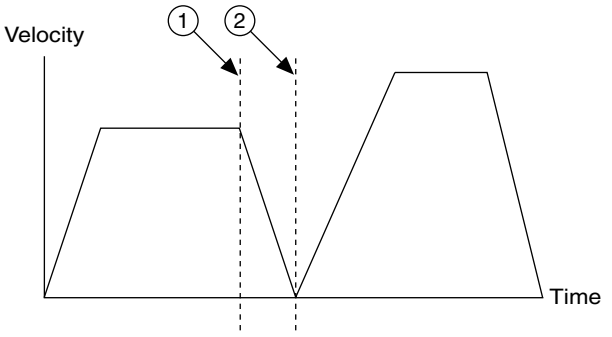


FIGURE 15.22 Blending after move complete showing (1) blending starting point and (2) blending complete.

for the two move segments. Because two move segments are always used while blending, it is very important that you wait for the blend to complete before loading the next move segment you want to blend.

15.3.2.6 Second- and Third-Order Profiles (Trapezoidal/s-Curve)

The trajectory generator uses the commanded parameters of position, acceleration, and velocity for a particular move to determine how much time it spends in the three primary move segments—acceleration, constant velocity, and deceleration—that define a velocity profile.

Trapezoidal Velocity Profile

When you use a trapezoidal profile, the axes accelerate at the acceleration value you specify, and then cruise at the maximum velocity you load. On the basis of the type of move and the distance being covered, it may be impossible to reach the maximum velocity you set. The velocity of the axis, or axes, in a coordinate space never exceeds the maximum velocity loaded. The axes decelerate to a stop at their final position, as shown in Figure 15.23.

S-curve Velocity Profile

The acceleration and deceleration portions of an s-curve motion profile are smooth, resulting in less abrupt transitions, as shown in Figure 15.24. This limits the jerk in the motion control system, but increases cycle time. The value by which the profile is smoothed is called the maximum jerk or s-curve value.

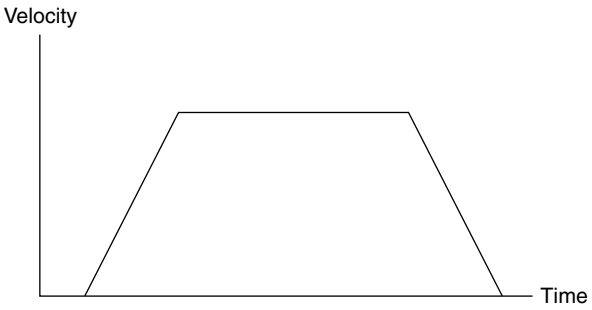


FIGURE 15.23 Trapezoidal move profile.

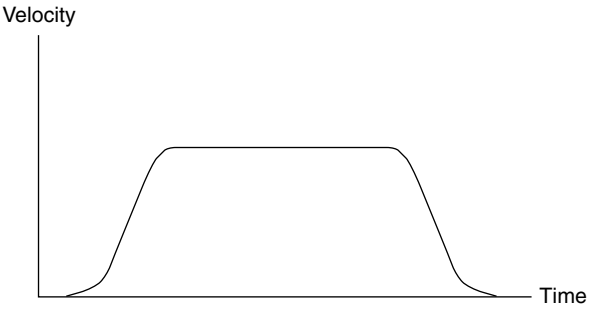


FIGURE 15.24 S-curve move profile.

15.3.2.7 Multiple Axis Synchronization (Coordinate/Vector Spaces)

With the exception of the arc move, you can execute all basic moves on either a single axis or on a coordinate space. A coordinate space is a logical grouping of axes, such as the XYZ axes. Arc moves always execute on a coordinate space. If you are performing a move that uses more than one axis, you must specify a coordinate space made up of the axes the move will use.

Typically, there exists a function in the motion controller to configure a coordinate space. This function creates a logical mapping of axes and treats the axes as part of a coordinate space. The function then executes the move generated by the trajectory generator on the vector, and treats all the move constraints as vector values.

Multistarts versus Coordinate Spaces

Coordinate spaces always start and end the motion of all axes simultaneously. You can use multistarts to create a similar effect without grouping axes into coordinate spaces. Using a multistart automatically starts all axes virtually simultaneously. To simultaneously end the moves, you must calculate the move constraints to end travel at the same time. In coordinate spaces, this behavior is calculated automatically.

15.3.3 Control Loop

15.3.3.1 Spline Interpolation

The trajectory generator and the proportional, integral, derivative (PID) control loop need not execute at the same loop rates.While the PID control loop needs to run at microseconds, the trajectory generator may generate a new set point every millisecond. To maintain the steady flow of set points from the trajectory generator to the PID control loop, a motion controller needs a spline interpolator that interpolates between set points given by the trajectory generator to create finer set points to be acted on by the control loop (Figure 15.25).

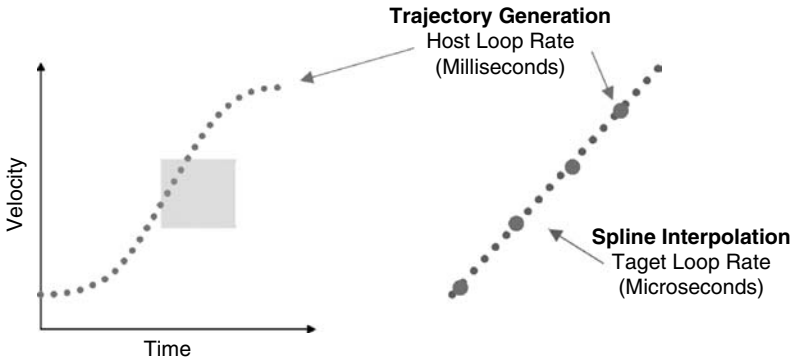


FIGURE 15.25 Spline engine function uses cubic spline algorithm to calculate interpolated positions between two positions from the trajectory generator.

An example of a spline interpolator is a cubic spline algorithm that uses four set points to calculate interpolated positions between two positions calculated by the trajectory generator. The polynomial used to calculate interpolated points can be represented as:

$$P(t) = a_3 t^3 \times a_2 t^2 \times a_1 t^1 \times a_0 t^0$$

Given four set points, P_0, P_1, P_2, and P_3, the spline coefficients are determined in real time as:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

The number of interpolated points created depends on the rate of the trajectory generator loop and control loop. For example, if the trajectory generator loop runs at 1 ms and the control loop runs at 100 μs, the spline engine calculates 1 ms/100 μs =10 interpolated points. Using the spline interpolator results in smoother motion and allows you to run the trajectory generator loop slower than the control loop.

15.3.3.2 Position and Velocity Loops

The control loop creates the command signal on the basis of the set point provided by the trajectory generator. In most cases, the control loop includes both a position and a velocity loop, but in some cases the control loop may include only a position loop. The position is typically read from encoders, but also may be read from analog inputs. The velocity is calculated from the position values, and may be read directly from a velocity sensor, such as a tachometer. Because feedback is not required for stepper motors, the control loop concerts the set point generated by the trajectory generator into stepper signals (step/direction).

The position and velocity control loops on the motion system correct for errors and maintain tight control of the trajectory. The position and velocity loops typically incorporate the PIDalgorithm. The PID algorithm consists of three basic coefficients: proportional, integral, and derivative, which are varied to get optimal response (Figure 15.26).

In addition to parameters such as proportional gain, integral gain, and derivative gain, the PID algorithm-implemented part of a motion control system includes additional parameters such as velocity feedback, velocity feed forward, and acceleration feed forward for optimal position control.

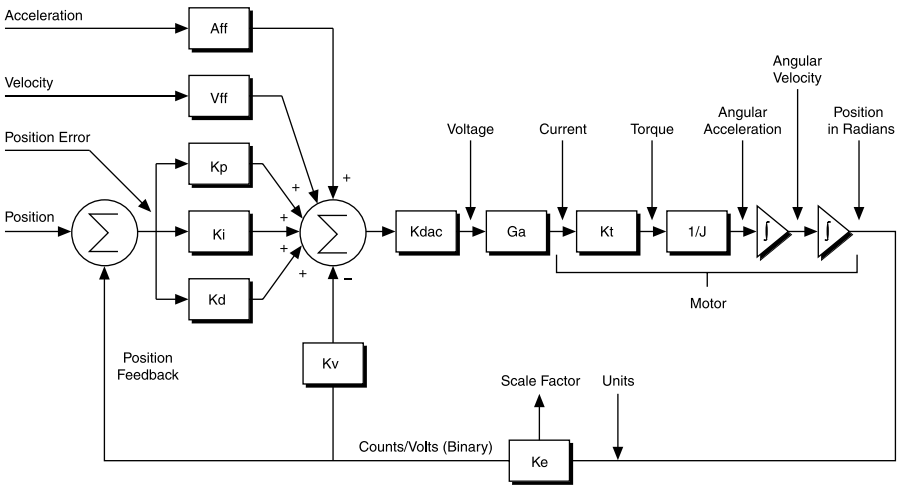


FIGURE 15.26 PID servo loop.

Kp (Proportional Gain)
The proportional gain (Kp) determines the contribution of restoring force that is directly proportional to the position error. This restoring force functions in much the same way as a spring in a mechanical system. An axis with too small a value of Kp is unable to hold the axis in position and is very soft. Increasing Kp stiffens the axis and improves its disturbance torque rejection. However, too large a value of Kp often results in instability.

Ki (Integral Gain)
The integral gain (Ki) determines the contribution of restoring force that increases with time, ensuring that the static position error in the servo loop is forced to zero. This restoring force works against constant torque loads to help achieve zero position error when the axis is stopped. In applications with small static torque loads, this value can be left at its default value of zero (0). For systems having high static torque loads, this value should be tuned to minimize position error when the axis is stopped. Although nonzero values of Ki cause reduced static position error, they tend to cause increased position error during acceleration and deceleration. This effect can be mitigated through the use of the integration limit parameter. Too high a value of Ki often results in servo loop instability.

Kd (Derivative Gain)
The derivative gain (Kd) determines the contribution of restoring force proportional to the rate of change (derivative) of position error. This force acts much like viscous damping in a damped spring and mass mechanical system. A shock absorber is an example of this effect. A nonzero value of Kd is required for all systems that use torque block amplifiers, where the command output is proportional to motor torque, for the servo loop operation to be stable. Too small a Kd value results in servo loop instability. With velocity block amplifiers, where the command output is proportional to motor velocity, it is typical to set Kd to zero or a very small positive value.

Kv (Velocity Feedback)
You can use a primary or secondary feedback encoder for velocity feedback. Setting the velocity feedback gain (Kv) to a value other than zero (0) enables velocity feedback using the secondary encoder, if configured, or the primary encoder if a secondary encoder is not configured. The derivative gain scales the derivative of the position error, which is the difference between the instantaneous trajectory position and the primary feedback position. Velocity feedback is estimated through a combination of speed-dependent algorithms.

Vff (Velocity Feedforward)
The velocity feedforward gain (Vff) determines the contribution in the command output that is directly proportional to the instantaneous trajectory velocity. This value is used to minimize following error during the constant velocity portion of a move and can be changed at any time to tune the PID loop. Velocity feedforward is an open-loop compensation technique and cannot affect the stability of the system. However, if you use too large a value for Vff, following error can reverse during the constant velocity portion, thus degrading performance, rather than improving it. Velocity feedforward is typically used when operating in PIVff mode with either a velocity block amplifier or substantial amount of velocity feedback. In these cases, the uncompensated following error is directly proportional to the desired velocity.

Aff (Acceleration Feedforward)
The acceleration feedforward gain (Aff) determines the contribution in the command output that is directly proportional to the instantaneous trajectory acceleration. Aff is used to minimize following error (position error) during acceleration and deceleration and can be changed at any time to tune the PID loop. Acceleration feedforward is an open-loop compensation technique and cannot affect the stability of the system. However, if you use too large a value of Aff, following error can reverse during acceleration and deceleration, thus degrading performance, rather than improving it.

Dual-Loop Feedback
Motion control systems often use gears to increase output torque, increase resolution, or convert rotary motion to linear motion. The main disadvantage of using gears is the backlash created between the motor and the load. This backlash can cause a loss of position accuracy and system instability. The control loop on the motion system corrects for errors and maintains tight control over the trajectory. The control loop consists of three main parts—proportional, integral, and derivative—known as PID parameters. The derivative part estimates motor velocity by differentiating the following error (position error) signal. This velocity signal adds, to the loop, damping and stability. If backlash is present between the motor and the position sensor, the positions of the motor and the sensor are no longer the same. This difference causes the derived velocity to become ineffective for loop damping purposes, which creates inaccuracy in position and system instability.

Using two position sensors for an axis can help solve the problems caused by backlash. As shown in Figure 15.27, one position sensor resides on the load and the other on the motor before the gears. The motor sensor is used to generate the required damping and the load sensor for position feedback. The mix of these two signals provides the correct position feedback with damping and stability.

15.3.3.3 Commutation

When a current-carrying conductor is placed in a magnetic field, it experiences a force. Motors are based on this principle of physics. Motors comprise of permanent magnets and current-carrying conductors in a mechanical assembly. Maintaining the correct direction of current and magnetic fields is critical to proper operation of the motor. This functionality is called commutation.

In case of brushed DC servo motors, a physical brush performs commutation by switching the current between different conductors. However, in case of brushless servo motors, three phases of current are

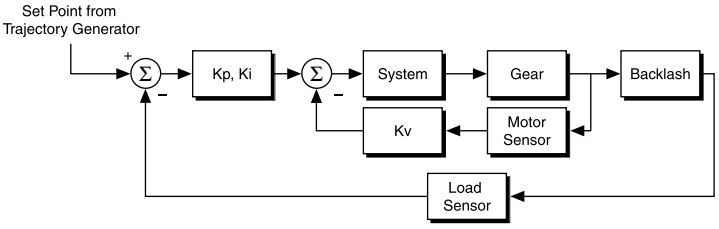


FIGURE 15.27 Dual-loop feedback.

electrically commutated. This requires commutation to be performed either by the motion controller or the drive.

15.3.3.4 Current Loop (Torque Loop)

A current loop takes a set point from the position and velocity loop and uses feedback from a current sensor to generate a voltage output signal. This voltage is used to control the power transistors in the drive to ensure that the right amount of torque is being generated. While in brushed DC servo motors the current command is directly proportional to the current in one winding in the motor, in brushless servo motors it is dependent on current in three windings. As a result, brushless servo motors often come paired with drives that include commutation and a pretuned current loop.

15.4 Motion Controller Hardware

A motion controller can be implemented on dedicated hardware connected to an analog drive, independent of the PC processor that hosts the user application. It can also be designed to share resources on the PC processor with the user application with some functions being executed within a digital drive, thus eliminating the need for dedicated motion controller hardware. These two architectures for implementing a motion controller have their advantages and disadvantages.

15.4.1 Architecture of an Analog Motion Controller (Dedicated Processor)

A motion controller is responsible for executing supervisory control, trajectory generation, and control-loop functions. In case of an analog motion controller, these functions are executed on dedicated hardware that communicates with the host PC via an interface such as the PCI bus, USB, Ethernet, Serial RS 232, and so forth. Commands from the user application running on the host computer are sent to the motion controller via this interface. The motion controller performs supervisory control, trajectory generation, and I/O synchronization, and executes position and velocity control loops. The motion controller then sends commands in the form of 10V analog voltage signals to an analog drive that closes the current loop. There are four main hardware components in a typical analog motion controller.

15.4.1.1 Microprocessor

A microprocessor runs amultitasking, real-time operating system, and handles host communication with the PC and supervisory control including command processing and fault handling.

15.4.1.2 Digital Signal Processor

A digital signal processor is used for math-intensive operations such as trajectory generation and position and velocity loops.

In some motion controllers, all the supervisory control, trajectory generation, and control-loop functions are performed on a single microcontroller that includes I/O interfaces, memory, and clock in addition to the CPU in an integrated circuit.

15.4.1.3 Field-Programmable Gate Array

Afield-programmable gate array (FPGA) is used to perform I/O synchronization functions such as encoder decoding, position capture and position compare, motion I/O processing, and stepper pulse generation.

15.4.1.4 Motion I/O

An analog motion controller connects to different types of motors as follows:

1. Brushed DC servo motor—An analog motion controller provides a ± 10 V analog output signal to a servo drive that closes the current loop to move a brushed DC servo motor.
2. Brushless servo motor—An analog motion controller provides a ± 10 V analog output signal to a servo drive that performs sinusoidal commutation and closes the current loop to move a brushless

servo motor. In case the drive does not perform sinusoidal commutation, the analog motion controller provides two analog output signals to the drive. The motion controller in this case performs sinusoidal commutation in addition to closing the position and velocity loops.

3. Stepper motor—An analog motion controller provides step and direction signals to a stepper drive to move a stepper motor. The motion controller performs step generation to output step and direction signals.

15.4.2 Architecture of a SoftMotion-Based Motion Controller (Shared Processor with User Application)

In case of a soft motion-based motion controller, functions such as supervisory control and trajectory generation execute on a real-time operating system shared with the operating system used for the user interface on the same processor. Control loops are executed within the drive, thus eliminating the need for dedicated motion controller hardware. Some configurations allow for the control loops to be also closed on the shared processor.

Common operating systems that are used for soft motion-based motion controllers include off-the shelf RTOSes such as QNX, Pharlap ETS, RTX, VxWorks, or Windows CE or, in many cases, proprietary implementations by motion control vendors.

15.4.2.1 PC/PLC-Based with Analog Interface (Motion I/O Connected to the PC/PLC via PCI or ISA Bus)

A PLC-based SoftMotion controller runs on the real-time operating system on the PLC. It connects to analog drives with a ± 10 V analog interface. In this case, the motion controller performs supervisory control, trajectory generation on the PLC processor, and closes position and velocity control loops on the PLC motion control modules. These modules connect to analog drives that handle the task of commutation, closing the current loops. In this case, the motion controller tasks run at the highest priority on the PLC processor and coexist with the user's sequential logic tasks.

A PC-based SoftMotion controller runs on a real-time operating system (VxWin, VxCE, InTime) or real-time extension (RTX, TwinCAT) that runs on a PC and shares the processor with Windows or any desktop OS. In this case, the motion controller performs supervisory control, trajectory generation on the PC processor, and closes position and velocity control loops on the PCI/ISA-based motion control card plugged into the PC. These cards connect to analog drives that handle the task of commutation, closing the current loops. In this case, since the motion controller runs on the real-time OS on the PC, it is guaranteed bandwidth and priority over Windows or any desktop OS sharing the processor with it.

15.4.2.2 PC/PLC-Based with Deterministic Digital Network (Distributed Motion, I/O Connected to the PC/PLC via a Deterministic Network)

As discussed in Section 15.4.2.1, a SoftMotion controller shares the processor with either the PLC tasks or tasks running on the Windows or desktop OS on a PC. The primary difference between this mode of operation and the former is that in this mode the communication of the motion controller with the drives is via a deterministic bus. This solves a number of issues faced in Section 15.4.2.1 such as connectivity and signal integrity. This architecture is also termed as "DistributedMotion." Similar to the previous case, the motion controller performs supervisory control, trajectory generation, and in some cases also closes the position and velocity control loops. The drives close handle the commutation and close the current loops. They also support closing the position and the velocity loops that allow for lower bandwidth utilization of the deterministic network.

A primary requirement for such a configuration is that the digital network connecting the motion controller to the digital drive is deterministic and that it is able to transmit data deterministically. Emerging networks include IEEE 1394, Ethernet PowerLink, EtherCAT, SERCOS III, PROFInet (IRT), EtherNET/IP, CANopen, and PROFIbus (MC).

IEEE 1394
IEEE 1394 (isochronous) (www.1394ta.org) is a high speed, real-time digital network based on the serial bus. It requires an IEEE 1394 interface on the host PC. It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

Ethernet PowerLink
Ethernet PowerLink (www.ethernet-powerlink.org) is an Ethernet-based digital network based on TCP/IP and isochronous slot communication network management (SCNM). It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

EtherCAT
EtherCAT (www.ethercat.org) is an Ethernet-based digital network based on the principle of reading and writing EtherCAT telegram packets as they pass through the different slave devices in cyclic order. It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

SERCOS III
SERCOS-III (www.sercos.com) is an Ethernet-based digital network based on TCP/IP and the principle of cyclic data transfer with an exact time pattern. It requires the use of a master to control the timing behavior of the data transfer to the devices on the network. It provides submillisecond response rates for use with soft motion-based motion controllers.

PROFInet (IRT)
PROFInet (isochronous real time) (www.profinet.com) is an Ethernet-based digital network that leverages TCP/IP and DCOM standards, and works with internal switches in all devices to handle synchronization. It provides submillisecond response rates for use with soft motion-based motion controllers.

EtherNET/IP
EtherNET/IP (www.odva.org) is an Ethernet-based digital network that is based on TCP/IP and Control switches and uses time stamps for synchronization. It provides clock-cycle synchronous data transmission, and Information Protocol (CIP) standards. It supports a star topologywith devices connected via managed and millisecond response rates for use with soft motion-based motion controllers that performsupervisory control and trajectory generation.

CANopen
CANopen (www.canopen.de) is a layer on CAN that is based on the DS301 communication profile and the DS402 device profile for motion control. It supports deterministic data transmission, and millisecond response rates for use with soft motion-based motion controllers that perform supervisory control and trajectory generation. The control loops are closed on the digital drive.

PROFIbus (MC)
PROFIbus (Motion Control) (www.profibus.org) is based on the PROFIbus digital network with enhancements to response times and clock synchronization. PROFIbus (MC) has response rates of several milliseconds making it suitable for soft motion-based motion controllers that perform supervisory control and trajectory generation. The control loops are closed on the digital drive.

15.5 Summary

Motion control systems are used to control the position, velocity, and torque of a rotational or linear electromechanical device. A typical motion control system consists of a variety of components including user interface, motion controller, drive, motor, feedback device, and motion I/O.The functions of a motion controller include trajectory generation, supervisory control, and a position and velocity feedback control loop.

16

Real-Time Monitoring and Control

Gerardo Garcia
National Instruments, Inc.

16.1 Introduction to Real-Time Systems

Real-time operating systems originated with the need to solve two main types of applications: event response and closed-loop control systems. Event response applications require a response to a stimulus in a determined amount of time; an example of such a system is an automotive airbag system. Closed-loop control systems continuously process feedback in order to adjust an output; an automotive cruise control system is an example of a closed-loop control system. Both of these types of systems require the completion of an operation within a specific deadline. This type of performance is referred to as determinism.

Real-time systems are sometimes classified as "soft" or "hard." Soft real-time typically means the utility of a system is inversely proportionate to how long it has been since a deadline is missed. For example, when pressing a cell phone button to answer an incoming call, the connection must be established soon after the button has been pressed. However, the deadline is not mission-critical and small delays can be tolerated. Hard real-time systems are those in which the utility of a system becomes zero in the event of a missed deadline. An automotive engine control unit (ECU) must process incoming signals and calculate spark plug timing within a deadline. If the deadline is missed, the engine will fail to operate correctly. The usefulness of a task after a deadline is missed depends on whether the system is a soft real-time or a hard real-time system, as shown in Figure 16.1.

Operating systems such as Microsoft Windows and Mac OS provide an excellent platform for developing and running your noncritical measurement and control applications. However, because these operating

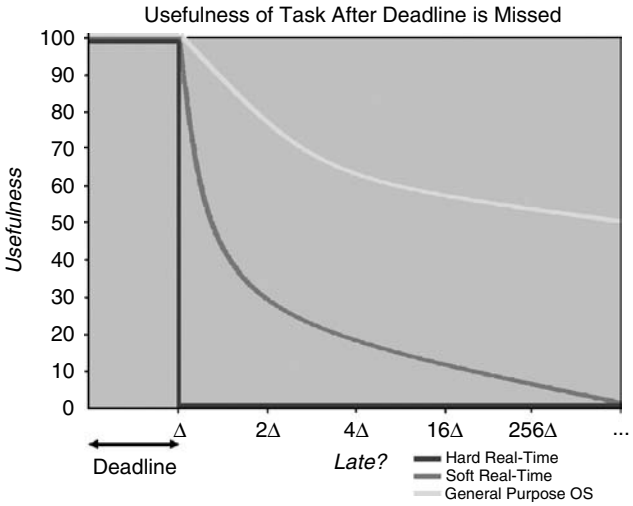


FIGURE 16.1 Differences between hard and soft real-time.

systems are designed for general purpose use, they are not the ideal platforms for running applications that require deterministic performance or extended uptime.

General-purpose operating systems are optimized to run a variety of applications simultaneously, ensuring that all applications receive some processing time. These operating systems must also respond to interrupts from peripherals such as the mouse and keyboard. The user has limited control regarding how these tasks are handled by the processor. As a result, high-priority tasks can be preempted by lower priority tasks, making it impossible to guarantee a response time for your critical applications.

In contrast, real-time operating systems give users the ability to prioritize tasks so that the most critical task can always take control of the processor when needed. This property enables you to program an application with predictable results.

Real-time operating systems are required when the processor is involved in operations such as closed loop control and time-critical decision making. These applications require timely decisions to be made on the basis of incoming data. For example, an I/O device samples an input signal and sends it directly to memory. Then, the processor must analyze the signal and send the appropriate response to the I/O device. In this application, the software must be involved in the loop; therefore, you need a real-time operating system to guarantee response within a fixed amount of time. In addition, applications requiring extended run-times or stand-alone operation are often implemented with real-time operating systems.

16.2 Real-Time Development Tools

Traditional real-time software development requires a code editor to develop code, a compiler to generate embedded code, and a compatible debugger to validate the application, as well as tools to profile the application performance, track memory usage, and trace system thread execution. Many software development packages integrate these tools into one environment. An example of such an environment is the LabVIEW Real-Time Development Module. Figure 16.2 depicts the real-time development environment, including the host computer for software development and the real-time target for execution.

System profiling tools are important for optimizing the execution of a real-time system. The LabVIEW Execution Trace Toolkit is an example of a tracing tool that captures low-level execution details of a realtime application. Figure 16.3 shows a typical trace capture that displays when application tasks executed in time.

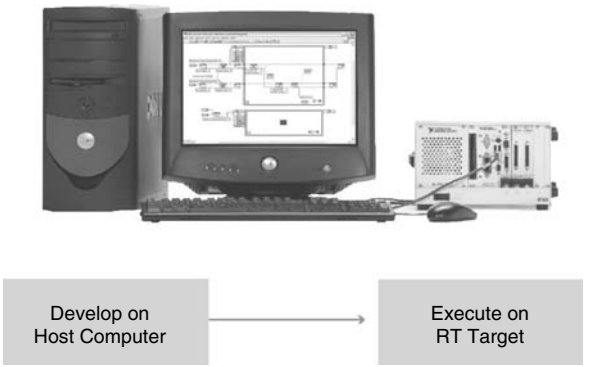


FIGURE 16.2 Real-time development process.

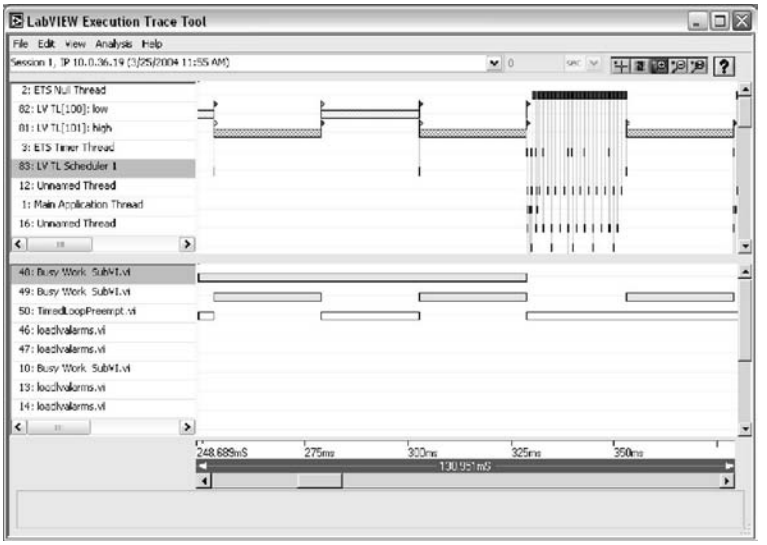


FIGURE 16.3 Application tracing tool.

Hardware deployment platforms vary depending on the application. Real-time applications use desktop PCs, modular hardware platforms such as PXI, off-the-shelf rugged controllers such as programmable logic controllers (PLCs) and programmable automation controllers (PACs), single-board computers, and embedded microprocessors on custom hardware. The popular NI CompactRIO PAC is shown in Figure 16.4. An appropriate real-time operating system is used in these hardware platforms to execute the application software.

16.2.1 Operating System Scheduler

All real-time operating systems have a scheduler that coordinates the execution of tasks (threads) by the CPU. There are various scheduling methods that are used by operating systems. Round-robin scheduling shares processor time between threads based on equal slices of time.

To illustrate round-robin scheduling, we consider the analogy of an automobile repair shop. The mechanic represents the CPU, the shop manager represents the scheduler, and multiple cars represent the multiple threads of our system. Using round-robin scheduling, the mechanic will cycle between each car for a set period of time. Round-robin scheduling guarantees that each thread will have some time with the processor.

FIGURE 16.4 NI CompactRIO programmable automation controller.

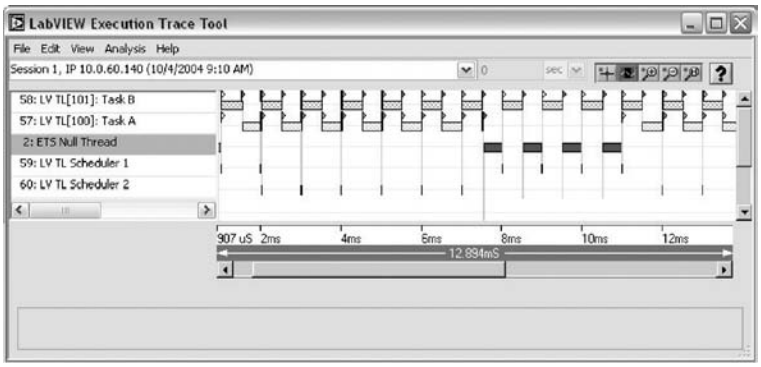


FIGURE 16.5 Example trace of preemptive scheduling.

Another method of scheduling is called preemptive scheduling, where priority can be given to the time-critical tasks.When a time-critical thread needs time from the processor, the other threads will wait until the time-critical thread is finished. In the repair shop example, an ambulance is assigned time-critical priority. As a result, it will be serviced as soon as it arrives. All other cars will be put on hold until the ambulance service is complete. Once the ambulance service is complete, the other cars will continue to share time with the mechanic. Figure 16.5 shows an example of two tasks running at different priority levels. Task B has a higher priority and runs every half millisecond, while Task A runs every 10 ms. The red flags indicate when Task A is preempted so that the higher priority Task B can be executed.

16.3 Real-Time Software Architecture

When developing a real-time application, you must decide which operations need to have real-time performance and which operations do not. The components that need real-time performance are those that need to occur within a specific period of time, meaning they are deterministic. These time-critical tasks should be assigned the highest priority in the application.

Operations that are of lower priority will run only when the time-critical operations are suspended. Examples of low priority operations are communication with a host computer, reading and writing to files, and nondeterministic communication with external instruments. These operations are sometimes called the house-keeping tasks.

16.4 Deterministic Timing

Determinism is how consistently a systemis able to perform operations within a fixed period of time. This is the most fundamental component of all real-time systems. Jitter is the variance between the expected execution time of a real-time task and the actual execution time. All real-time systems have some jitter; however, the jitter is bound and can be quantified. Systems that are not real time have very large or unbounded jitter. Figure 16.6 depicts a jitter diagram.

The most basic source of jitter is variation in the clock. However, you can induce additional jitter if you do not follow certain programming practices. To reduce jitter, tasks are programmed to run periodically on the basis of a hardware clock that can be derived from an I/O signal or a precise CPU timer. Figure 16.7 shows the configuration of a real-time task set to run every 500 ms.

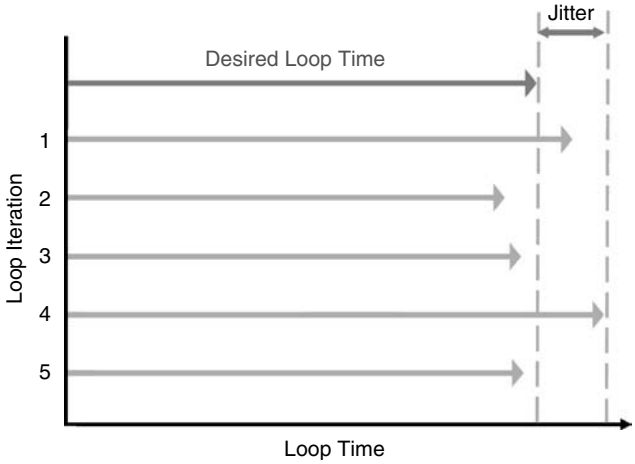


FIGURE 16.6 A jitter diagram.

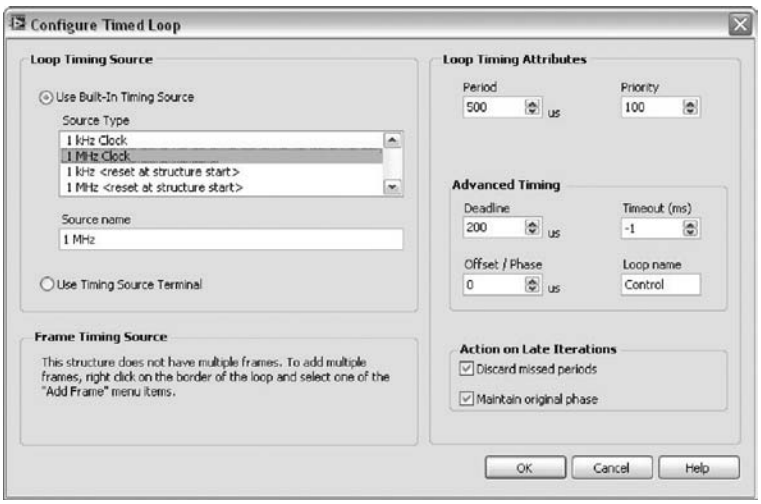


FIGURE 16.7 Real-time task configuration.

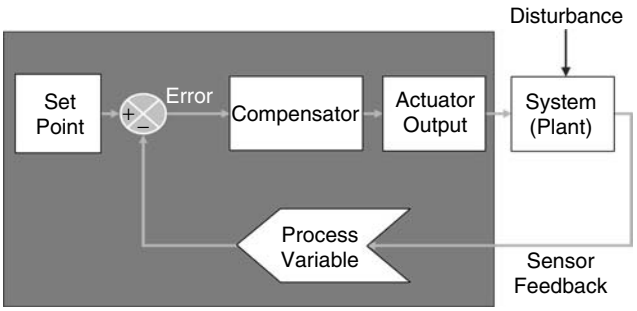


FIGURE 16.8 The block diagram of a closed-loop control system.

16.5 Implementing Real-Time Control

Control algorithm implementation is an important part of real-time system development. Control strategies can vary in complexity, from simple digital logic to advanced mathematical models that are carefully tuned to control dynamic systems. A typical block diagram of a closed-loop control system is shown in Figure 16.8.

16.5.1 Proportional-Integral-Derivative Control

The proportional-integral-derivative (PID) algorithm is the most common control algorithm used in industry. With PID, you specify a process variable and a set point. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the set point is the desired value for the parameter you are controlling.

The proportional gain determines the ratio of output response to the error. The error is the difference between the set point and the process variable. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate.

The integral component sums the error term over time. The integral response will continually increase over time unless the error is zero, so the effect is to drive the steady state error to zero.

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable.Most practical control systems use very small derivative time, because the derivative response is highly sensitive to noise in the process variable signal.

16.5.2 Fuzzy Logic Control

Fuzzy logic control is most often used for expert systems and process control. It is a method of rule-based decision making that emulates the rule of thumb thought process of humans. If an expert can qualitatively describe a control strategy, fuzzy logic can be used to develop the controller.

16.5.3 Model-Based Control Design

Another method for designing control systems is by using model-based design methods. This involves using software tools to model the behavior of dynamic systems completely in software. Then, designers can use mathematical software tools to develop control algorithms and test them with the software simulation of the dynamic system. Figure 16.9 shows a simulation diagram created in LabVIEW.

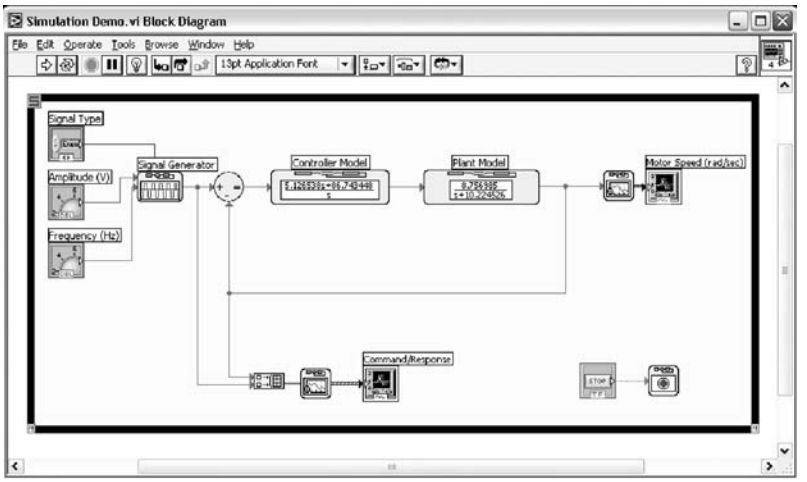


FIGURE 16.9 Simulation diagram.

16.6 Monitoring Systems

Real-time systems are usually monitored by one or more human–machine interfaces (HMI) that send and receive data from the real-time controllers. These HMIs can be in the form of powerful desktop servers, laptops, PDAs, web browser interfaces, or touch-panel displays. Communication is done through a variety of media and protocols. Ethernet is widely used as a media of data transfer with differing protocols depending on the application. These include TCP/IP for point-to-point streaming, OLE for Process Control (OPC) for distributed control systems, or web servers for remote viewing. Other methods for communicating are serial interfaces, fieldbuses such as PROFIBUS and DeviceNet, and wireless media such as WIFI and Bluetooth.

Large distributed real-time systems typically use a Supervisory Control and Data Acquisition (SCADA) system for monitoring. These systems have visualization features and can display data from real-time controllers, show alarm conditions, and log data to databases. An example of a SCADA software is the LabVIEW Datalogging and Supervisory Control (DSC) Module. A SCADA visualization of a heat exchanger is shown in Figure 16.10.

For distributed real-time control systems, data can be stored on multiple computers and monitored centrally, or it can be stored on one central server. The most difficult challenge is to communicate with live data. OPC is an industry standard interface through which software and hardware can communicate irrespective of the manufacturer. Controller hardwares from different manufacturers communicate with their own OPC server on a host system and publish I/O and calculated values as tags. SCADA systems act as OPC clients that subscribe to tags from all the individual OPC servers. Figure 16.11 shows a LabVIEW project with a list of tags that can be read from OPC servers to develop an operator interface.

For some applications, OPC data transfer may not be fast enough. In these cases, using lower-level protocols such as TCP/IP and UDP allows you to send data at faster rates.

16.7 Summary

In summary, real-time systems solve a variety of applications that require determinism and reliability. To achieve such requirements, real-time operating systems define strict scheduling behavior, which all

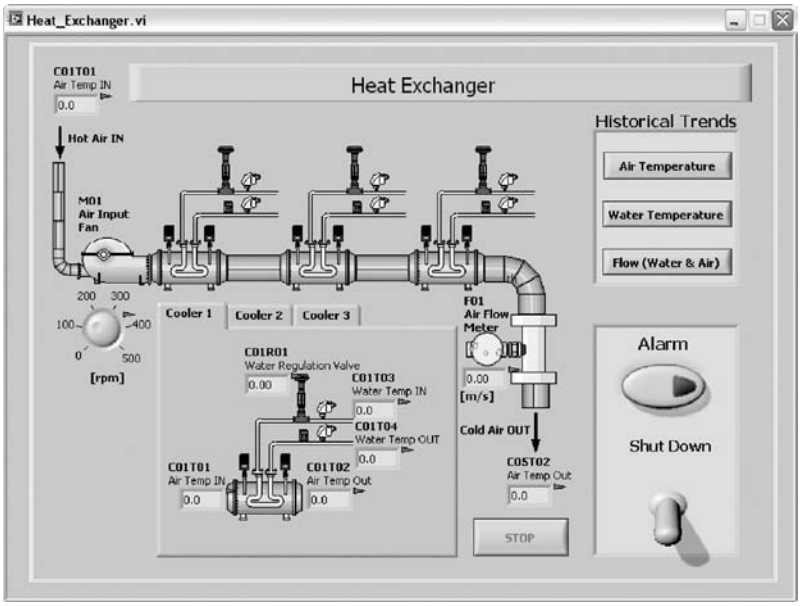


FIGURE 16.10 SCADA visualization.

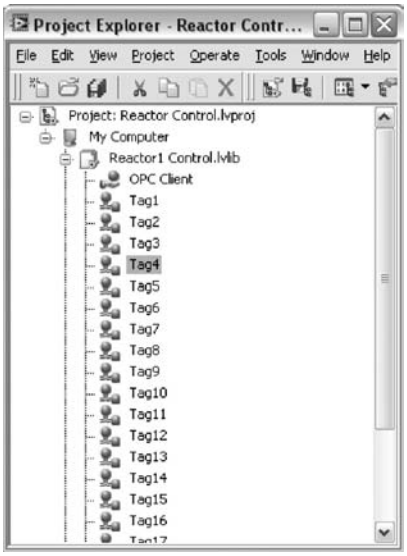


FIGURE 16.11 OPC tags.

software abides by. Due to the complex nature of real-time applications and the small margin for error, developers must employ good programming practices in order to be successful.

Further Information

Web sites

1. http://www.omimo.be/encyc/publications/faq/rtfaq.htm
2. http://www.ni.com/realtime

3. http://zone.ni.com/devzone/cda/tut/p/id/3938
4. http://zone.ni.com/devzone/cda/tut/p/id/4040

References

1. J.E. Cooling, *Software Design for Real-time Systems*, ISBN 0-412-34180-8, Chapman & Hall, London, 1991.
2. Stuart Bennett, *Real-Time Computer Control*, ISBN 0-13-764176-1, Prentice Hall, Upper Saddle River, NJ 2006.

17

Micromechatronics and Microelectromechanical Motion Devices*

Sergey Edward Lyshevski
Rochester Institute of Technology

17.1 Micromechatronic Systems Design

Micromechatronic systems integrate actuators, sensors, power electronics, and ICs. Within a focus on high-performance micromechatronic systems, books [1–3] cover the general issues in the systems design. Important topics and issues are outlined and covered in this section. One of the major components of micromechatronic systems is a high-performance microelectromechanical motion device that (1) converts physical stimuli to electrical or mechanical signals and vice versa, and, (2) performs actuation and sensing. These motion devices are controlled by ICs. Electromagnetic and electromechanical features of microelectromechanical motion devices are basics of their operation, design, analysis, and fabrication. Correspondingly, motion devices and ICs are designed taking into account possible system architectures [1–3]. The baseline step-by-step procedure in the design of micromechatronic systems is

1. Define application and environmental requirements
2. Specify performance specifications

*Some results reported in this chapter were published in [1–3].

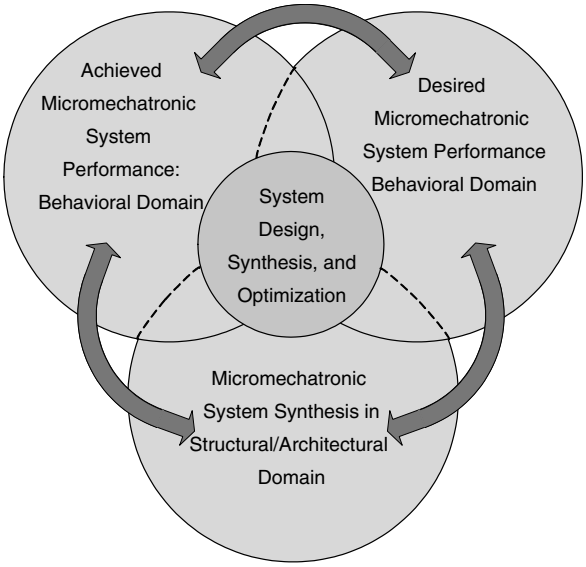


FIGURE 17.1 Design flow in synthesis of micromechatronic systems.

3. Devise (synthesize) microelectromechanical motion devices (actuators and sensors) researching operating principles, topologies, configurations, geometry, electromagnetic systems, and so forth
4. Perform electromagnetic, mechanical, and sizing-dimension estimates
5. Define technologies, techniques, processes, and materials to fabricate electromechanical motion devices
6. Design ICs to control electromechanical motion devices
7. Develop high-fidelity mathematical modelswith minimumlevel of simplifications and assumptions to examine integrated electromagnetic–mechanical phenomena and effects
8. Perform coherent electromagnetic and mechanical analysis
9. Modify and refine the design optimizing performance
10. Design control laws to control motion devices and implement these controllers using ICs
11. Integrate micromechatronic system

One of the most challenging design problems is the system architecture synthesis, system integration, optimization, and hardware (actuators, sensors, power electronics, ICs, microcontrollers, and DSPs) selection. The design starts with a given set of requirements and specifications. High-level functional design is performed first in order to produce detailed design at the subsystem and component level. Using the advanced subsystems and components, the initial design is performed, and the closed-loop electromechanical system performance is tested against the requirements. At each level of the design hierarchy, the system performance in the behavioral domain is used to evaluate and refine the design. The design flow is illustrated in Figure 17.1.

The micromechatronic system is illustrated in Figure 17.2. The systemperformance is measured against many criteria, for example, stability, robustness, transient behavior, accuracy, disturbance attenuation, and so forth. In the behavioral domain, the designer can examine and optimize the input–output transient dynamics using the tracking error $e(t) = y(t) - r(t)$, where $y(t)$ and $r(t)$ are the output and reference (command) variables. The tracking error $e(t)$ is minimized and dynamics is optimized using different performance criteria. For example,

$$J = \min_{t,e} \int_0^\infty t\,|\,e\,|\,\mathrm{d}t, \quad J = \min_{e} \int_0^\infty |\,e\,|\,\mathrm{d}t, \quad \text{or} \quad J = \min_{e} \int_0^\infty e^2 \mathrm{d}t.$$

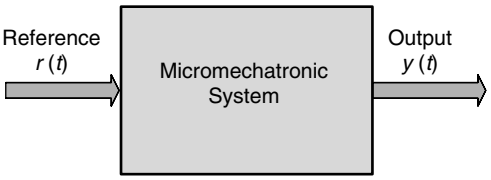


FIGURE 17.2 Dynamic micromechatronic system.

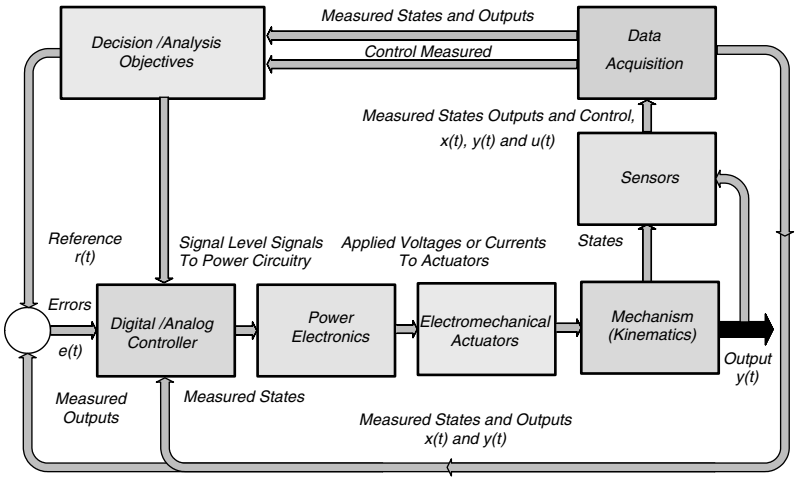


FIGURE 17.3 High-level functional block diagram of the closed-loop micromechatronic system.

Mechatronic systems (robots, electric drives, servomechanisms, pointing systems, assembles, etc.) are actuated by electromechanical motion devices that are highly nonlinear systems. Therefore, accurate actuator actuation and control are very important problems. Depending on operation and functionality of motion devices, onemust derive the expressions for the voltages applied to the phase windings in order to maximize the electromagnetic torque and ensure optimality. Actuators and sensors must be designed and integrated with the corresponding power electronics and ICs. The control laws are implemented using analog controllers, which can be integrated with power electronics (PWM amplifiers), as well as digital controllers (microcontrollers and DSPs). The principles of matching and compliance are general design principles that require that the system architectures should be synthesized integrating all subsystems and components. The matching conditions have to be determined and guaranteed, and actuators— sensors—power electronics—ICs—controller compliance must be ensured.

A functional block diagram of a controlled micromechatronic system is illustrated in Figure 17.3. Different techniques to design optimal controllers are reported in the next section. Those controllers should be derived using the baseline electromagnetic laws that define the operating principles of electromechanical motion devices. It was emphasized that depending on the motion devices, distinct power electronics must be utilized. High-performance micromechatronic systems can be designed using permanent-magnet electromechanical motion devices that ensure high torque and power densities. Radial and axial technology permanent-magnet motion devices are covered in the following sections.

17.2 Tracking Control of Micromechatronic Systems

In general, mechatronic systems are nonlinear [1–3]. Using the Hamilton–Jacobi theory and Lyapunov concepts, design of control algorithms for nonlinear systems is reported in References 1–3. To introduce

the design, we consider linear and nonlinear models. A linear model is given as

$$x^{sys}(t) = Ax^{sys} + Bu \qquad y = Hx^{sys} \qquad x^{sys}(t_0) = x_0^{sys}, \tag{17.1}$$

where $x^{sys} \in X^{sys} \subset \mathbb{R}^n$ is the state vector with auxiliary conditions; $u \in U \subset \mathbb{R}^m$ is the control vector; $y \in Y \subset \mathbb{R}^b$ is the output vector; and $A^{sys} \in \mathbb{R}^{n\times n}$ and $B^{sys} \in \mathbb{R}^{n\times m}$ are the constant-coefficient matrices. Define the tracking error vector as

$$e(t) = Nr(t) - y(t) = Nr(t) - Hx^{sys}(t). \tag{17.2}$$

Therefore, from Equations 17.1 and 17.2, one finds

$$\dot{e}(t) = N\dot{r}(t) - \dot{y}(t) = N\dot{r}(t) - H\dot{x}^{sys}(t) = N\dot{r}(t) - HA^{sys}x^{sys} - HB^{sys}u. \tag{17.3}$$

Using the expanded state vector $x(t) = \begin{bmatrix} x^{sys}(t) \\ e(t) \end{bmatrix}$, we have

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} \dot{x}^{sys}(t) \\ \dot{e}(t) \end{bmatrix} = \begin{bmatrix} A^{sys} & 0 \\ -HA^{sys} & 0 \end{bmatrix}\begin{bmatrix} x^{sys} \\ e \end{bmatrix} + \begin{bmatrix} B^{sys} \\ -HB^{sys} \end{bmatrix} u + \begin{bmatrix} 0 \\ N \end{bmatrix}\dot{r} \\ &= Ax + Bu + \begin{bmatrix} 0 \\ N \end{bmatrix}\dot{r} \quad y = Hx^{sys}. \end{aligned} \tag{17.4}$$

The *space transformation* method is based on the use of the following z and v vectors: $z = \begin{bmatrix} x \\ u \end{bmatrix}$ and $v = \dot{u}$. Thus, from Equation 17.4, we obtain the system

$$\dot{z}(t) = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} z + \begin{bmatrix} 0 \\ I \end{bmatrix} v = A_z z + B_z v, \quad y = Hx^{sys} \qquad z(t_0) = z_0. \tag{17.5}$$

Minimizing the quadratic performance functional

$$J = \int_{t_0}^{t_f} \left(z^T Q_z z + v^T G_z v \right) dt, \; Q_z \in \mathbb{R}^{(n+m)\times(n+m)}, \; Q_z \geq 0, \; G \in \mathbb{R}^{m\times m} \quad G > 0, \tag{17.6}$$

with respect to the system dynamics (Equation 17.5), the application of the first-order necessary condition for optimality gives

$$v = -G_z^{-1} B_z^T K z. \tag{17.7}$$

The Riccati equation

$$-\dot{K} = KA_z + A_z^T K - KB_z G_z^{-1} B_z^T K + Q_z, \; K(t_f) = K_f \tag{17.8}$$

is solved to find the unknown matrix $K \in \mathbb{R}^{(n+m)\times(n+m)}$. Taking note of Equation 17.7, one has

$$\begin{aligned} \dot{u}(t) &= -G_z^{-1} B_z^T K z = -G_z^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}^T \begin{bmatrix} K_{11} & K_{21}^T \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \\ &= -G_z^{-1} K_{21} x - G_z^{-1} K_{22} u = K_{f1} x + K_{f2} u. \end{aligned} \tag{17.9}$$

Using $\dot{x}(t) = Ax + Bu$, wehave $u = B^{-1}(\dot{x}(t) - Ax) = (\mathrm{BTB})^{-1}\,\mathrm{BT}\,(\dot{x}(t) - Ax)$. From Equation 17.9, one obtains

$$\begin{aligned}\dot{u}(t) &= K_{f1}x + K_{f2}u = K_{f1}x + K_{f2}\left(B^T B\right)^{-1} B^T(\dot{x}(t) - Ax)\\ &= \left[K_{f1} - K_{f2}\left(B^T B\right)^{-1} B^T A\right]x(t) + K_{f2}\left(B^T B\right)^{-1} B^T \dot{x}(t)\\ &= \left(K_{f1} - K_{F1}A\right)x(t) + K_{F1}\dot{x}(t) = K_{F2}x(t) + K_{F1}\dot{x}(t).\end{aligned} \tag{17.10}$$

Using Equation 17.10, and taking note of $x(t) = \begin{bmatrix} x^{\mathrm{sys}}(t) \\ e(t) \end{bmatrix}$, one finally derives the proportional-integral controller in the following form

$$\begin{aligned}u(t) &= K_{F1}x(t) - K_{F1}x_0 + \int K_{F2}x(\tau)d\tau + u_0\\ &= K_{F1}\begin{bmatrix} x^{\mathrm{sys}}(t) \\ e(t) \end{bmatrix} - K_{F1}\begin{bmatrix} x_0^{\mathrm{sys}}(t) \\ e_0(t) \end{bmatrix} + \int K_{F2}\begin{bmatrix} x^{\mathrm{sys}}(\tau) \\ e(\tau) \end{bmatrix} d\tau + u_0.\end{aligned} \tag{17.11}$$

For nonlinear micromechatronic systems, as given by

$$\dot{x}^{\mathrm{sys}}(t) = F(x^{\mathrm{sys}}) + B(x^{\mathrm{sys}})u, \qquad y = Hx^{\mathrm{sys}}, \qquad x^{\mathrm{sys}}(t_0) = x_0^{\mathrm{sys}}, \tag{17.12}$$

the proposed procedure can be straightforwardly used. In Equation 17.12), $F(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{n}}$ and $B(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{n\times m}}$ are the smooth Lipschitz maps; $H \in \mathbb{R}^{\mathrm{b\times c}}$ is the output matrix with constant coefficients.

The proportional-integral controller for Equation 17.12 is given as

$$\dot{u}(t) = -G_z^{-1}B_z^T\frac{\partial V}{\partial z} = -G_z^{-1}\begin{bmatrix} 0 \\ I \end{bmatrix}^T \frac{\partial V(x,u)}{\partial [x\ u]^T}, \tag{17.13}$$

where $V(x, u)$ is the return function.

The control law is bounded, and $u_{\min} \le u \ge u_{\max}$. To design the *admissible* control laws, we minimize

$$J = \int_{t_0}^{t_f}\left[z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v\right]\mathrm{d}t, \tag{17.14}$$

where $\Phi(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}}$ is the bounded, integrable, one-to-one, real-analytic, globally Lipschitz continuous function, $\Phi \in U \subset \mathbb{R}^{\mathrm{m}}$.

Minimizing Equation 17.14 with $u_{\min} \le u \ge u_{\max}$, $u \nmid U$, for linear and nonlinear systems, we have

$$-\frac{\partial V}{\partial t} = \min_{u\in U}\left\{z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v + \frac{\partial V}{\partial z}^T (A_z z + B_z v)\right\}, \tag{17.15}$$

$$\text{and } -\frac{\partial V}{\partial t} = \min_{u\in U}\left\{z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v + \frac{\partial V}{\partial z}^T [F_z(z) + B_z(z)v]\right\} \tag{17.16}$$

Using the first-order necessary condition for optimality, the minimization of Equations 17.15 and 17.16 results in the *admissible* controllers

$$\dot{u}(t) = -\Phi\left(G_z^{-1}B_z^T\frac{\partial V}{\partial z}\right) \quad u \in U. \tag{17.17}$$

Assuming that the solution of the Hamilton-Jacobi equation is approximated by the quadratic return function, from Equation 17.17, one obtains

$$u(t) = \Phi\left(K_{F1}x(t) + \int K_{F2}x(\tau)d\tau\right) = \Phi\left(K_{F1}\begin{bmatrix} x^{\text{sys}}(t) \\ e(t) \end{bmatrix} + \int K_{F2}\begin{bmatrix} x^{\text{sys}}(\tau) \\ e(\tau) \end{bmatrix} d\tau\right) \quad u \in U. \tag{17.18}$$

The *admissible* control law synthesized is bounded, and the second-order necessary condition for optimality is satisfied. However, the sufficient conditions must be examined. Using the *admissibility* concept, it is demonstrated in References 1–3 that a system with bounded control (Equation 17.18) is stable in $X(X_0, U)$, and robust tracking is guaranteed in the convex and compact set $E(E_0, Y, R)$ if for the reference input $r \in R$ there exists a positive-definite function $V(e,x)$ such that $[\mathrm{d}V(e, x)]/\mathrm{d}t \leq 0$. If the criteria imposed on the Lyapunov pair, $V(e, x) > 0$ and $[\mathrm{d}V(e, x)]/\mathrm{d}t \leq 0$, are guaranteed for all $x_0 \in X_0$, $e_0 \in E_0$, $u \in U$, and $r \in R$, then, the closed-loop system is robustly stable in $X(X_0, U)$, and robust tracking is guaranteed in the convex and compact set $E(E_0, Y, R)$. By computing the derivative of the $V(e, x)$, the unknown feedback gains can be found to satisfy the sufficient conditions for stability.

The designed analog controllers may be implemented using microcontrollers. Therefore, analog controllers are discretized. The feedback gains k_{dp} and k_{di} of the digital control law are found using the proportional, integral, and derivative coefficients of the analog PID controller (k_p and k_i) as well as the sampling period T_s. We have $k_{dp} = k_{\text{p}} \cdot \frac{1}{2}\, k_{\text{di}}$ and $k_{\text{di}} = T_s k_{\text{i}}$. In particular, the controller can be implemented as $u\,(kT_{\text{s}}) = k_{\text{p}} e\,(kT_{\text{s}}) + \frac{1}{2}\, k_{\text{i}} T_{\text{s}} \sum_{i=1}^{k} [e\,((i-1)T_{\text{s}}) + e\,(iT_{\text{s}})]$. The proportional-integral controllers with the state feedbacks are implemented in the similar way, and the state variables are sampled at the specified sampling period T_s.

17.3 Synthesis of Microelectromechanical Motion Devices

The designer synthesizes microelectromechanical motion devices by devising and examining baseline physics and operational principles. Axial and radial topology, ac and dc, electromagnetic and electrostatic, and other actuators are covered in [1–3]. To introduce the synthesis taxonomy, we consider radial topology two-phase permanent-magnet synchronous motion devices as shown in Figure 17.4. The electromagnetic system is *endless*, and different geometries can be utilized. In contrast, the translational (linear) synchronous machines have the *open-ended* electromagnetic system. Thus, the device geometry and electromagnetic systems can be integrated within the synthesis, classification, analysis, design, and optimization taxonomy. In particular, motion devices can have different geometries (plate, spherical, torroidal, conical, cylindrical, and asymmetrical) and electromagnetic systems. Using these distinct features, one can classify motion devices.

The basic types of electromagnetic motion devices are induction, synchronous, while topologies are rotational and translational (linear). That is, devices can be classified using a type classifier as given by $Y = \{y : y \in Y\}$. As illustrated in Table 17.1, electromechanical motion devices are categorized using a geometric classifier (plate *P*, spherical *S*, torroidal *T*, conical *N*, cylindrical *C*, or asymmetrical *A* geometry), and an electromagnetic system classifier (*endless E*, *open-ended O*, or *integrated I)*. The classifier, as documented in Table 17.1, is partitioned into 3 horizontal and 6 vertical strips, and contains 18 sections, each identified by ordered pairs of characters, such as *(E, P)* or *(O, C)*. In each ordered pair, the first entry is a letter chosen from the bounded electromagnetic system set $M = \{E, O, I\}$. The second entry is a letter chosen from the geometric set $G = \{P, S, T, N, C, A\}$. That is, the electromagnetic system—geometric set is

$$M \times G = \{(E, F), (E, S), (E, T), \cdots, (I, N), (I, C), (I, A)\}.$$

In general, we have $M \times G = \{(m, g) : m \in M \text{ and } g \in G\}$.

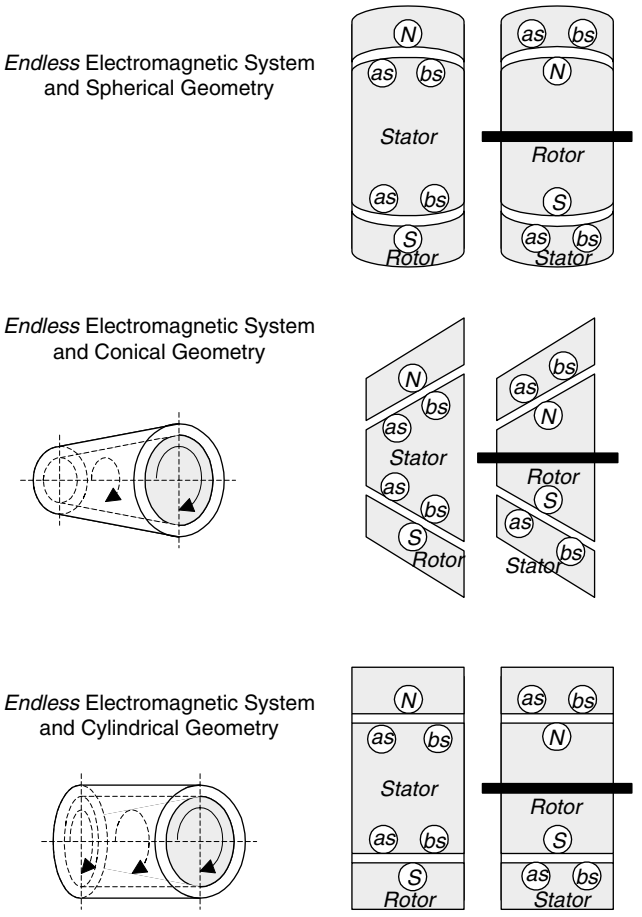


FIGURE 17.4 Radial topology permanent-magnet synchronous motion devices with *endless* electromagnetic system and different geometries.

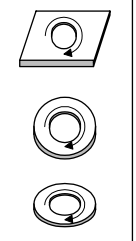
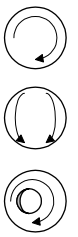
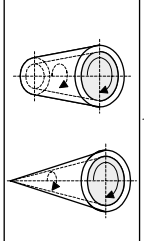
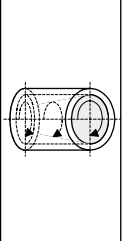
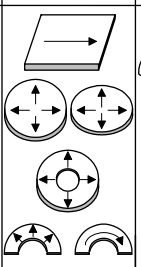
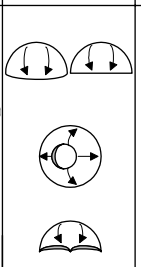
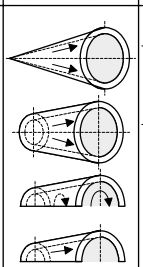
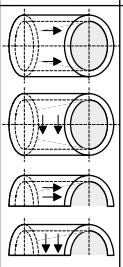
Other categorization can be applied. For example, multiphase motion devices are classified using a phase classifier $H = \{h : h \in H\}$. Therefore, we have

$$Y \times M \times G \times H = \{(y,\ m,\ g,\ h) : y \in Y,\ m \in M,\ g, \in G \text{ and } h \in H\}$$

Topology (radial, axial, or integrated), permanent magnet shapes (strip, arc, disk, rectangular, rhomb, triangular, etc.), permanent magnet characteristics (*BH*demagnetization curve, energy product, hysteresis minor loop, etc.), *electromotive force* distribution, cooling, power, torque, size, torque-speed characteristics, bearing, packaging, as well as other distinct features can be classified. Hence, microelectromechanical motion devices can be devised and classified by an *N*-tuple as: motion device type, electromagnetic system, geometry, topology, phase, winding, sizing, bearing, cooling, fabrication, materials, packaging, and so forth.

By using distinct geometry, electromagnetic systems (*endless*, *open ended*, and *integrated*), topologies, and other features, novel high-performance motion devices can be synthesized by utilizing the Synthesis and Classification Solver. For example, the spherical, conical, and cylindrical geometry of two-phase permanent-magnet synchronous motion devices are illustrated in Figure 17.5. The cross-section of the slotless radial-topology synchronous microactuator, fabricated on the silicon substrate with polysilicon stator (with deposited windings), polysilicon rotor (with deposited permanent magnets), and contact

TABLE 17.1 Classification of Microelectromechanical Motion Devices Using the Electromagnetic System—Geometry Using Synthesis and Classification Solver

M \ G	Plate, P	Spherical, S	Torroidal, T	Conical, N	Cylindrical, C	Asymmetrical,A
Electromagnetic System: *Endless* (Closed), E						
Electromagnetic System: *Open-Ended* (Open), O						
Electromagnetic System: *Integrated*, I						

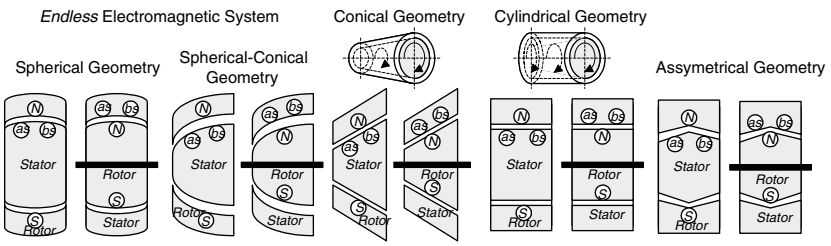


FIGURE 17.5 Two-phase permanent-magnet synchronous motion devices with *endless* electromagnetic system and distinct geometries.

bearing is illustrated in Figure 17.6. The fabrication of this microactuator and the processes are reported in References 1–3.

The major task is to devise high-performance motion devices relaxing fabrication difficulties guarantying affordability. The electrostatic and planar motion devices fabricated and tested to date are found to be inadequate for a wide range of applications owing to specifications imposed on performance [1–3]. Figure 17.7 illustrates the axial topology motion device. The stator is made on the substrate with deposited windings (surface micromachined or printed coils can be made using the fabrication processes as well as using double-sided substrate, applying conventional photolithography and etching processes [1–3]). The bearing post is fabricated on the stator substrate and the bearing hold is a part of the rotor microstructure. The rotor with permanent-magnet thin films rotates because of the electromagnetic torque developed. It is important to emphasize that stator and rotor are made using the conventional well-developed processes and materials.

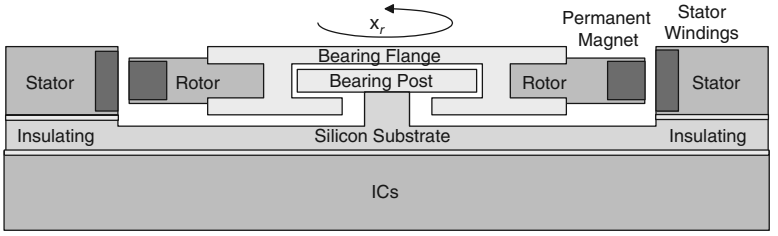


FIGURE 17.6 Cross-section schematics of a slotless radial-topology permanent-magnet brushless microactuator with controlling ICs.

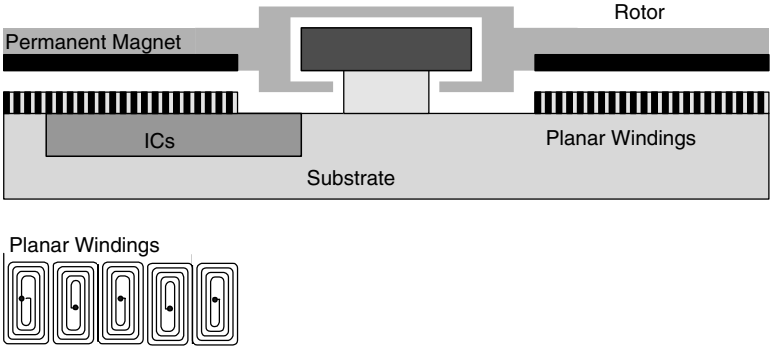


FIGURE 17.7 Cross-section of an axial topology motion device with controlling ICs.

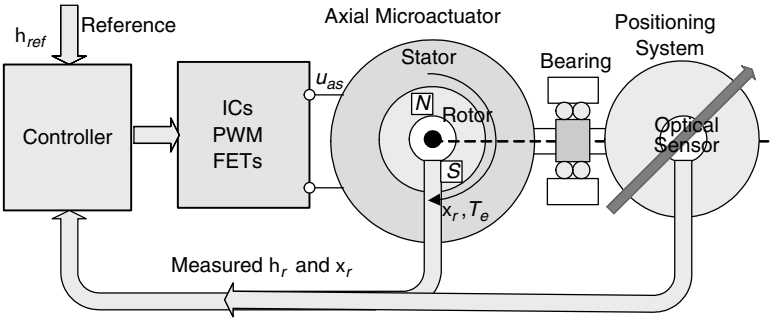


FIGURE 17.8 Schematic diagram of a servosystem: controller; ICs; actuator/sensors; and mechanism.

17.4 Micromechatronic System with an Axial Topology Motion Device

High-torque-density permanent-magnet axial-topology actuators with high switching-frequency power electronics and ICs, controlled by controllers (microcontroller or analog controllers), are found to be a very promising solution. High performance is achieved by utilizing: (1) high-torque-density brushless motion devices; (2) designing tracking controllers implemented utilizing controllers that integrate filters, IO devices, and advanced converters; (3) high-performance two- and four-quadrant power stages. The control laws are designed to satisfy the specifications imposed. Consider a servosystem as depicted in Figure 17.8. The reference signal is the angular displacement. Using the reference θ_{ref} and actual θ_t angular displacements (measured by the high-accuracy optical sensor), the controller develops PWM signals to

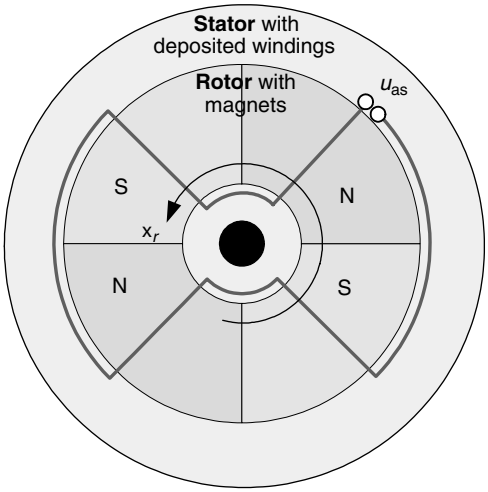


FIGURE 17.9 Axial topology actuator.

drive high-frequency field effect transistors in the power stage. The output voltage u_{as} is controlled by changing the duty cycle.

A limited-angle axial-topology permanent-magnet microactuator is documented in Figure 17.9. This device is used in the studied hard drive reported in Figure 17.8. To rotate the disk and displace the pointer, we utilize axial topology direct-drive synchronous actuators with the segmented array of the permanent-magnet strips. These high-torque-density permanent-magnet actuators exceed performance characteristics achieved by other servomotors [1–3]. The mathematical model must be found and used in the design of controllers as well as numerical analysis. The magnetic field developed by the magnets is approximated as continuous or discontinuous functions, for example, $B(\theta_r) = B_{max}\,\mathrm{sgn}(\sin(\frac{1}{2}\,\mathrm{N}_m\theta_r))$, where B_{max} is the magnetic flux density from the permanent magnets as viewed at the coils; Nm is the number of magnets. For the limited-angle actuator, depending on the geometry, magnetization, and separations of magnets, one can approximate $B(\theta_r)$ as $B(\theta_r) = B_{max}\tanh^q(k\theta_r)$, where q is the technologydependent integer, and usually one finds $q = 1$ or $q = 3$; k is the technology-dependent coefficient.

The electromagnetic torque is derived using the magnetic dipole moment **m**, for example, $T_e = \mathbf{m} \times \mathbf{B}$. That is, for the symmetric two-filament limited angle actuator, for $q = 1$ we obtain $T_e = 2NlB_{max}\,|\tanh(k\theta_r)|\,i_{as}$, where N is the number turns in filaments; l is the equivalent coil active length; i_{as} is the current in the coil.

The mathematical model is found by using Kirchhoff's and Newton's second laws. In particular, the following circuitry and *torsional–mechanical* equations are used.

$$u_{as} = r_s i_{as} + \frac{d\psi_{as}}{dt},$$

$$T_e - B_m\omega_r - T_L = J\frac{d^2\theta_r}{dt^2},$$

where u_{as} is the applied voltage; r_s is coil resistance; ψ_{as} is the flux linkages as viewed from the winding; B_m is the friction coefficient; T_L is the load torque; J is the equivalent moment of inertia.

The flux linkage is expressed as

$$\psi_{as} = L_{as}i_{as} + A_eB(\theta_r),$$

where L_{as} is the self-inductance, and A_e is the equivalent cross-sectional area.

A set of nonlinear differential equations that models an axial topology microactuator is

$$\frac{\mathrm{d}i_{\mathrm{as}}}{\mathrm{d}t} = \frac{1}{L_{\mathrm{as}}}\left(-r_{\mathrm{s}}i_{\mathrm{as}} - \underbrace{A_{\mathrm{e}}\frac{\mathrm{d}B(\theta_{\mathrm{r}})}{\mathrm{d}t}}_{\text{back emf}} + u_{\mathrm{as}}\right) = \frac{1}{L_{\mathrm{as}}}\left(-r_{\mathrm{s}}i_{\mathrm{as}} - A_{\mathrm{e}}B_{\max}k\,\mathrm{sech}^2(k\theta_{\mathrm{r}})\omega_{\mathrm{r}} + u_{\mathrm{as}}\right),$$

$$\frac{\mathrm{d}\omega_{\mathrm{r}}}{\mathrm{d}t} = \frac{1}{J}\left[2NlB_{\max}\left|\tanh(k\theta_{\mathrm{r}})\right| i_{\mathrm{as}} - B_{\mathrm{m}}\omega_{\mathrm{r}} - T_{\mathrm{L}}\right],$$

$$\frac{\mathrm{d}\theta_{\mathrm{r}}}{\mathrm{d}t} = \omega_{\mathrm{r}}.$$

The bounded proportional-integral tracking controller with state feedbacks (Equation 17.18) is designed using the *state transformation* method. We use the expanded state vector as $z = \begin{bmatrix} x \\ u \end{bmatrix}$, $x(t) = \begin{bmatrix} x^{sys}(t) \\ e(t) \end{bmatrix}$, where $x^{\mathrm{sys}}(t) = \begin{bmatrix} i_{\mathrm{as}}(t) \\ \omega_r(t) \\ \theta_r(t) \end{bmatrix}$. Using different weighting matrices Q_z and G_z in Equation 17.14, distinct feedback gains are obtained. Letting $Q_z = I$ and $G_z = 1$, for the bounded controller (Equation 17.18) synthesized we have $k_p = 31$ and $k_i = 24$. The output dynamics for $\theta_{\mathrm{ref}} = 0.349$ rad is shown in Figure 17.10 for $T_s = 0.0014$ s. For $k_p = 31$ and $k_i = 24$ with the state feedbacks, the settling time is 0.027 s, and the overshoot is 27%. It is possible to minimize the overshoot to 13%. The application of proportional-integral controllersmay not ensure the desired transient behavior as reported in Figure 17.10, when $k_p = 20$ and $k_i = 19$. We tested the servo without a return spring that is commonly used to ensure mechanical damping. Thus, the most challenging problem is considered. Pointer acceleration and deceleration are achieved by properly controlling the actuator that operates at very high efficiency and develops high electromagnetic torque.We conclude that optimal performance is achieved.

17.5 Synchronous Micromachines

In micromechatronics systems, rotational and translational motion devices (actuators and sensors), controlled by ICs, are widely used. Among various devices, synchronous permanent-magnet motion devices exhibit superior performance. The advantages of axial topology motion devices are feasibility, efficiency, and reliability. Fabrication simplicity results because: (1) magnets are flat; (2) there are no strict shape requirements imposed on magnets; (3) rotor back ferromagnetic material is not required; (4) it is easy to deposit planar windings on the flat stator. An axial topology permanent-magnet two-phase synchronous motion device is documented in Figure 17.11.

For two-phase actuator, one supplies two phase voltages u_{as} and u_{bs} in order to develop an electromagnetic torque. Assuming that the magnetic flux is constant through the magnetic plane (current loop), the torque on a planar current loop of any size and shape in the uniformmagnetic field is $\mathbf{T} = \boldsymbol{i}\mathbf{s} \times \mathbf{B} = \mathbf{m} \times \mathrm{B}$, where i is the current in the loop (winding); $\mathbf{m}$ is the magnetic dipole moment. The electromagnetic force is given as $\mathrm{F} = \oint id\mathbf{l} \times \mathbf{B}$. The interaction of the current (in windings) and magnets will produce the rotating electromagnetic torque. Microscale synchronous transducers can be used as motors (actuators) and generators (sensors). Microgenerators (velocity and position sensors) convert mechanical energy into electrical energy, while micromotors (actuators) convert electrical energy into mechanical energy. A broad spectrum of synchronous microtransducers can be used as actuators, for example, microscale drives, servos, and power systems.

Distinct electrostatic and electromagnetic motion devices can be designed and fabricated using the surface micromachining technology. However, all high-performance, high-power, and torque density devices are electromagnetic, and these devices use permanent magnets. Therefore, the issues of fabrication and analysis of magnets are of great interest. Different soft and hard magnets can be fabricated using surface micromachining. In particular, Ni, Fe, Co, NiFe, and other magnets can be made. In general, four types of

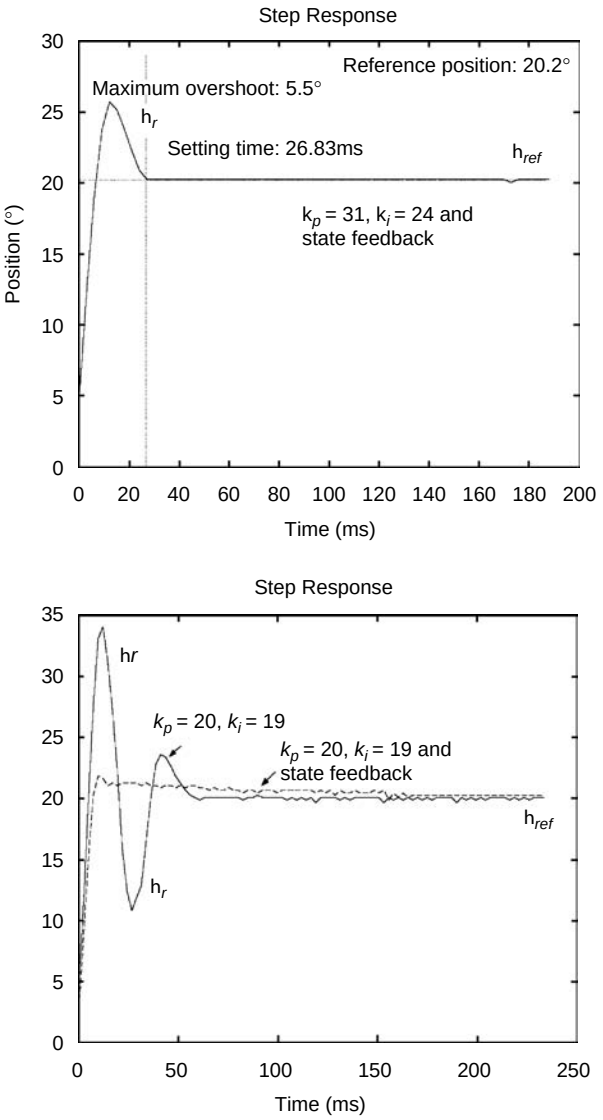


FIGURE 17.10 Transient dynamics of the closed-loop servo with the bounded proportional-integral controller with state feedback.

high-performance magnets have been commonly used in electromechanical motion devices, for example, neodymium iron boron ($Nd_2Fe_{14}B$), samarium cobalt (usually Sm_1Co_5 and Sm_2Co_{17}), ceramic (ferrite), and alnico (AlNiCo). The term soft is used to refer to magnets that have high saturation magnetization and a low coercivity (narrow *BH* curve). Another property of these magnets is their low magnetostriction. The soft magnets have been widely used in magnetic recording heads, and, in particular, NiFe thin films with the desired properties have been utilized. Hard magnets have wide *BH* curves (high coercivity) and, therefore, high-energy storage capacity. These magnets should be used in devices in order to attain high force, torque, and power densities. Unfortunately, limited progress has been made in the fabrication of hard permanent magnets.

The energy density is given as the area enclosed by the *BH* curve, for example, $w_m = \frac{1}{2}\mathbf{B} \cdot \mathbf{H}$. When magnets are used in motion devices, the demagnetization curve (second quadrant of the BH curve) is of

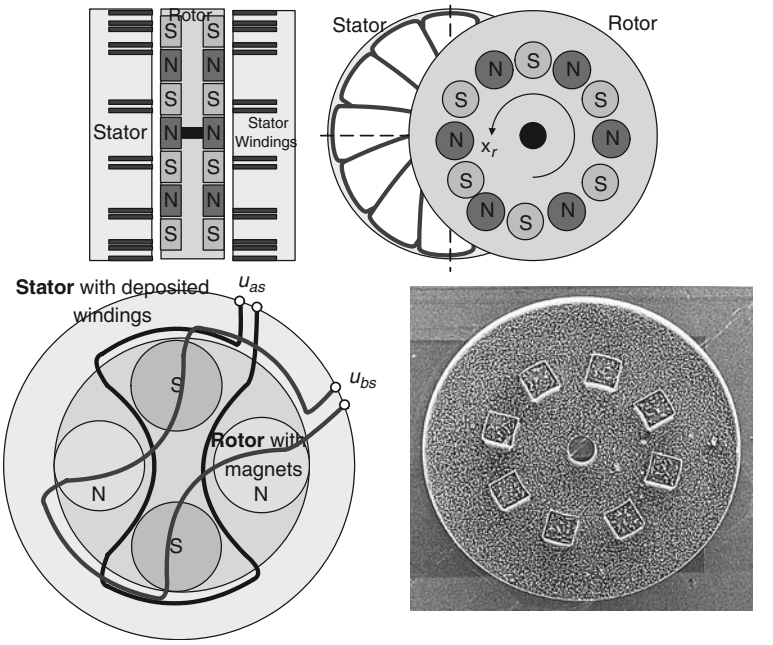


FIGURE 17.11 Axial permanent-magnet synchronous motion device.

interest. Permanent magnets store, exchange, and convert energy. In particular, permanent magnets produce stationary magnetic field without external energy sources or radiation devices.We apply Kirchhoff's and Newton's laws to derive the equations of motion for axial topology permanent-magnet synchronous motion devices. Using an axial permanent-magnet synchronous device documented in Figure 17.11, we assume that this variation of the flux density is sinusoidal, for example, $B(\theta_r) = B\max \sin^n(\frac{1}{2} N\mathrm{m}\theta_r)$, $n = 1$, 3, 5, · · ·, where B_{max} is the maximum flux density in the airgap produced by the magnet as viewed from the winding; n is the integer that is a function of the magnet geometry and the waveform of the airgap B. It should be emphasized that B_{max} depends on the magnets used, airgap length, temperature, and so forth.

The electromagnetic torque, developed by a single-phase axial topology permanent-magnet synchronous actuator, is found using the expression for the coenergy $W_c(i_{as}, \theta_r)$ that is given as $W_c(i_{as}, \theta_r) = NA_eB(\theta_r)i_{as}$. Assuming that the airgap flux density obeys $B(\theta_r) = B_{max} \sin(\frac{1}{2} N_m\theta_r)$, we have $W_c(i_{as}, \theta_r) = NA_eB_{max} \sin(\frac{1}{2} N_m\theta_r)i_{as}$, and the electromagnetic torque is

$$T_e = \frac{\partial W_c(i_{as}, \theta_r)}{\partial \theta_r} = \frac{\partial \left(NA_eB_{max} \sin(\frac{1}{2}N_m\theta_r)i_{as}\right)}{\partial \theta_r} = \frac{1}{2}N_mNA_eB_{max} \cos\left(\frac{1}{2}N_m\theta_r\right) i_{as}.$$

As we fed the following currentwith the magnitude i_M to the micromotorwinding $i_{as} = i_M \cos(\frac{1}{2} N_m\theta_r)$, the electromagnetic torque is $T_e = \frac{1}{2} N_mNA_eB_{max}i_M \cos^2(\frac{1}{2} N_m\theta_r) \neq 0$. The mathematical model of the single-phase permanent-magnet micromotor is found by using Kirchhoff's and Newton's second laws. In particular, one obtains

$$u_{as} = r_s i_{as} + \frac{d\psi_{as}}{dt} \text{ (circuitry equation)},$$

$$T_e - B_m\omega_r - T_L = J\frac{d^2\theta_r}{dt^2} \text{ (torsional–mechanical equation)}.$$

Taking note of the flux linkage equation $\psi_{as} = L_{as} i_{as} + NA_e B(\theta_r)$, we have a set of three first-order nonlinear differential equations that models a single-phase axial topology permanent-magnet synchronous motion device

$$\frac{di_{as}}{dt} = \frac{1}{L_{as}}\left(-r_s i_{as} - \frac{1}{2}N_m N A_e B_{max}\cos\left(\frac{1}{2}N_m\theta_r\right)\omega_r + u_{as}\right),$$
$$\frac{d\omega_r}{dt} = \frac{1}{J}\left(\frac{1}{2}N_m N A_e B_{max}\cos\left(\frac{1}{2}N_m\theta_r\right) i_{as} - B_m\omega_r - T_L\right),$$
$$\frac{d\theta_r}{dt} = \omega_r.$$

Single-phase axial and radial topologies permanent-magnet synchronous actuators have a torque ripple. The torque ripple can be minimized feeding $i_{as} = i_M\left[\cos(\frac{1}{2}N_m\theta_r) + \varepsilon\right]^{-1}$, $|i_{as}| \le i_M$, where $\varepsilon > 0$. To overcome this deficiency (torque ripple and velocity chattering) and increase torque density, two-phase motion devices are used. For two-phase actuators, the coenergy is given as $W_c(i_{as}, \theta_r) = NA_{ag}B_{max}\left(\sin(\frac{1}{2}N_m\theta_r)i_{as} - \cos(\frac{1}{2}N_m\theta_r)i_{bs}\right)$, and, hence, the electromagnetic torque is $T_e = \frac{\partial W_c(i_{as},\theta_r)}{\partial\theta_r} = \frac{1}{2}N_m N A_e B_{max}\left(\cos(\frac{1}{2}N_m\theta_r)i_{as} + \sin(\frac{1}{2}N_m\theta_r)i_{bs}\right)$. Thus, feeding the phase currents i_{as} and i_{bs} as $i_{as} = i_M\cos(\frac{1}{2}N_m\theta_r)$ and $i_{bs} = i_M\sin(\frac{1}{2}N_m\theta_r)$, we maximize the electromagnetic torque because $T_e = \frac{1}{2}N_m N A_e B_{max} i_M$.

17.6 Fabrication Aspects

Mini and microscale electromechanical motion devices can be fabricated by creating (etching or depositing) of conductors (coils and windings), ferromagnetic core, magnets, insulating layers, as well as other microstructures (radiating energy devices, movable and stationarymembers, and their components including bearings and posts). The subsequent order of the processes, sequential steps, and materials are different depending on the devised, designed, analyzed, and optimized devices. Several textbooks [1–6] provide the reader with the basic fabrication features and processes. In partcilular, the electrostatic devices are covered in References 1–6, while the electromagnetic motion devices are reported in References 1–3. Complementary metal oxide semiconductor (CMOS), high aspect ratio (LIGA and LIGA-like), and surface micromachining technologies are key features for fabrication of microdevices and structures. The LIGA (Lithography–Galvanoforming–Molding, or in German Lithografie–Galvanik–Abformung) technology allows one to make three-dimensional microstructures with the aspect ratio (depth versus lateral dimension is more than 100). The LIGA technology is based on the x-ray lithography that ensures short wavelength (from few to 10 Å), which leads to negligible diffraction effects and larger depth of focus compared with photolithography. The major processes in the machine's microfabrication are diffusion, deposition, patterning, lithography, etching, metallization, planarization, assembling, and packaging. Thin film fabrication processes were developed and used for polysilicon, silicon dioxide, silicon nitride, and other different materials (e.g., metals, alloys, composites, etc.). The basic steps are lithography, deposition of thin films and materials (electroplating, chemical vapor deposition, plasma-enhanced chemical vapor deposition, evaporation, sputtering, spraying, screen printing, etc.), removal of material (patterning) by wet or dry techniques, etching (plasma etching, reactive ion etching, laser etching, etc.), doping, bonding (fusion, anodic, and other), and planarization.

To fabricate motion and radiating energy microscale structures and devices, different fabrication technologies are used [1–6]. New processes were developed and novel materials were applied modifying the CMOS, surface micromachining, and LIGA technologies. Currently, the state-of-the-art technology in microfabrication has been progressed to the nanometer minimal features. Motion devices and structures (stator, rotor, bearing, coils, etc.) are defined photographically, and the high-resolution photolithography is applied to define two- (planar) and three-dimensional shapes (geometry). Deep ultraviolet lithography processes were developed to decrease the feature sizes of microstructures to 0.1 $f\hat{E}$m and less. Different

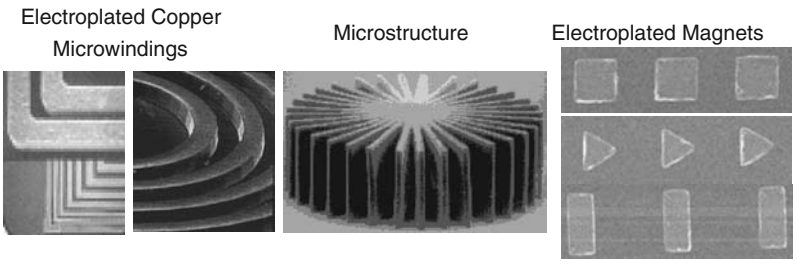


FIGURE 17.12 Deposited copper microwindings, microstructure, and permanent magnets.

exposure wavelengths λ (435, 365, 248, 193 nm) are used. Using the Rayleigh model for image resolution, the expressions for image resolution i_R and the depth of focus d_F are $i_R = k_i \frac{\lambda}{N_A}$ and $d_F = k_d \frac{\lambda}{N_A^2}$, where k_i and k_d are the lithographic process constants; λ is the exposure wavelength; N_A is the numerical aperture coefficient, and for high-numerical aperture we have $N_A = 0.5$–0.6.

The g- and i-line IBM lithography processes (with wavelengths of 435 and 365 nm, respectively) allow one to attain minimal features in the range of 0.35 μm. The deep ultraviolet light sources (mercury source or excimer lasers) with 248-nm wavelength enable the industry to achieve 0.25 μm resolution, leading to the minimal features in the range of 0.13 μm. The changes to short exposure wavelength possess challenges and present new highly desired possibilities. Using advanced lithography, 90-nm features were achieved. Different lithography processes commonly applied are photolithography, screen printing, electron-beam lithography, x-ray lithography (high-aspect ratio technology), and so forth. Althoughmachine topologies and configurations vary, magnetic and insulating materials, magnets, and windings are used in all motion devices. Figure 17.12 illustrates the electroplated 10 μm wide and thick with 10 μm spaced insulated copper microwindings (deposited on ferromagnetic cores), microstructures, and permanent magnets (electroplated NiFe alloy). Finally, it must be emphasized that usually the size of microelectromechanical motion devices is defined by the torque and power densities specified. Those requirements, as well as physical limits, define the device dimensionality.

Acknowledgments

The author sincerely acknowledges the support from the NSF under the NSF awards DUE 0311588 and EEC 0407281. *Disclaimer*—Any opinion, findings, and conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

1. V. Giurgiutiu and S. E. Lyshevski, *Micromechatronics: Modeling, Analysis and Design with MATLAB*, CRC Press, Boca Raton, FL, 2003.
2. S. E. Lyshevski, *MEMS and NEMS: Systems, Devices, and Structures*, CRC Press, Boca Raton, FL, 2001.
3. S. E. Lyshevski, *Nano- and Micro-Electromechanical Systems: Fundamental of Micro- and Nano-Engineering*, CRC Press, Boca Raton, FL, 2005.
4. S. A. Campbell, *The Science and Engineering of Microelectronic Fabrication*, Oxford University Press, New York, 2001.
5. G. T. A. Kovacs, *Micromachined Transducers Sourcebook*,WCB McGraw-Hill, Boston, MA, 1998.
6. M. J. Madou, *Fundamentals of Microfabrication*, CRC Press, Boca Raton, FL, 2001.

II

Computers and Logic Systems

15

Motion Control

Rahul Kulkarni
National Instruments, Inc

15.1 Introduction to Motion Control

15.1.1 Definition

Motion control involves precisely controlling the position, velocity, and torque of a rotational or linear electromechanical device. Examples of such devices include hydraulic pumps, linear actuators, and electric motors.

15.1.2 History

In 1831, Michael Faraday discovered electromagnetic induction—electricity being generated in a wire by means of the electromagnetic effect of a current in another wire, and magneto-electric induction—electricity being generated in a wire by means of the electromagnetic effect of a magnet. These results led to the creation of the modern electric motor.

The industrial revolution put the motor to use in a variety of applications. A single motor was typically used by multiple machines connected to a drive train using belts for transmission and gears for speed reduction. A clutch was used to engage and disengage different machines on the drive train.

Complex machines were designed for mass production using a motor connected to amechanical system based on gears and cams. Hundreds of parts were produced on the same line as long as no parameter in the design changed.

As world moves from mass production to mass customization, machine designers want the same machine to handle a variety of different products at different times in the day, with minimal

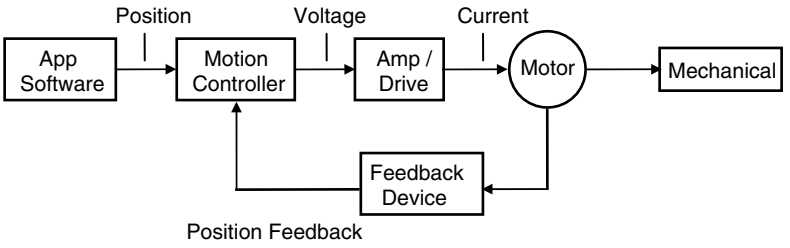


FIGURE 15.1 Components of a motion control system.

changeover time. Servo-based motion control systems enabled tight control over position and move profiles. Thus most machines today are complex electromechanical systems involving servo-based control.

15.2 Components of a Typical Motion Control System

A typical motion control system consists of a variety of components including user interface, motion controller, drive, motor, feedback device, and motion I/O. These components work together seamlessly to control the position, velocity, or torque of an electromechanical device. Figure 15.1 shows the different components of a motion control system.

15.2.1 User Interface

The user interface component accepts commands from the engineer or operator and sends them to the motion controller for execution. These commands include parameters such as target position, and motion control profiles for smooth movement. The user interface typically includes a graphical interface with switches and knobs to resemble the control panel of a machine in operation, and elements such as graphs or XY plots to represent the current position of the object in motion.

15.2.2 Motion Controller

A motion controller is at the center of a typical motion system, which consists of supervisory control, trajectory generation, and a control loop. The controller converts high-level user commands into command signals that drives use to move actuators. The motion controller also monitors the system for error conditions, faults, and asynchronous events that can cause the system to change speed, direction, or start/stop the actuators.

15.2.3 Drive

In a motion control system, the drive is the component that translates the command signals from the controller to current that produces torque or rotation in the motor. Drives can be classified on the basis of the type of motor they actuate—brushed servo, brushless servo, or stepper, or on the basis of the level of computation carried out in the drive—analog or digital. We will learn more about different types of drives in Sections 15.4 and 15.5.

15.2.4 Motor

Motors turn electrical energy into mechanical energy and produce the torque required to move to the desired target position. Motors are designed to provide torque to some mechanical elements including linear slides, robotic arms, and special actuators. There are three main types of motors—brushed DC

TABLE 15.1 Types of Motors

	Pros	Cons	Applications
Stepper Motors	Inexpensive, generally run open loop, good low-end torque, clean rooms	Noisy and resonant, poor high-speed torque, not for hot environments, not for variable loads	Positioning, micro movement
Brushed DC Servo Motors	Inexpensive, moderate speed, good high-end torque, simple drives	Maintenance required, no clean rooms, brush sparking causes EMI and danger in explosive environments	Velocity control, high-speed position control
Brushless Servo Motors	Maintenance-free, long lifetime, no sparking, high speeds, clean rooms, quiet, run cool	Expensive and complicated drives	Robotics, pick-and-place, high-torque applications

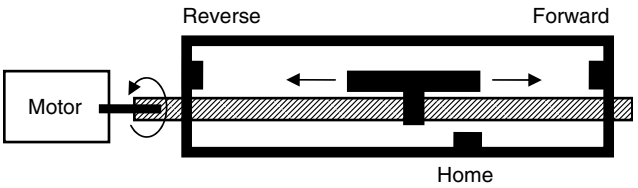


FIGURE 15.2 Limit and home switches in a motion control system.

servo motors, brushless servo motors, and stepper motors. Table 15.1 summarizes the pros, cons, and typical application areas for each type of motor.

15.2.5 Feedback Device

In most motion control systems, amethod for measuring the characteristic being controlled (i.e., position, velocity, etc.) needs to be implemented. This is known as feedback. Some of the different kinds of feedback used in motion control are relative position, absolute position, velocity, and analog voltage feedback. For relative position feedback, the most common device used is the incremental encoder. This device increments as it turns and can be used to keep track of the total distance traveled from some point of reference. The most common absolute position feedback devices are absolute encoders and resolvers. An absolute encoder is similar to an incremental encoder except that it does not need to have a reference position and can know what position the shaft is in on start-up. A resolver is also another absolute position device that is commonly used for both position and velocity feedback. For velocity feedback, a tachometer is commonly used. Some applications require feedback other than the type that has been mentioned above. In these cases, analog feedback describing some characteristic of the system such as temperature or pressure can be useful in controlling motors. A position feedback device is not required for some motion control applications, such as controlling stepper motors, but is vital for servo motors.

15.2.6 Motion I/O

A motion control system also includes digital I/O such as limit switches, home switches, position compare outputs, and position capture inputs.

Limit switches provide information about the end of travel to help you avoid damaging your system. When a motion system hits a limit switch, it typically stops moving. Home switches, on the other hand, indicate the system home position to help you define a reference point. This is important for applications such as pick-and-place (Figure 15.2).

Triggers such as position compare outputs or position capture inputs help when synchronizing moves with other external devices and sensors such as data acquisition and vision.With position compare outputs, you can set up a trigger that fires at a defined position. This type of action is very useful in operations such as scanning, where you might want to trigger a system to take measurements at a series of defined positions. Position capture inputs, on the other hand, cause the motion controller to immediately capture a position on the occurrence of an external event such as a sensor being encountered while moving. This is useful for resynchronizing the position of an axis (registration moves for "cut on the fly" applications) or recording positions for analysis (scanning defects on a printed circuit board (PCB)).

15.3 Functions of a Motion Controller

If we look at the functions of a motion controller, they can be broadly split up into three components—trajectory generation that involves path planning on the basis of the profile desired by the user, supervisory control that includes system initialization, event handling, fault detection, and command response, and the control loop that includes closing the position and velocity loops on the basis of feedback. Figure 15.3 shows the parts and processes of a typical motion controller.

These three components of a motion controller work in synchronization with each other. The supervisory control is the interface that handles user-defined requests and commands the trajectory generator to start calculating set points. The trajectory generator calculates a set point per iteration and writes it to the control loop. The control loop takes this set point and using a control algorithm ensures that the actuator follows this set point accurately. In the case, the control loop can run faster than the trajectory generator; the control loop uses an interpolation algorithm that creates intermediate set points per iteration of the control loop.

15.3.1 Supervisory Control

The supervisory control is the main loop of the motion control system. This loop intercepts commands from the user and signals the trajectory generator to start/stop moves. The supervisory control loop also monitors all I/O needed to perform initialization tasks, such as finding the reference or origin. Supervisory controlmonitors I/O for fault conditions and user-defined events and accordingly commands the trajectory generator to take action.

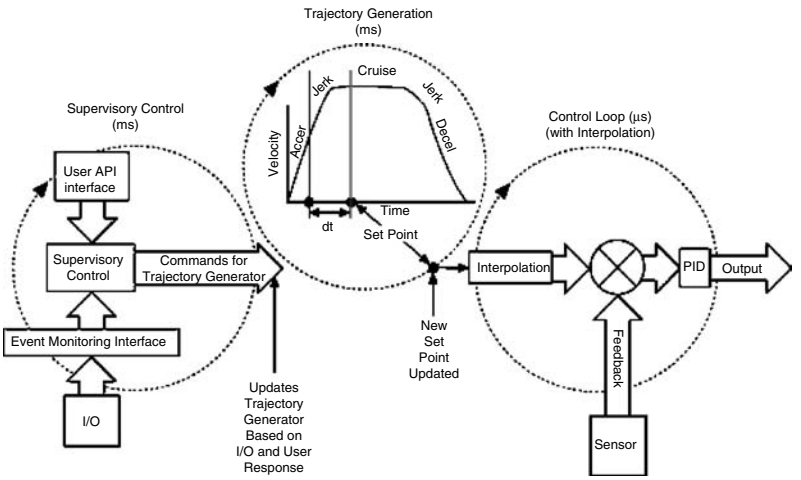


FIGURE 15.3 Motion controller architecture.

15.3.1.1 Axis Initialization (Reference Moves)

A reference move resets the axes of a motion script to a known location or state. Reference moves are generally used at the beginning of a motion script to prepare the motion system for subsequent motion. The following are valid types of reference moves—finding the home switch, finding the encoder index, finding the forward limit switch, finding the reverse limit switch, and finding the center for a system.

Finding the Home Switch

Home reference moves perform a course reference move by traversing the entirety of the motion path in search of a physical home switch. A home switch consists of a logical ON state bordered by a front edge and a back edge. Home reference moves can operate when there is a physical home switch present or if there is a physical end-of-travel present, such as on a stage.

Finding the Encoder Index

Index reference moves perform a find reference move by moving the axis until the index state on the encoder is reached. The index state is a specific position on the 360° encoder that is passed during each revolution of the motor. An index reference move is usually performed after a home reference move for fine reference adjustment. The encoder index signal is accurate to one encoder count and provides amuch more repeatable reference than using just a home switch edge.

Finding the Forward Limit and Reverse Limit Switches

A forward limit reference move initiates a search sequence to find the forward limit. A reverse limit reference move initiates a search sequence to find the reverse limit.When the search operation is initiated, the axis starts moving in the direction of the limit. When the limit in that direction is encountered, the search is completed. A fine-tuning operation is performed when the limit is reached to minimize the effects of motion system windup, backlash, and/or home sensor hysteresis. This type of reference move is used when a home switch is not available on a system.

Finding the Center

Center reference moves initiate a search sequence positioning the axis in the middle of the forward and reverse limits.When the search direction is forward, the axis starts moving in the forward direction.When the limit in that direction is encountered, the direction is reversed to find the reverse limit. After both limits are reached, a center position can be calculated from the recorded positions. The axis then proceeds to the calculated center.

After finding the reference position for an axis the internal position register for that axis is typically updated by:

Resetting the Position Counter Resetting the position counter for an axis causes the axis to start counting its position at a specified starting number. This starting number is typically zero, but resetting to a nonzero number can be useful when you want to begin the moves relative to an absolute zero, rather than the current position. For example, a steering system for a vehicle might come online with the wheels turned 7000 counts to the right of absolute zero. In this example, absolute zero points the wheels straight ahead. To ensure the motion system recognizes that the wheels are not starting off pointed straight ahead, reset the position to the absolute count of 7000.

Offsetting the Axis Position This is achieved by moving the axis by an amount equal to a predetermined offset. Offsetting the position preserves the value of performing the reference moves, but adds another move to create distance from the home switch or index. This is usually done to prevent the axis from sitting on a switch or encoder index.

15.3.1.2 Electronic Gearing

Electronic gearing allows one slave motor to be driven in proportion to a master motor or feedback sensor, such as an encoder or torque (analog) sensor. As the slave follows the master position at a constant ratio, the effect is similar to that of two axes mechanically geared. For example, every turn of the master axis may cause a slave axis to turn twice.

Electronic gearing has several advantages over mechanical gears. The most notable is flexibility because you can change gear ratios on-the-fly. The other major advantage to electronic gearing is that you can superimpose amove over a geared axis. The superimposedmove is added to the geared profile of the slave axis, which allows the slave axis to be synchronized on-the-fly (Figure 15.4).

The gear ratio is used to determine how far the slave axis must move in proportion to the master when gearing is enabled. The gear ratio can be absolute or relative.

$$\text{Slave axis move} = \text{Master axis position} \times \text{Gear ratio}$$

For example, if you have a gearing ratio of 2:1 (slave:master), the slave moves 20 counts when the master device moves 10 counts.

Absolute gearing behaves similarly to relative gearing in that when gearing is enabled, the slave axis follows the master axis movement as it is defined by the gear ratio. The difference between relative and absolute gearing is that the reference position calculated for the master axis is updated only when gearing is enabled. This difference is apparent when the gear ratio is updated on-the-fly (Figure 15.5).

For example, if the gear ratio is 2:1, the current master position is 1010, the current slave position is 3020, and the gear ratio is changed to 3:1; the slave axis jumps from 3020 to 3030 but the master position remains the same (Figure 15.6).

Changing a gear ratio on-the-fly during absolute gearing allows you to quickly synchronize the slave axis with the master axis.

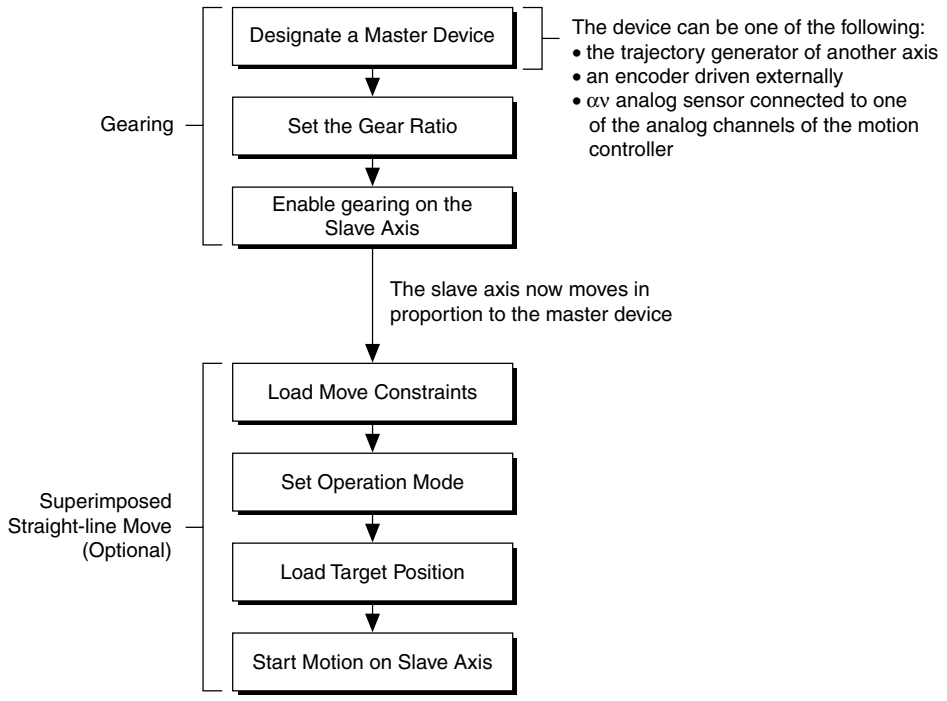


FIGURE 15.4 Electronic gearing algorithm.

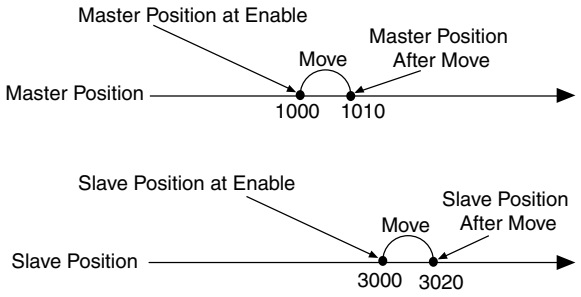


FIGURE 15.5 Relative gearing at enable.

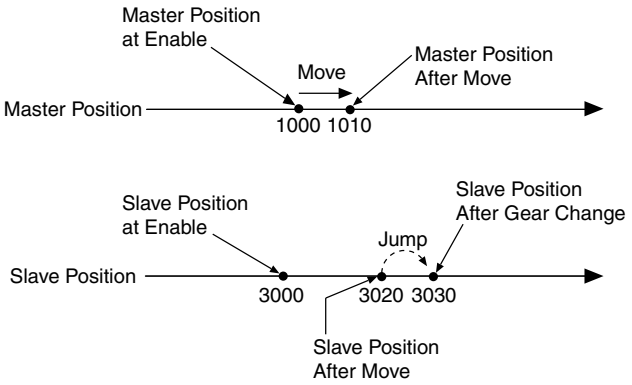


FIGURE 15.6 Absolute gearing at gear ratio change.

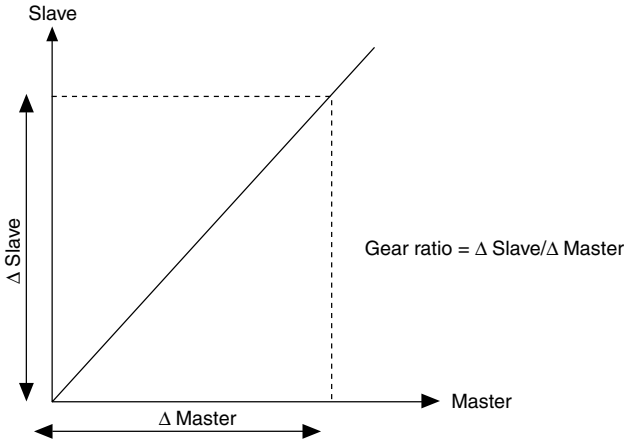


FIGURE 15.7 Master/slave ratio in gearing.

15.3.1.3 Electronic Camming (Includes Registration Moves)

Electronic camming operates similarly to electronic gearing in that the move distance of an axis is proportional to the move distance of its master device. Camming differs from gearing in how the master/slave ratio is handled by the motion controller. Gearing is used in applications where a constant gear value creates a linear slave position profile, as shown in Figure 15.7.

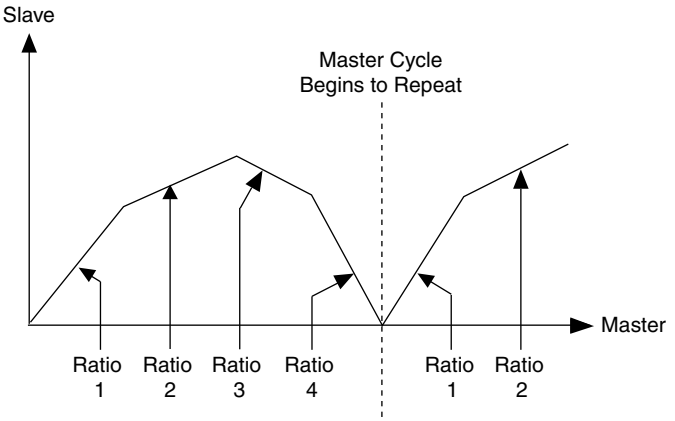


FIGURE 15.8 Multiple camming gear ratios.

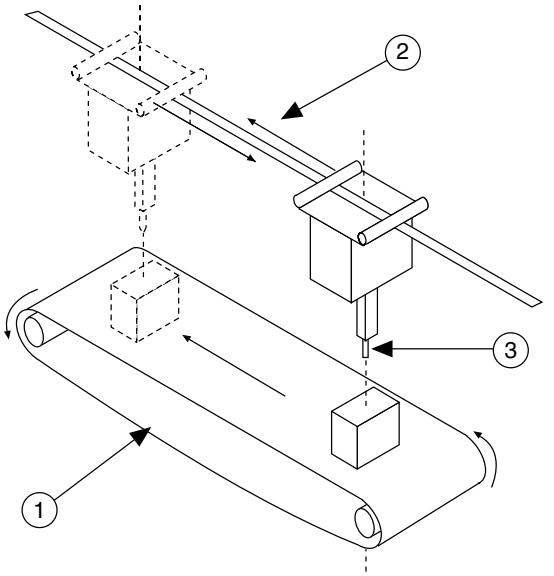


FIGURE 15.9 Welding application showing (1) conveyor belt, (2) movement of the welder as it follows the object and then returns to the initial position, and (3) welding point.

Camming creates a more flexible profile by using more master/slave ratios. These ratios are handled automatically by the motion controller, allowing precise switching of the gear ratios, as shown in Figure 15.8. Camming is used in applications where the slave axis follows a nonlinear profile from a master device.

Figure 15.9 shows that a welding point moves to the first position, and then welds the material for a couple of seconds. Because the conveyor belt keeps moving at a constant rate, the welding point must follow the material at the same speed as the conveyor belt during the weld process. When the welding process is finished for one item, the welding point must quickly return to its initial position, and the process repeats.

In this application, the master device is the position encoder attached to the conveyor belt, and the slave axis is the actuator that moves the welding point. The slave axis repeatedly performs a two-segment movement—first, it follows the material with the same speed as the conveyor belt, and next, it returns to

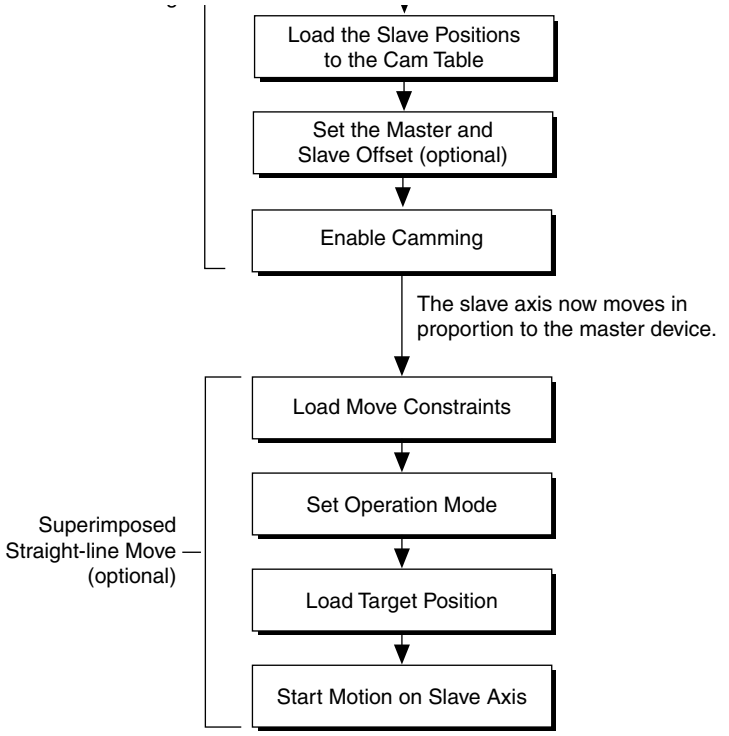


FIGURE 15.10 Camming algorithm.

the initial position as the next material approaches. Each segment of the move is represented with a gear ratio that dictates how fast and which direction the welding point is moving relative to the conveyor belt. This application requires the slave axis to switch from one ratio to the other at the correct master position, otherwise the welding process is not repeatable. If this application used gearing instead of camming, the latency, or delay, of loading a new gear ratio might cause an accumulation of position errors.

Similar to gearing, in a camming application, a slave axis can perform any move when camming is enabled. The move profile is superimposed over the camming profile (Figure 15.10).

15.3.1.4 Exception Handling (E-Stop and Limit Switches)

A motion controller needs to be able to detect fault conditions and handle these exceptions in order of severity. Examples of exceptions include scenarios where the actuator hit a forward or reverse limit switch or if the E-stop switch was pressed.

Limit switches can be either hardware or software limits. The hardware limit inputs are typically connected to end-of-travel limit switches or sensors. An enabled limit input causes a full torque halt stop on the axis when the input becomes active. Active limit inputs also prohibit attempts to start motion that would cause additional travel in the direction of the limit. Similarly, software limits are often used to restrict the range of travel further and avoid ever hitting the hardware limit switches. An enabled software limit causes the axis to smoothly decelerate to a stop when the limit position is reached or exceeded.

15.3.1.5 I/O Synchronization (Captures, Compares, Time-Sampled Data Logging)

You may need to synchronize motion control with data acquisition or image acquisition devices within the same systemusing position compares, position captures and time-sampled data logging. Let us understand first what each of these terms mean.

In scanning applications, you typically inspect an object such as a wafer or an LCD display for defects using a camera. The wafer or LCDdisplay rests on an XY stage that moves in two dimensions. The objective

of the scan is to cover as much space on the wafer as possible in the shortest amount of time and trigger the camera to take an image at specific points along the scan. In such applications, you need to send a digital signal (called a breakpoint) to the trigger line of a camera to command it to take an image based on a specific position reached by the XY stage. This functionality is called position compares (breakpoints), which can be classified into absolute breakpoints, relative position breakpoints, modulo breakpoints, periodically occurring breakpoints, and buffered breakpoints.

In some cases, you may need to synchronize position with some measurement occurring externally to the motion controller. You can think of this conceptually as a conveyor belt application running at a constant velocity. Each time a manufactured item moves down the conveyor belt and breaks a light beam across the belt, a stamp is placed on the item, a set distance from the point where the beam is broken. This allows all items to be stamped in the same relative position without necessarily being equally spaced along the conveyor belt. You need to command amove of a specified length relative to the position at which a trigger input is received to perform such a stamping operation. This functionality is called high-speed position captures (triggers), which could be buffered or nonbuffered.

In vision-guided motion applications, you may need to log position and velocity data periodically for off-line analysis and correlation with vision data obtained concurrently. This functionality is known as time-sampled data logging. Data is collected at hardware/firmware timed period and stored in the motion controller's internal memory. This data can be retrieved from the motion controller asynchronously.

Position Compares (Breakpoints)

You can configure a motion controller to generate a trigger signal or toggle the state of an I/O line when the motor reaches a specified position. This specified position is called a breakpoint and is similar to a switch that is triggered after the motor passes a certain position. To use breakpoints, you must have a board with closed-loop encoder feedback, because the controller needs the encoder position to determine where the breakpoint occurs. You can specify breakpoints for both servo motors and stepper motors. You can use breakpoints as a trigger for another piece of hardware because it sends a signal to an I/O line. For example, when a motor reaches a certain position, you can use a breakpoint signal to initiate an image capture or a data acquisition operation. The five ways in which breakpoints can be configured are as follows:

Absolute Mode: With absolute position breakpoints, you can trigger external activities as the motors reach specified positions. For example, if you need to use an image acquisition device to capture an image from a certain position while the device under test is in continuous motion, the motion controller must be able to trigger the image acquisition device as it reaches those positions. The current position is continuously compared against the specified breakpoint position by the encoder circuitry to produce a latency of less than 100 ns. After a breakpoint triggers, you must re-enable it for the breakpoint to work again. In certain cases, such as buffered and periodic breakpoints, the motion controller automatically re-enables the breakpoints.

Relative Mode: Relative position breakpoints trigger events on the basis of a change in position relative to the position at which the breakpoint was enabled. Instead of keeping track of absolute positions and the current position, you can use relative breakpoints to specify the breakpoint relative to the position where the breakpoint is enabled. For example, if you are creating a motion control system to control the 2D movement of a microscope, you might use relative position breakpoints to move the microscope a specific distance in a direction, and then hit a breakpoint that triggers a camera snap. The relative breakpoint is useful in this example because the current position is not important. The application must move the axis a specific number of counts from wherever it is, and then generate a breakpoint.

Periodic Mode: You can program the motion controller to generate multiple breakpoints at fixed and exact intervals, regardless of the direction of travel or velocity. Periodic breakpoints require that you specify an initial breakpoint and an ongoing repeat period.When enabled, the periodic breakpoints begin when the initial breakpoint occurs. From then on, a new breakpoint occurs each time the axis

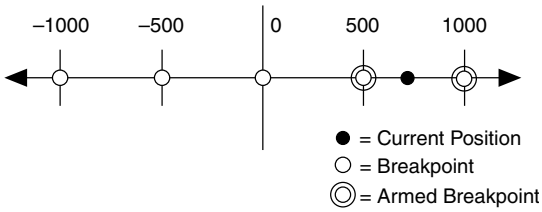


FIGURE 15.11 Periodic breakpoint every 1000 counts/steps.

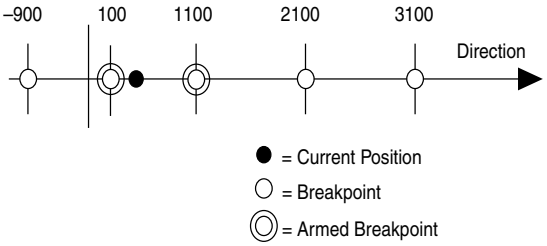


FIGURE 15.12 Breakpoint modulus of 500.

moves a distance equal to the repeat period, with no re-enabling required. For example, if an axis is enabled at position zero, the initial breakpoint is set for position 100, and the breakpoint period is set at 1000, then the axis behaves as shown in Figure 15.11.

Modulo Mode: Modulo breakpoints use a breakpoint window that defines an area around the current position. The two breakpoints around the current position are always enabled. The breakpoint modulus creates a repeat period for the breakpoints, and the breakpoint position is the offset from absolute zero. For example, to create a breakpoint every 500 counts, set the repeat period to 500 and the breakpoint position to 0. If the breakpoint is enabled when the axis is at 710, the breakpoints at 1000 and 500 are both armed, as shown in Figure 15.12.

Buffered Mode: Buffered breakpoints are a way of specifying absolute or relative breakpoint positions in a buffer that can be preloaded into the motion controller memory. The motion controller then automatically arms breakpoint positions from the buffer as and when the breakpoints triggers. The, arming of breakpoints occurs on a firmware-timed basis, which provides higher bandwidth.

Position Captures (Triggers)

Some motion control applications require that you execute amove and record the position where external triggers occur. For example, you might be aligning two fiber-optic cables, in which case, the maximum optical power needs to correspond with the alignment position. To align the fibers, the external device that is recording the optical power must trigger the motion controller so that you can synchronize and analyze positions and optical power measurements. This functionality is known as high-speed capture or trigger inputs. The implementation for high-speed capture is divided into the buffered and nonbuffered high-speed capture methods:

Buffered High-Speed Captures: With buffered high-speed capture, you can create a buffer that holds captured positions that you can read asynchronously from the motion controller. The motion controller automatically arms the next high-speed capture, and writes the captured high-speed data into its onboard buffer. The enabling of high-speed capture occurs on a firmware-timed basis, which provides better frequency than the nonbuffered high-speed capture method.

Nonbuffered High-Speed Captures: With nonbuffered, high-speed capture, you can configure a single high-speed capture event. For multiple high-speed captures, you must re-enable the high-speed capture each time before it triggers.

Time-Sampled Data Logging
Some motion controllers can acquire a hardware-timed buffer of position and velocity data for time-sampled data logging. After you command the motion controller to acquire position and velocity data, a separate acquire data task is created in the motion controller, which reads time-sampled position and velocity data into a FIFO buffer on the motion controller. You can read data in from this buffer number of samples that consist of position data, in counts or steps, and velocity data, in counts/s or asynchronously from the user interface on the host computer. Typically, the FIFO buffer holds a certain steps/s.

15.3.2 Trajectory Generation

The trajectory generator on a motion controller is responsible for the path planning on the basis of geometry specified by the user. Different kinds of geometries are typically supported: straight lines, circular arcs, helical arcs, and spherical arcs. In addition to the predefined geometries, the user can specify a custom geometry via a buffer of points that the motion controller uses to interpolate through using a cubic spline algorithm. This is known as a contoured move.

The user can also specify the velocity profiles that he wants used when traversing desired geometries. This can be specified using move constraints that include maximum velocity, maximum acceleration, and maximum deceleration and jerk (rate of change of acceleration/deceleration). These parameters are used to generate a trapezoidal or s-curve profile. The trajectory generator on the motion controller ensures that these move constraints are never violated while making a move. In addition to this, the user can specify custom profiles by providing position, velocity, and time data (Position Velocity Time, PVT). The trajectory generator also typically supports blending velocity profiles. In this case, the users can concatenate multiple profiles specifying the point of concatenation.

The output of the trajectory generator are set points that are fed to the control loop to act on the basis of the path that needs to be followed.

To be highly reactive and allow for on-the-fly changes, the trajectory generator on the motion controller creates these set points in real time. This allows users to pre-empt the current trajectory by changing end positions, velocity, and acceleration. It also allows for stopping and restarting a move that is currently in progress.

15.3.2.1 Straight-Line Moves

A straight-line move executes the shortest move between two points. Straight-line moves can be position based or velocity based.

Position Based
Position-based straight-line moves use the specified target position to generate the move trajectory. For example, if the motor is currently at position zero, and the target position is 100, a position-based move creates a trajectory that moves 100 counts (steps).

The controller requires the following information to move to another position in a straight line:

- Start position—current position, normally held over from a previous move or initialized to zero
- End position—also known as the target position, or where you want to move to
- Move constraints—maximum velocity, maximum acceleration, maximum deceleration, and maximum jerk

The motion controller uses the given information to create a trajectory that never exceeds the move constraints and that moves an axis or axes to the end position you specify. The controller generates the trajectory in real time, so you can change any of the parameters while the axes are moving. Figure 15.13 illustrates a typical algorithm for a position-based straight-line move.

Velocity Based
Some motion applications require moves that travel in a straight line for a specific amount of time at a given speed. This type of move is known as velocity profiling or jogging. You can use a motion control

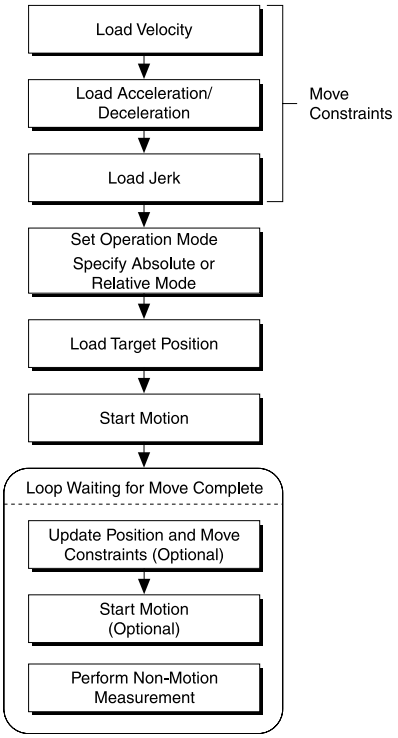


FIGURE 15.13 Position-based straight-line move algorithm.

application to move a motor at a given speed for a specific time, and then change the speed without stopping the axis. The sign of the loaded velocity specifies the direction of motion. Figure 15.14 illustrates an algorithm for a typical velocity-based straight-line move.

15.3.2.2 Arc Moves

An arc move causes a coordinate space of axes to move on a circular, spherical, or helical path. You can move two-dimensional vector spaces in a circle only on a 2D plane. You can move a three-dimensional vector space on a spherical or helical path. Each arc generated by the motion controller passes through a cubic spline algorithm that ensures the smoothest arc. A cubic spline algorithm generates multiple points between every two points of the arc, ensuring smooth motion, minimum jerk, and maximum accuracy at all times.

Circular Arcs

A circular arc defines an arc in the XY plane of a 2D or 3D coordinate space. The arc is specified by a radius, start angle, and travel angle. In addition, like all coordinate space moves, the arc uses the values of move constraints—maximum velocity, maximum acceleration, and maximum deceleration.

To move axes in a circular arc, the motion controller needs the following information:

- Radius—specifies the distance from the center of the arc to its edge.
- Start Angle—orients the arc on its plane using the starting point as an axis to spin around. Because the starting point for a new arc is fixed on the basis of the current position, moving its center around the starting point alters the orientation of a new arc.
- Travel Angle—indicates how far the arc travels in a 360° circle. For example, a travel angle of 90° executes a quarter circle, a travel angle of 360° creates a full circle, and a travel angle of 720° creates two full circles (Figure 15.15).

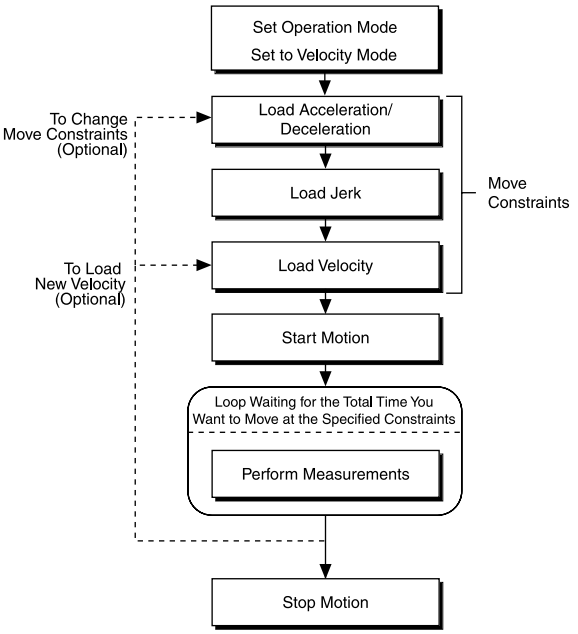


FIGURE 15.14 Velocity-based straight-line move algorithm.

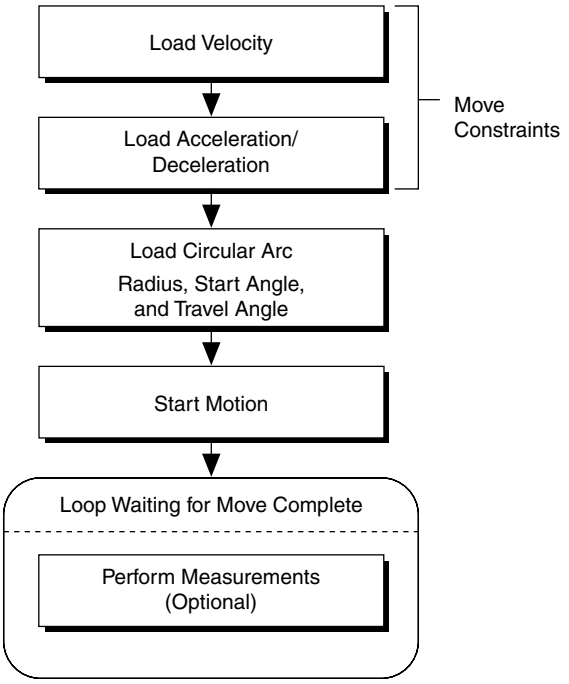


FIGURE 15.15 Circular arc move algorithm.

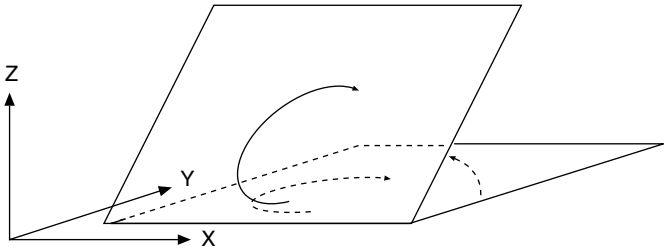


FIGURE 15.16 Changing pitch by rotating the X-axis.

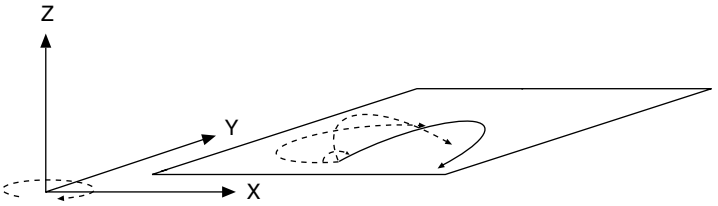


FIGURE 15.17 Changing yaw by rotating the Z-axis.

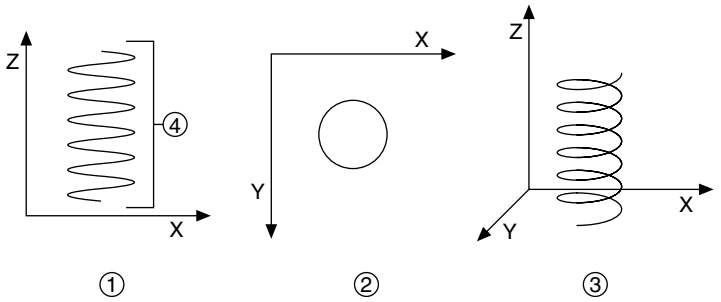


FIGURE 15.18 Helical Arc showing (1) side view, (2) top view, (3) isometric view, and (4) linear travel.

Spherical Arcs

A 3D spherical arc defines a 2D circular arc in the XY plane of a coordinate system that is transformed by rotation in pitch and yaw from the normal 3D coordinate space (XYZ), as shown in Figures 15.16 and 15.17.

In the transformed X′Y′Z′ space, the 3D arc is reduced to a simpler 2D arc. The 3D arc is defined as a 2D circular arc in the X′Y′ plane of a transformed vector space X′Y′Z′. This transformed vector space, X′Y′Z′, is defined in orientation only, with no absolute position offset. Its orientation is relative to the XYZ vector space, and is defined in terms of pitch and yaw angles.When rotating through the pitch angle, the Y and Y′ axes stay aligned with each other while the X′Z′ plane rotates around them. When rotating through the yaw angle, the Y′ axis never leaves the original XY plane, as the newly-defined X′Y′Z′ vector space rotates around the original Z-axis. The radius, start angle, and travel angle parameters also apply to a spherical arc that defines the arc in two dimensions.

Helical Arc

Ahelical arc defines an arc in a 3Dcoordinate space that consists of a circle in the XY plane and synchronized linear travel in the Z-axis. The arc is specified by a radius, start angle, travel angle, and Z-axis linear travel. Linear travel is the linear distance traversed by the helical arc on the Z-axis, as shown in Figure 15.18.

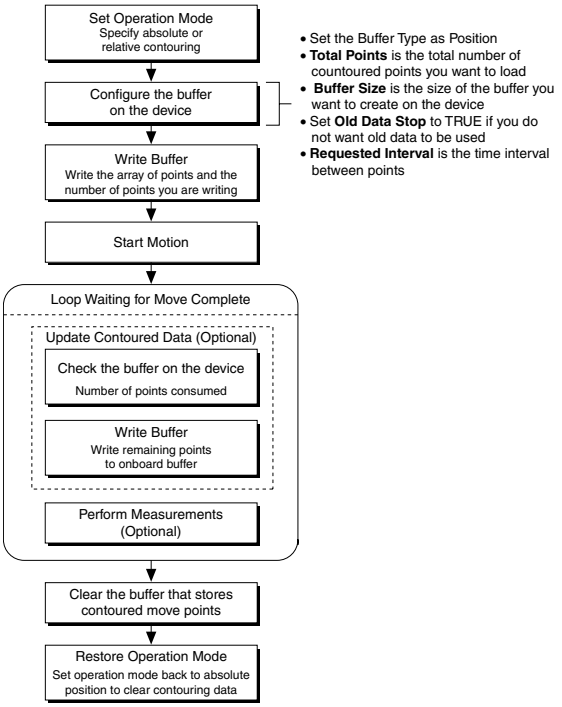


FIGURE 15.19 Contoured move algorithm.

15.3.2.3 Contoured Moves

A contoured move moves an axis or a coordinate space of axes in a pattern that you define. The trajectory generator on the motion controller is not used during a contoured move. The controller takes position data in the form of an array, and splines the data before outputting it to the control loop.

Contoured moves are useful when you want to generate a trajectory that cannot be constructed from straight lines and arcs. To ensure that the motion is smooth with minimum jerk, the motion controller creates intermediate points using a cubic spline algorithm. The move constraints commonly used to limit other types of moves, such as maximum velocity, maximum acceleration, maximum deceleration, and maximum jerk, have no effect on contoured moves. These need to be embedded in the series of contoured points provided (by spacing the points) (Figure 15.19).

15.3.2.4 Position Velocity Time Profiles

Position velocity time moves give you maximum control over the velocity of the move at certain times throughout the move. PVT moves specify a series of move segments by giving several move points for each axis. In addition, PVT moves specify both the time, in ms, each move segment takes and the velocity at which all axes are traveling at the end of each segment.

The following table demonstrates typical PVT inputs:

X	Y	Z	V	T
4317	4317	4317	1000	750
12245	11245	13245	500	1500
15173	15173	15173	500	2500

T: Time begins at 0 for each PVT move, and is cumulative through the end of the move. In this example, there are three segments of duration 750, 750, and 1000.

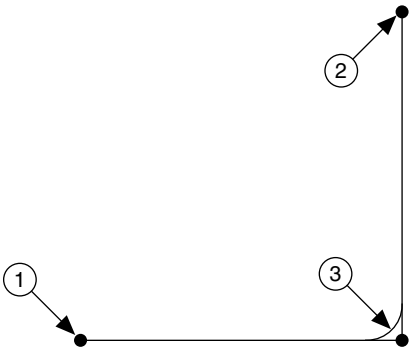


FIGURE 15.20 Two blended straight-linemoves showing (1) starting position, (2) end point, and (3) corner rounded by blending.

V: PVT moves assume a beginning velocity of zero. Therefore, you must bring each axis to a complete stop in the previous move before beginning a PVT move.

X, Y, Z: PVT moves start a position counter for each axis at 0, independent of where you are in relation to the original reference move. The numbers in the spreadsheet or text file are absolute in relation to the new counter. In this example, there are three move segments of lengths 4317, 7928, and 2928, respectively.

15.3.2.5 Blending

Blending, also called velocity blending, superimposes the velocity profiles of two moves to maintain continuous motion. Blending is useful when continuous motion between concatenated move segments is important. Examples of some applications that can use blending are scanning, welding, inspection, and fluid dispensing. Blending must occur on velocity profiles of two move segments, so the end positions of each move segment may or may not be reached. For example, if you are blending two straight-line moves that form a 90° angle, the blended move must round the corner to make the move continuous. In this case, the move never reaches the exact position where the two straight lines meet, but instead follows the rounded corner, as shown in Figure 15.20.

Motion controllers can perform blending between two straight-line moves, between two arc moves, or between straight-line and arc moves. Blending typically is not supported for reference and contoured moves.

Superimposing Two Moves

Superimposing two moves is the most common use of blending. In this case, the motion controller tries to maintain continuous motion by superimposing the two move segments such that the second move segment starts its profile while the first move is decelerating, as shown in Figure 15.21.

The velocity during the superimposition depends on the cruising velocity, deceleration, and jerk of the first move segment, and the jerk, acceleration, and cruising velocity of the second move segment.

Blending Moves after the First Move Is Complete

Blending moves after the first move is complete; this causes the first move segment to come to a complete stop before starting the profile of the second segment, as shown in Figure 15.22. This type of blending is useful if you want to start two move segments, one after the other, with no delay between them.

Blend after Delay

You can blend two moves after a delay at the end of the first move. Blending in this manner is useful if you want to start two move segments after a deterministic delay. The two move segments can be either straight-line moves or arc moves.

Because blending occurs on velocity profiles, the effect of reaching the end positions of the move segments and the maximum velocity depends on the velocity, acceleration, deceleration, and jerk loaded

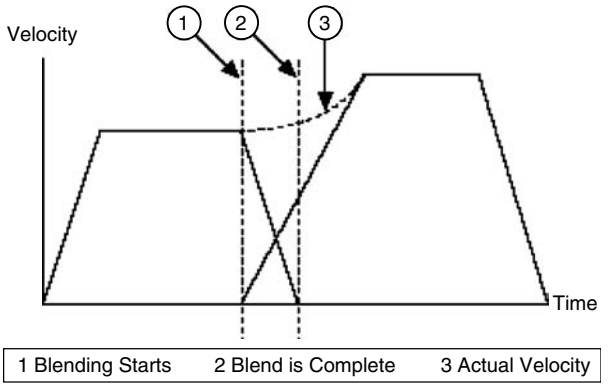


FIGURE 15.21 Superimposing two moves.

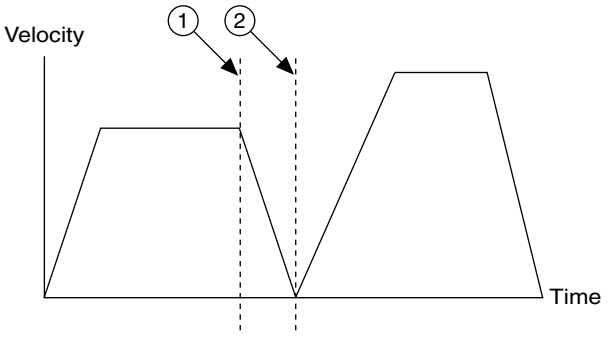


FIGURE 15.22 Blending after move complete showing (1) blending starting point and (2) blending complete.

for the two move segments. Because two move segments are always used while blending, it is very important that you wait for the blend to complete before loading the next move segment you want to blend.

15.3.2.6 Second- and Third-Order Profiles (Trapezoidal/s-Curve)

The trajectory generator uses the commanded parameters of position, acceleration, and velocity for a particular move to determine how much time it spends in the three primary move segments—acceleration, constant velocity, and deceleration—that define a velocity profile.

Trapezoidal Velocity Profile

When you use a trapezoidal profile, the axes accelerate at the acceleration value you specify, and then cruise at the maximum velocity you load. On the basis of the type of move and the distance being covered, it may be impossible to reach the maximum velocity you set. The velocity of the axis, or axes, in a coordinate space never exceeds the maximum velocity loaded. The axes decelerate to a stop at their final position, as shown in Figure 15.23.

S-curve Velocity Profile

The acceleration and deceleration portions of an s-curve motion profile are smooth, resulting in less abrupt transitions, as shown in Figure 15.24. This limits the jerk in the motion control system, but increases cycle time. The value by which the profile is smoothed is called the maximum jerk or s-curve value.

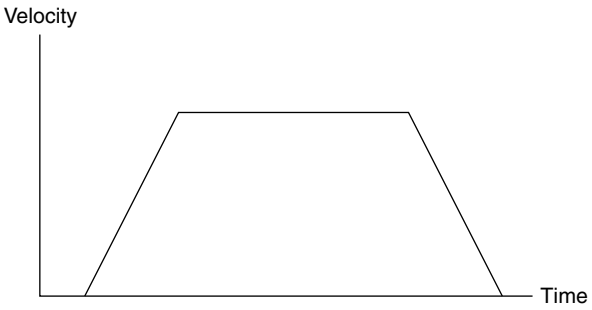


FIGURE 15.23 Trapezoidal move profile.

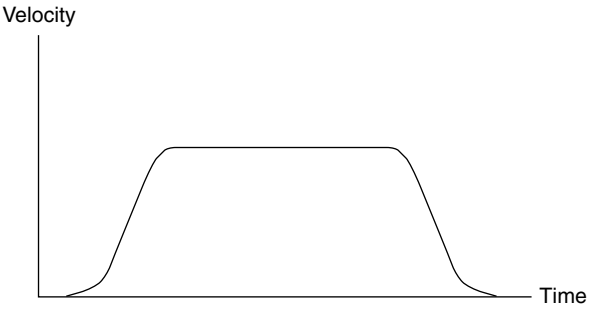


FIGURE 15.24 S-curve move profile.

15.3.2.7 Multiple Axis Synchronization (Coordinate/Vector Spaces)

With the exception of the arc move, you can execute all basic moves on either a single axis or on a coordinate space. A coordinate space is a logical grouping of axes, such as the XYZ axes. Arc moves always execute on a coordinate space. If you are performing a move that uses more than one axis, you must specify a coordinate space made up of the axes the move will use.

Typically, there exists a function in the motion controller to configure a coordinate space. This function creates a logical mapping of axes and treats the axes as part of a coordinate space. The function then executes the move generated by the trajectory generator on the vector, and treats all the move constraints as vector values.

Multistarts versus Coordinate Spaces

Coordinate spaces always start and end the motion of all axes simultaneously. You can use multistarts to create a similar effect without grouping axes into coordinate spaces. Using a multistart automatically starts all axes virtually simultaneously. To simultaneously end the moves, you must calculate the move constraints to end travel at the same time. In coordinate spaces, this behavior is calculated automatically.

15.3.3 Control Loop

15.3.3.1 Spline Interpolation

The trajectory generator and the proportional, integral, derivative (PID) control loop need not execute at the same loop rates.While the PID control loop needs to run at microseconds, the trajectory generator may generate a new set point every millisecond. To maintain the steady flow of set points from the trajectory generator to the PID control loop, a motion controller needs a spline interpolator that interpolates between set points given by the trajectory generator to create finer set points to be acted on by the control loop (Figure 15.25).

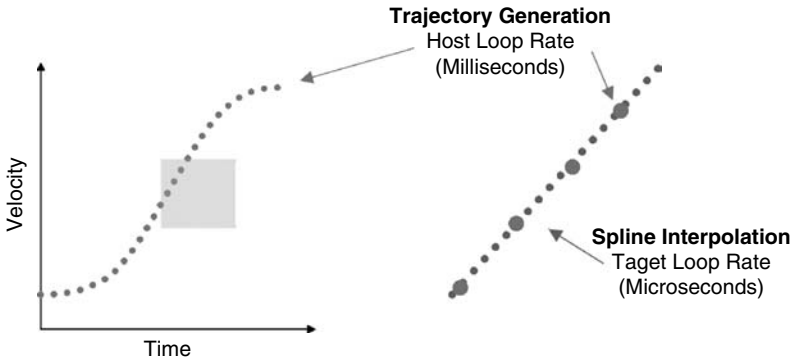


FIGURE 15.25 Spline engine function uses cubic spline algorithm to calculate interpolated positions between two positions from the trajectory generator.

An example of a spline interpolator is a cubic spline algorithm that uses four set points to calculate interpolated positions between two positions calculated by the trajectory generator. The polynomial used to calculate interpolated points can be represented as:

$$P(t) = a_3 t^3 \times a_2 t^2 \times a_1 t^1 \times a_0 t^0$$

Given four set points, P_0, P_1, P_2, and P_3, the spline coefficients are determined in real time as:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

The number of interpolated points created depends on the rate of the trajectory generator loop and control loop. For example, if the trajectory generator loop runs at 1 ms and the control loop runs at 100 μs, the spline engine calculates 1 ms/100 μs =10 interpolated points. Using the spline interpolator results in smoother motion and allows you to run the trajectory generator loop slower than the control loop.

15.3.3.2 Position and Velocity Loops

The control loop creates the command signal on the basis of the set point provided by the trajectory generator. In most cases, the control loop includes both a position and a velocity loop, but in some cases the control loop may include only a position loop. The position is typically read from encoders, but also may be read from analog inputs. The velocity is calculated from the position values, and may be read directly from a velocity sensor, such as a tachometer. Because feedback is not required for stepper motors, the control loop concerts the set point generated by the trajectory generator into stepper signals (step/direction).

The position and velocity control loops on the motion system correct for errors and maintain tight control of the trajectory. The position and velocity loops typically incorporate the PIDalgorithm. The PID algorithm consists of three basic coefficients: proportional, integral, and derivative, which are varied to get optimal response (Figure 15.26).

In addition to parameters such as proportional gain, integral gain, and derivative gain, the PID algorithm-implemented part of a motion control system includes additional parameters such as velocity feedback, velocity feed forward, and acceleration feed forward for optimal position control.

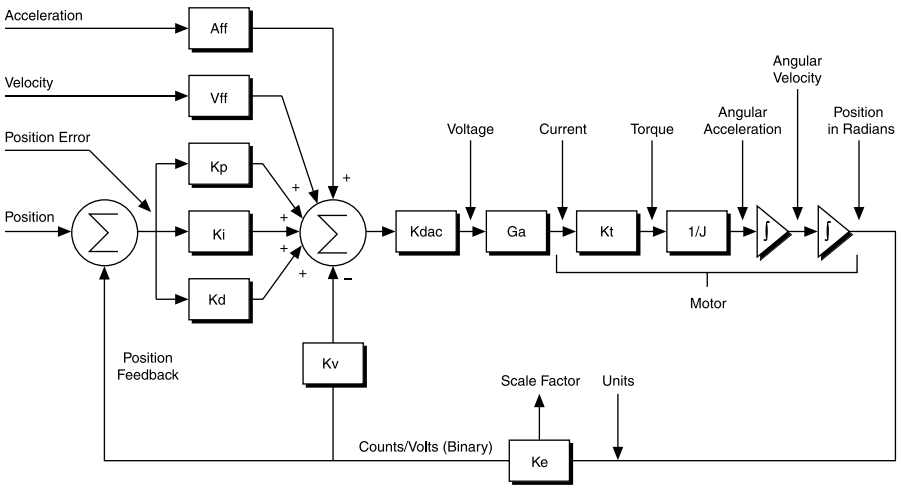


FIGURE 15.26 PID servo loop.

Kp (Proportional Gain)
The proportional gain (Kp) determines the contribution of restoring force that is directly proportional to the position error. This restoring force functions in much the same way as a spring in a mechanical system. An axis with too small a value of Kp is unable to hold the axis in position and is very soft. Increasing Kp stiffens the axis and improves its disturbance torque rejection. However, too large a value of Kp often results in instability.

Ki (Integral Gain)
The integral gain (Ki) determines the contribution of restoring force that increases with time, ensuring that the static position error in the servo loop is forced to zero. This restoring force works against constant torque loads to help achieve zero position error when the axis is stopped. In applications with small static torque loads, this value can be left at its default value of zero (0). For systems having high static torque loads, this value should be tuned to minimize position error when the axis is stopped. Although nonzero values of Ki cause reduced static position error, they tend to cause increased position error during acceleration and deceleration. This effect can be mitigated through the use of the integration limit parameter. Too high a value of Ki often results in servo loop instability.

Kd (Derivative Gain)
The derivative gain (Kd) determines the contribution of restoring force proportional to the rate of change (derivative) of position error. This force acts much like viscous damping in a damped spring and mass mechanical system. A shock absorber is an example of this effect. A nonzero value of Kd is required for all systems that use torque block amplifiers, where the command output is proportional to motor torque, for the servo loop operation to be stable. Too small a Kd value results in servo loop instability. With velocity block amplifiers, where the command output is proportional to motor velocity, it is typical to set Kd to zero or a very small positive value.

Kv (Velocity Feedback)
You can use a primary or secondary feedback encoder for velocity feedback. Setting the velocity feedback gain (Kv) to a value other than zero (0) enables velocity feedback using the secondary encoder, if configured, or the primary encoder if a secondary encoder is not configured. The derivative gain scales the derivative of the position error, which is the difference between the instantaneous trajectory position and the primary feedback position. Velocity feedback is estimated through a combination of speed-dependent algorithms.

Vff (Velocity Feedforward)
The velocity feedforward gain (Vff) determines the contribution in the command output that is directly proportional to the instantaneous trajectory velocity. This value is used to minimize following error during the constant velocity portion of a move and can be changed at any time to tune the PID loop. Velocity feedforward is an open-loop compensation technique and cannot affect the stability of the system. However, if you use too large a value for Vff, following error can reverse during the constant velocity portion, thus degrading performance, rather than improving it. Velocity feedforward is typically used when operating in PIVff mode with either a velocity block amplifier or substantial amount of velocity feedback. In these cases, the uncompensated following error is directly proportional to the desired velocity.

Aff (Acceleration Feedforward)
The acceleration feedforward gain (Aff) determines the contribution in the command output that is directly proportional to the instantaneous trajectory acceleration. Aff is used to minimize following error (position error) during acceleration and deceleration and can be changed at any time to tune the PID loop. Acceleration feedforward is an open-loop compensation technique and cannot affect the stability of the system. However, if you use too large a value of Aff, following error can reverse during acceleration and deceleration, thus degrading performance, rather than improving it.

Dual-Loop Feedback
Motion control systems often use gears to increase output torque, increase resolution, or convert rotary motion to linear motion. The main disadvantage of using gears is the backlash created between the motor and the load. This backlash can cause a loss of position accuracy and system instability. The control loop on the motion system corrects for errors and maintains tight control over the trajectory. The control loop consists of three main parts—proportional, integral, and derivative—known as PID parameters. The derivative part estimates motor velocity by differentiating the following error (position error) signal. This velocity signal adds, to the loop, damping and stability. If backlash is present between the motor and the position sensor, the positions of the motor and the sensor are no longer the same. This difference causes the derived velocity to become ineffective for loop damping purposes, which creates inaccuracy in position and system instability.

Using two position sensors for an axis can help solve the problems caused by backlash. As shown in Figure 15.27, one position sensor resides on the load and the other on the motor before the gears. The motor sensor is used to generate the required damping and the load sensor for position feedback. The mix of these two signals provides the correct position feedback with damping and stability.

15.3.3.3 Commutation

When a current-carrying conductor is placed in a magnetic field, it experiences a force. Motors are based on this principle of physics. Motors comprise of permanent magnets and current-carrying conductors in a mechanical assembly. Maintaining the correct direction of current and magnetic fields is critical to proper operation of the motor. This functionality is called commutation.

In case of brushed DC servo motors, a physical brush performs commutation by switching the current between different conductors. However, in case of brushless servo motors, three phases of current are

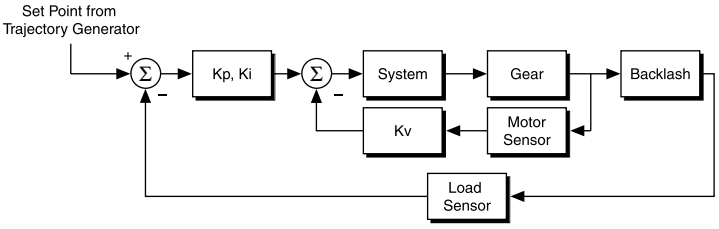


FIGURE 15.27 Dual-loop feedback.

electrically commutated. This requires commutation to be performed either by the motion controller or the drive.

15.3.3.4 Current Loop (Torque Loop)

A current loop takes a set point from the position and velocity loop and uses feedback from a current sensor to generate a voltage output signal. This voltage is used to control the power transistors in the drive to ensure that the right amount of torque is being generated. While in brushed DC servo motors the current command is directly proportional to the current in one winding in the motor, in brushless servo motors it is dependent on current in three windings. As a result, brushless servo motors often come paired with drives that include commutation and a pretuned current loop.

15.4 Motion Controller Hardware

A motion controller can be implemented on dedicated hardware connected to an analog drive, independent of the PC processor that hosts the user application. It can also be designed to share resources on the PC processor with the user application with some functions being executed within a digital drive, thus eliminating the need for dedicated motion controller hardware. These two architectures for implementing a motion controller have their advantages and disadvantages.

15.4.1 Architecture of an Analog Motion Controller (Dedicated Processor)

A motion controller is responsible for executing supervisory control, trajectory generation, and control-loop functions. In case of an analog motion controller, these functions are executed on dedicated hardware that communicates with the host PC via an interface such as the PCI bus, USB, Ethernet, Serial RS 232, and so forth. Commands from the user application running on the host computer are sent to the motion controller via this interface. The motion controller performs supervisory control, trajectory generation, and I/O synchronization, and executes position and velocity control loops. The motion controller then sends commands in the form of 10V analog voltage signals to an analog drive that closes the current loop. There are four main hardware components in a typical analog motion controller.

15.4.1.1 Microprocessor

A microprocessor runs amultitasking, real-time operating system, and handles host communication with the PC and supervisory control including command processing and fault handling.

15.4.1.2 Digital Signal Processor

A digital signal processor is used for math-intensive operations such as trajectory generation and position and velocity loops.

In some motion controllers, all the supervisory control, trajectory generation, and control-loop functions are performed on a single microcontroller that includes I/O interfaces, memory, and clock in addition to the CPU in an integrated circuit.

15.4.1.3 Field-Programmable Gate Array

Afield-programmable gate array (FPGA) is used to perform I/O synchronization functions such as encoder decoding, position capture and position compare, motion I/O processing, and stepper pulse generation.

15.4.1.4 Motion I/O

An analog motion controller connects to different types of motors as follows:

1. Brushed DC servo motor—An analog motion controller provides a ± 10 V analog output signal to a servo drive that closes the current loop to move a brushed DC servo motor.
2. Brushless servo motor—An analog motion controller provides a ± 10 V analog output signal to a servo drive that performs sinusoidal commutation and closes the current loop to move a brushless

servo motor. In case the drive does not perform sinusoidal commutation, the analog motion controller provides two analog output signals to the drive. The motion controller in this case performs sinusoidal commutation in addition to closing the position and velocity loops.

3. Stepper motor—An analog motion controller provides step and direction signals to a stepper drive to move a stepper motor. The motion controller performs step generation to output step and direction signals.

15.4.2 Architecture of a SoftMotion-Based Motion Controller (Shared Processor with User Application)

In case of a soft motion-based motion controller, functions such as supervisory control and trajectory generation execute on a real-time operating system shared with the operating system used for the user interface on the same processor. Control loops are executed within the drive, thus eliminating the need for dedicated motion controller hardware. Some configurations allow for the control loops to be also closed on the shared processor.

Common operating systems that are used for soft motion-based motion controllers include off-the shelf RTOSes such as QNX, Pharlap ETS, RTX, VxWorks, or Windows CE or, in many cases, proprietary implementations by motion control vendors.

15.4.2.1 PC/PLC-Based with Analog Interface (Motion I/O Connected to the PC/PLC via PCI or ISA Bus)

A PLC-based SoftMotion controller runs on the real-time operating system on the PLC. It connects to analog drives with a ± 10 V analog interface. In this case, the motion controller performs supervisory control, trajectory generation on the PLC processor, and closes position and velocity control loops on the PLC motion control modules. These modules connect to analog drives that handle the task of commutation, closing the current loops. In this case, the motion controller tasks run at the highest priority on the PLC processor and coexist with the user's sequential logic tasks.

A PC-based SoftMotion controller runs on a real-time operating system (VxWin, VxCE, InTime) or real-time extension (RTX, TwinCAT) that runs on a PC and shares the processor with Windows or any desktop OS. In this case, the motion controller performs supervisory control, trajectory generation on the PC processor, and closes position and velocity control loops on the PCI/ISA-based motion control card plugged into the PC. These cards connect to analog drives that handle the task of commutation, closing the current loops. In this case, since the motion controller runs on the real-time OS on the PC, it is guaranteed bandwidth and priority over Windows or any desktop OS sharing the processor with it.

15.4.2.2 PC/PLC-Based with Deterministic Digital Network (Distributed Motion, I/O Connected to the PC/PLC via a Deterministic Network)

As discussed in Section 15.4.2.1, a SoftMotion controller shares the processor with either the PLC tasks or tasks running on the Windows or desktop OS on a PC. The primary difference between this mode of operation and the former is that in this mode the communication of the motion controller with the drives is via a deterministic bus. This solves a number of issues faced in Section 15.4.2.1 such as connectivity and signal integrity. This architecture is also termed as "DistributedMotion." Similar to the previous case, the motion controller performs supervisory control, trajectory generation, and in some cases also closes the position and velocity control loops. The drives close handle the commutation and close the current loops. They also support closing the position and the velocity loops that allow for lower bandwidth utilization of the deterministic network.

A primary requirement for such a configuration is that the digital network connecting the motion controller to the digital drive is deterministic and that it is able to transmit data deterministically. Emerging networks include IEEE 1394, Ethernet PowerLink, EtherCAT, SERCOS III, PROFInet (IRT), EtherNET/IP, CANopen, and PROFIbus (MC).

IEEE 1394
IEEE 1394 (isochronous) (www.1394ta.org) is a high speed, real-time digital network based on the serial bus. It requires an IEEE 1394 interface on the host PC. It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

Ethernet PowerLink
Ethernet PowerLink (www.ethernet-powerlink.org) is an Ethernet-based digital network based on TCP/IP and isochronous slot communication network management (SCNM). It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

EtherCAT
EtherCAT (www.ethercat.org) is an Ethernet-based digital network based on the principle of reading and writing EtherCAT telegram packets as they pass through the different slave devices in cyclic order. It offers submillisecond response times and submicrosecond jitter for use with soft motion-based motion controllers.

SERCOS III
SERCOS-III (www.sercos.com) is an Ethernet-based digital network based on TCP/IP and the principle of cyclic data transfer with an exact time pattern. It requires the use of a master to control the timing behavior of the data transfer to the devices on the network. It provides submillisecond response rates for use with soft motion-based motion controllers.

PROFInet (IRT)
PROFInet (isochronous real time) (www.profinet.com) is an Ethernet-based digital network that leverages TCP/IP and DCOM standards, and works with internal switches in all devices to handle synchronization. It provides submillisecond response rates for use with soft motion-based motion controllers.

EtherNET/IP
EtherNET/IP (www.odva.org) is an Ethernet-based digital network that is based on TCP/IP and Control switches and uses time stamps for synchronization. It provides clock-cycle synchronous data transmission, and Information Protocol (CIP) standards. It supports a star topologywith devices connected via managed and millisecond response rates for use with soft motion-based motion controllers that performsupervisory control and trajectory generation.

CANopen
CANopen (www.canopen.de) is a layer on CAN that is based on the DS301 communication profile and the DS402 device profile for motion control. It supports deterministic data transmission, and millisecond response rates for use with soft motion-based motion controllers that perform supervisory control and trajectory generation. The control loops are closed on the digital drive.

PROFIbus (MC)
PROFIbus (Motion Control) (www.profibus.org) is based on the PROFIbus digital network with enhancements to response times and clock synchronization. PROFIbus (MC) has response rates of several milliseconds making it suitable for soft motion-based motion controllers that perform supervisory control and trajectory generation. The control loops are closed on the digital drive.

15.5 Summary

Motion control systems are used to control the position, velocity, and torque of a rotational or linear electromechanical device. A typical motion control system consists of a variety of components including user interface, motion controller, drive, motor, feedback device, and motion I/O.The functions of a motion controller include trajectory generation, supervisory control, and a position and velocity feedback control loop.

16

Real-Time Monitoring and Control

Gerardo Garcia
National Instruments, Inc.

16.1 Introduction to Real-Time Systems

Real-time operating systems originated with the need to solve two main types of applications: event response and closed-loop control systems. Event response applications require a response to a stimulus in a determined amount of time; an example of such a system is an automotive airbag system. Closed-loop control systems continuously process feedback in order to adjust an output; an automotive cruise control system is an example of a closed-loop control system. Both of these types of systems require the completion of an operation within a specific deadline. This type of performance is referred to as determinism.

Real-time systems are sometimes classified as "soft" or "hard." Soft real-time typically means the utility of a system is inversely proportionate to how long it has been since a deadline is missed. For example, when pressing a cell phone button to answer an incoming call, the connection must be established soon after the button has been pressed. However, the deadline is not mission-critical and small delays can be tolerated. Hard real-time systems are those in which the utility of a system becomes zero in the event of a missed deadline. An automotive engine control unit (ECU) must process incoming signals and calculate spark plug timing within a deadline. If the deadline is missed, the engine will fail to operate correctly. The usefulness of a task after a deadline is missed depends on whether the system is a soft real-time or a hard real-time system, as shown in Figure 16.1.

Operating systems such as Microsoft Windows and Mac OS provide an excellent platform for developing and running your noncritical measurement and control applications. However, because these operating

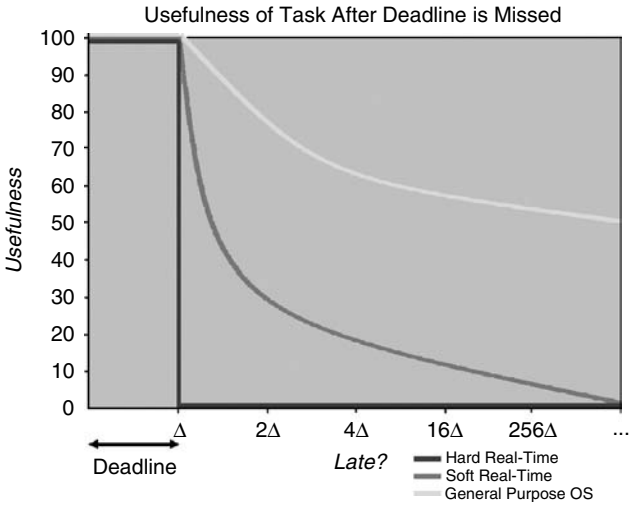


FIGURE 16.1 Differences between hard and soft real-time.

systems are designed for general purpose use, they are not the ideal platforms for running applications that require deterministic performance or extended uptime.

General-purpose operating systems are optimized to run a variety of applications simultaneously, ensuring that all applications receive some processing time. These operating systems must also respond to interrupts from peripherals such as the mouse and keyboard. The user has limited control regarding how these tasks are handled by the processor. As a result, high-priority tasks can be preempted by lower priority tasks, making it impossible to guarantee a response time for your critical applications.

In contrast, real-time operating systems give users the ability to prioritize tasks so that the most critical task can always take control of the processor when needed. This property enables you to program an application with predictable results.

Real-time operating systems are required when the processor is involved in operations such as closed loop control and time-critical decision making. These applications require timely decisions to be made on the basis of incoming data. For example, an I/O device samples an input signal and sends it directly to memory. Then, the processor must analyze the signal and send the appropriate response to the I/O device. In this application, the software must be involved in the loop; therefore, you need a real-time operating system to guarantee response within a fixed amount of time. In addition, applications requiring extended run-times or stand-alone operation are often implemented with real-time operating systems.

16.2 Real-Time Development Tools

Traditional real-time software development requires a code editor to develop code, a compiler to generate embedded code, and a compatible debugger to validate the application, as well as tools to profile the application performance, track memory usage, and trace system thread execution. Many software development packages integrate these tools into one environment. An example of such an environment is the LabVIEW Real-Time Development Module. Figure 16.2 depicts the real-time development environment, including the host computer for software development and the real-time target for execution.

System profiling tools are important for optimizing the execution of a real-time system. The LabVIEW Execution Trace Toolkit is an example of a tracing tool that captures low-level execution details of a realtime application. Figure 16.3 shows a typical trace capture that displays when application tasks executed in time.

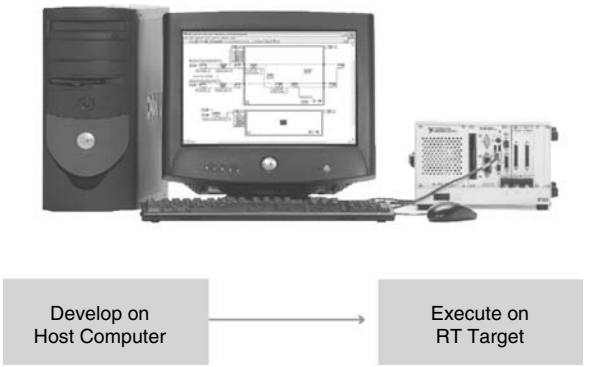


FIGURE 16.2 Real-time development process.

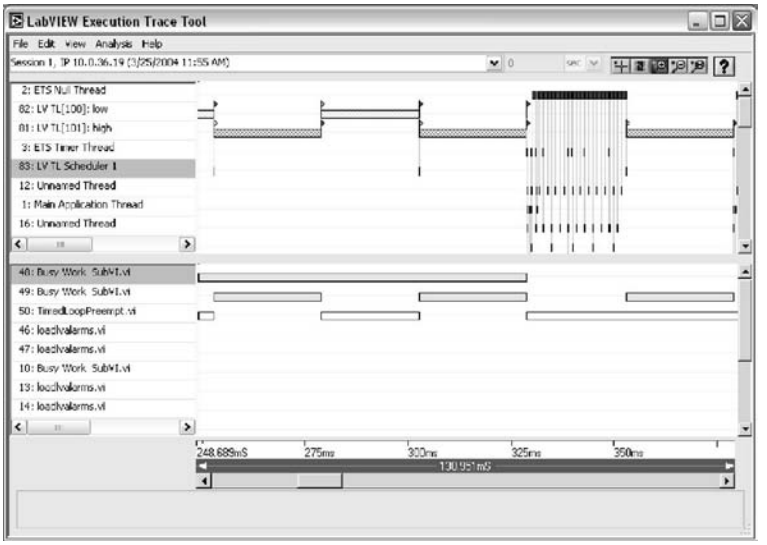


FIGURE 16.3 Application tracing tool.

Hardware deployment platforms vary depending on the application. Real-time applications use desktop PCs, modular hardware platforms such as PXI, off-the-shelf rugged controllers such as programmable logic controllers (PLCs) and programmable automation controllers (PACs), single-board computers, and embedded microprocessors on custom hardware. The popular NI CompactRIO PAC is shown in Figure 16.4. An appropriate real-time operating system is used in these hardware platforms to execute the application software.

16.2.1 Operating System Scheduler

All real-time operating systems have a scheduler that coordinates the execution of tasks (threads) by the CPU. There are various scheduling methods that are used by operating systems. Round-robin scheduling shares processor time between threads based on equal slices of time.

To illustrate round-robin scheduling, we consider the analogy of an automobile repair shop. The mechanic represents the CPU, the shop manager represents the scheduler, and multiple cars represent the multiple threads of our system. Using round-robin scheduling, the mechanic will cycle between each car for a set period of time. Round-robin scheduling guarantees that each thread will have some time with the processor.

FIGURE 16.4 NI CompactRIO programmable automation controller.

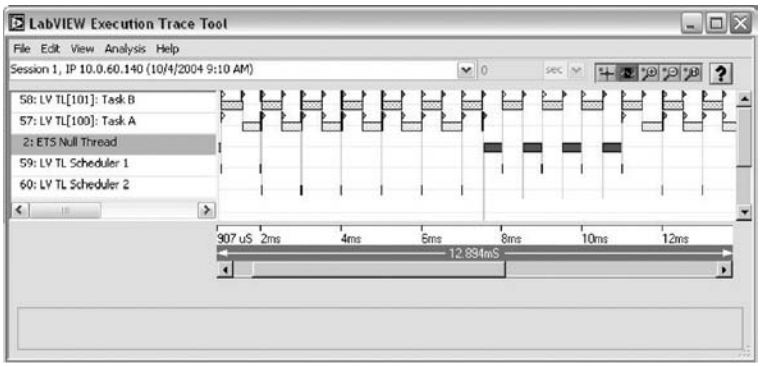


FIGURE 16.5 Example trace of preemptive scheduling.

Another method of scheduling is called preemptive scheduling, where priority can be given to the time-critical tasks.When a time-critical thread needs time from the processor, the other threads will wait until the time-critical thread is finished. In the repair shop example, an ambulance is assigned time-critical priority. As a result, it will be serviced as soon as it arrives. All other cars will be put on hold until the ambulance service is complete. Once the ambulance service is complete, the other cars will continue to share time with the mechanic. Figure 16.5 shows an example of two tasks running at different priority levels. Task B has a higher priority and runs every half millisecond, while Task A runs every 10 ms. The red flags indicate when Task A is preempted so that the higher priority Task B can be executed.

16.3 Real-Time Software Architecture

When developing a real-time application, you must decide which operations need to have real-time performance and which operations do not. The components that need real-time performance are those that need to occur within a specific period of time, meaning they are deterministic. These time-critical tasks should be assigned the highest priority in the application.

Operations that are of lower priority will run only when the time-critical operations are suspended. Examples of low priority operations are communication with a host computer, reading and writing to files, and nondeterministic communication with external instruments. These operations are sometimes called the house-keeping tasks.

16.4 Deterministic Timing

Determinism is how consistently a systemis able to perform operations within a fixed period of time. This is the most fundamental component of all real-time systems. Jitter is the variance between the expected execution time of a real-time task and the actual execution time. All real-time systems have some jitter; however, the jitter is bound and can be quantified. Systems that are not real time have very large or unbounded jitter. Figure 16.6 depicts a jitter diagram.

The most basic source of jitter is variation in the clock. However, you can induce additional jitter if you do not follow certain programming practices. To reduce jitter, tasks are programmed to run periodically on the basis of a hardware clock that can be derived from an I/O signal or a precise CPU timer. Figure 16.7 shows the configuration of a real-time task set to run every 500 ms.

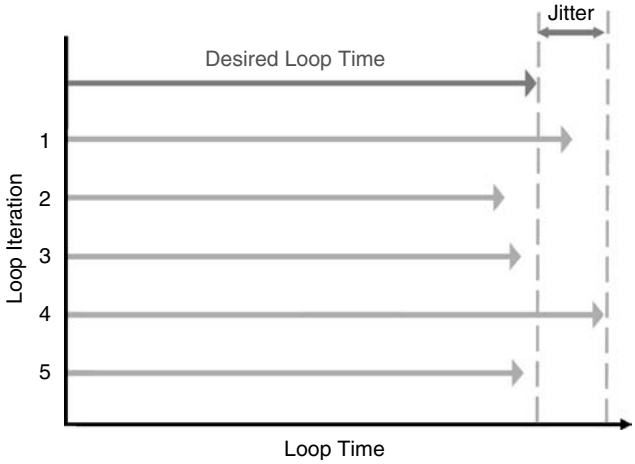


FIGURE 16.6 A jitter diagram.

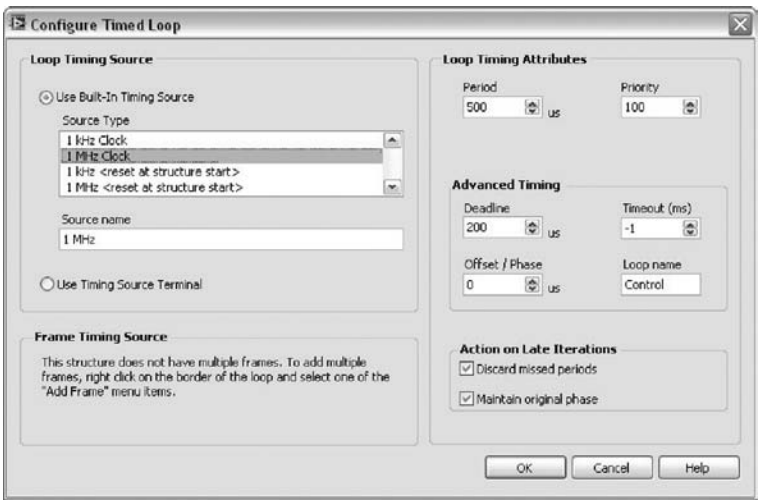


FIGURE 16.7 Real-time task configuration.

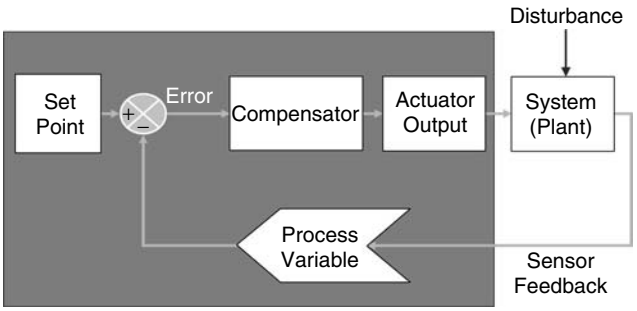


FIGURE 16.8 The block diagram of a closed-loop control system.

16.5 Implementing Real-Time Control

Control algorithm implementation is an important part of real-time system development. Control strategies can vary in complexity, from simple digital logic to advanced mathematical models that are carefully tuned to control dynamic systems. A typical block diagram of a closed-loop control system is shown in Figure 16.8.

16.5.1 Proportional-Integral-Derivative Control

The proportional-integral-derivative (PID) algorithm is the most common control algorithm used in industry. With PID, you specify a process variable and a set point. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the set point is the desired value for the parameter you are controlling.

The proportional gain determines the ratio of output response to the error. The error is the difference between the set point and the process variable. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate.

The integral component sums the error term over time. The integral response will continually increase over time unless the error is zero, so the effect is to drive the steady state error to zero.

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable.Most practical control systems use very small derivative time, because the derivative response is highly sensitive to noise in the process variable signal.

16.5.2 Fuzzy Logic Control

Fuzzy logic control is most often used for expert systems and process control. It is a method of rule-based decision making that emulates the rule of thumb thought process of humans. If an expert can qualitatively describe a control strategy, fuzzy logic can be used to develop the controller.

16.5.3 Model-Based Control Design

Another method for designing control systems is by using model-based design methods. This involves using software tools to model the behavior of dynamic systems completely in software. Then, designers can use mathematical software tools to develop control algorithms and test them with the software simulation of the dynamic system. Figure 16.9 shows a simulation diagram created in LabVIEW.

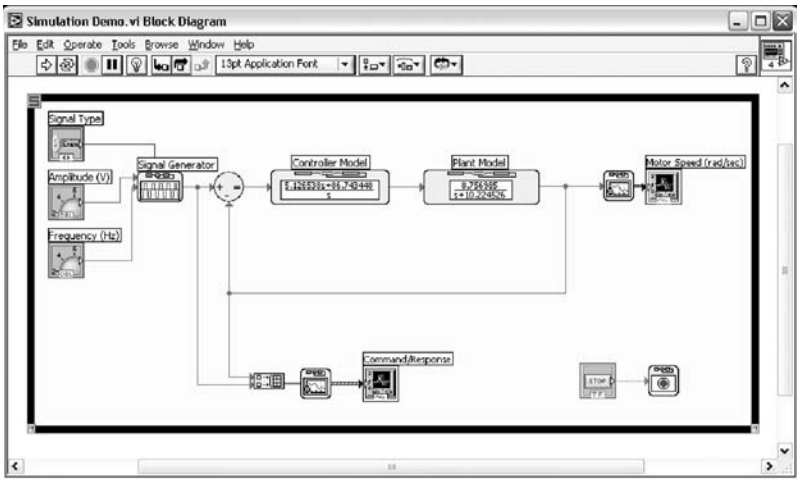


FIGURE 16.9 Simulation diagram.

16.6 Monitoring Systems

Real-time systems are usually monitored by one or more human–machine interfaces (HMI) that send and receive data from the real-time controllers. These HMIs can be in the form of powerful desktop servers, laptops, PDAs, web browser interfaces, or touch-panel displays. Communication is done through a variety of media and protocols. Ethernet is widely used as a media of data transfer with differing protocols depending on the application. These include TCP/IP for point-to-point streaming, OLE for Process Control (OPC) for distributed control systems, or web servers for remote viewing. Other methods for communicating are serial interfaces, fieldbuses such as PROFIBUS and DeviceNet, and wireless media such as WIFI and Bluetooth.

Large distributed real-time systems typically use a Supervisory Control and Data Acquisition (SCADA) system for monitoring. These systems have visualization features and can display data from real-time controllers, show alarm conditions, and log data to databases. An example of a SCADA software is the LabVIEW Datalogging and Supervisory Control (DSC) Module. A SCADA visualization of a heat exchanger is shown in Figure 16.10.

For distributed real-time control systems, data can be stored on multiple computers and monitored centrally, or it can be stored on one central server. The most difficult challenge is to communicate with live data. OPC is an industry standard interface through which software and hardware can communicate irrespective of the manufacturer. Controller hardwares from different manufacturers communicate with their own OPC server on a host system and publish I/O and calculated values as tags. SCADA systems act as OPC clients that subscribe to tags from all the individual OPC servers. Figure 16.11 shows a LabVIEW project with a list of tags that can be read from OPC servers to develop an operator interface.

For some applications, OPC data transfer may not be fast enough. In these cases, using lower-level protocols such as TCP/IP and UDP allows you to send data at faster rates.

16.7 Summary

In summary, real-time systems solve a variety of applications that require determinism and reliability. To achieve such requirements, real-time operating systems define strict scheduling behavior, which all

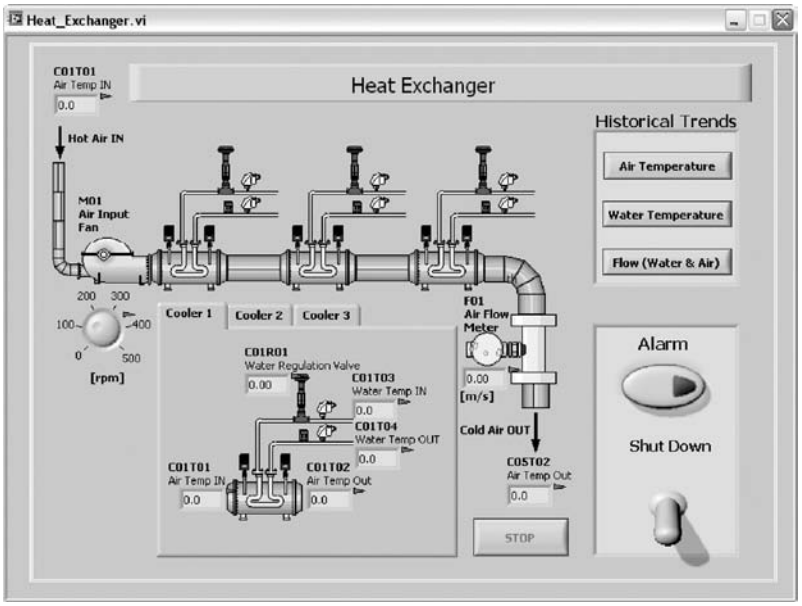


FIGURE 16.10 SCADA visualization.

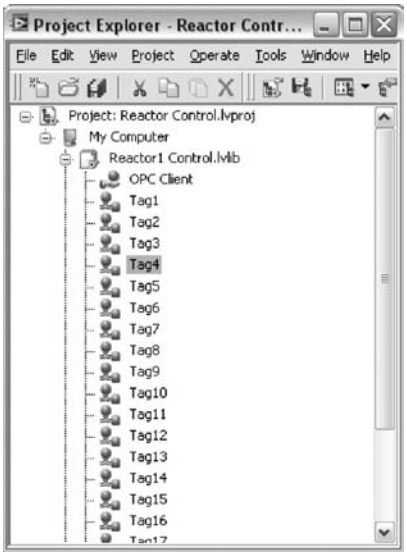


FIGURE 16.11 OPC tags.

software abides by. Due to the complex nature of real-time applications and the small margin for error, developers must employ good programming practices in order to be successful.

Further Information

Web sites

1. http://www.omimo.be/encyc/publications/faq/rtfaq.htm
2. http://www.ni.com/realtime

3. http://zone.ni.com/devzone/cda/tut/p/id/3938
4. http://zone.ni.com/devzone/cda/tut/p/id/4040

References

1. J.E. Cooling, *Software Design for Real-time Systems*, ISBN 0-412-34180-8, Chapman & Hall, London, 1991.
2. Stuart Bennett, *Real-Time Computer Control*, ISBN 0-13-764176-1, Prentice Hall, Upper Saddle River, NJ 2006.

17

Micromechatronics and Microelectromechanical Motion Devices*

Sergey Edward Lyshevski
Rochester Institute of Technology

17.1 Micromechatronic Systems Design

Micromechatronic systems integrate actuators, sensors, power electronics, and ICs. Within a focus on high-performance micromechatronic systems, books [1–3] cover the general issues in the systems design. Important topics and issues are outlined and covered in this section. One of the major components of micromechatronic systems is a high-performance microelectromechanical motion device that (1) converts physical stimuli to electrical or mechanical signals and vice versa, and, (2) performs actuation and sensing. These motion devices are controlled by ICs. Electromagnetic and electromechanical features of microelectromechanical motion devices are basics of their operation, design, analysis, and fabrication. Correspondingly, motion devices and ICs are designed taking into account possible system architectures [1–3]. The baseline step-by-step procedure in the design of micromechatronic systems is

1. Define application and environmental requirements
2. Specify performance specifications

*Some results reported in this chapter were published in [1–3].

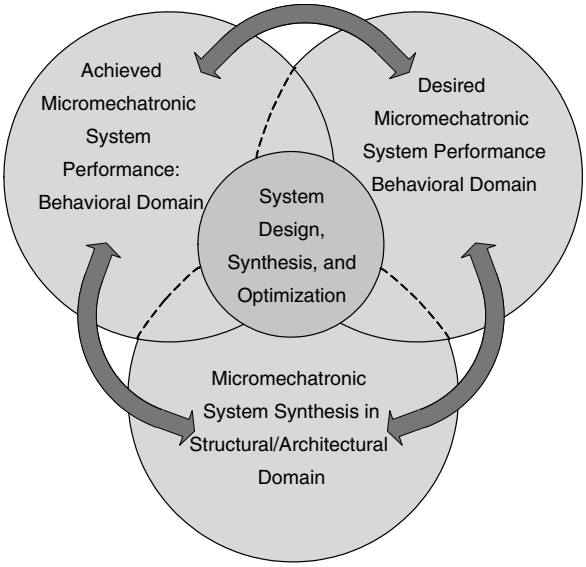


FIGURE 17.1 Design flow in synthesis of micromechatronic systems.

3. Devise (synthesize) microelectromechanical motion devices (actuators and sensors) researching operating principles, topologies, configurations, geometry, electromagnetic systems, and so forth
4. Perform electromagnetic, mechanical, and sizing-dimension estimates
5. Define technologies, techniques, processes, and materials to fabricate electromechanical motion devices
6. Design ICs to control electromechanical motion devices
7. Develop high-fidelity mathematical modelswith minimumlevel of simplifications and assumptions to examine integrated electromagnetic–mechanical phenomena and effects
8. Perform coherent electromagnetic and mechanical analysis
9. Modify and refine the design optimizing performance
10. Design control laws to control motion devices and implement these controllers using ICs
11. Integrate micromechatronic system

One of the most challenging design problems is the system architecture synthesis, system integration, optimization, and hardware (actuators, sensors, power electronics, ICs, microcontrollers, and DSPs) selection. The design starts with a given set of requirements and specifications. High-level functional design is performed first in order to produce detailed design at the subsystem and component level. Using the advanced subsystems and components, the initial design is performed, and the closed-loop electromechanical system performance is tested against the requirements. At each level of the design hierarchy, the system performance in the behavioral domain is used to evaluate and refine the design. The design flow is illustrated in Figure 17.1.

The micromechatronic system is illustrated in Figure 17.2. The systemperformance is measured against many criteria, for example, stability, robustness, transient behavior, accuracy, disturbance attenuation, and so forth. In the behavioral domain, the designer can examine and optimize the input–output transient dynamics using the tracking error $e(t) = y(t) - r(t)$, where $y(t)$ and $r(t)$ are the output and reference (command) variables. The tracking error $e(t)$ is minimized and dynamics is optimized using different performance criteria. For example,

$$J = \min_{t,e} \int_0^\infty t\,|\,e\,|\,\mathrm{d}t, \quad J = \min_{e} \int_0^\infty |\,e\,|\,\mathrm{d}t, \quad \text{or} \quad J = \min_{e} \int_0^\infty e^2 \mathrm{d}t.$$

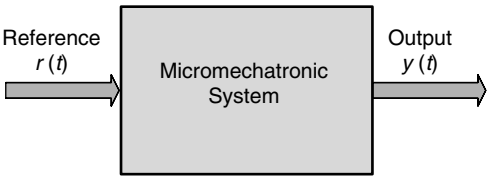


FIGURE 17.2 Dynamic micromechatronic system.

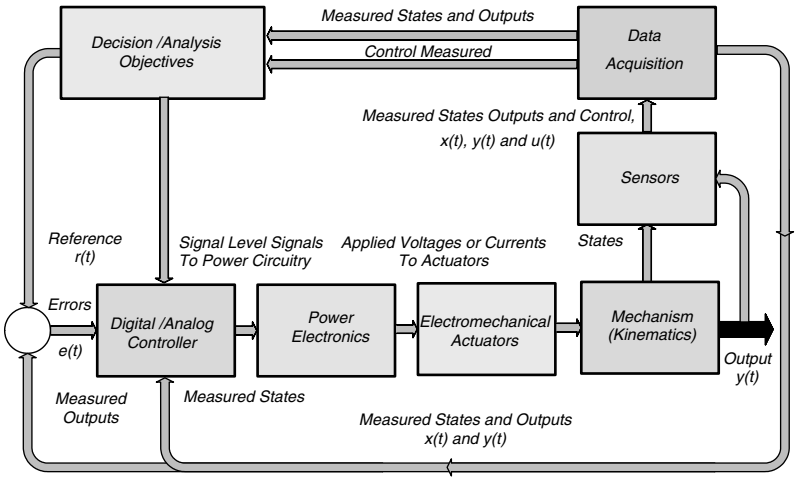


FIGURE 17.3 High-level functional block diagram of the closed-loop micromechatronic system.

Mechatronic systems (robots, electric drives, servomechanisms, pointing systems, assembles, etc.) are actuated by electromechanical motion devices that are highly nonlinear systems. Therefore, accurate actuator actuation and control are very important problems. Depending on operation and functionality of motion devices, onemust derive the expressions for the voltages applied to the phase windings in order to maximize the electromagnetic torque and ensure optimality. Actuators and sensors must be designed and integrated with the corresponding power electronics and ICs. The control laws are implemented using analog controllers, which can be integrated with power electronics (PWM amplifiers), as well as digital controllers (microcontrollers and DSPs). The principles of matching and compliance are general design principles that require that the system architectures should be synthesized integrating all subsystems and components. The matching conditions have to be determined and guaranteed, and actuators— sensors—power electronics—ICs—controller compliance must be ensured.

A functional block diagram of a controlled micromechatronic system is illustrated in Figure 17.3. Different techniques to design optimal controllers are reported in the next section. Those controllers should be derived using the baseline electromagnetic laws that define the operating principles of electromechanical motion devices. It was emphasized that depending on the motion devices, distinct power electronics must be utilized. High-performance micromechatronic systems can be designed using permanent-magnet electromechanical motion devices that ensure high torque and power densities. Radial and axial technology permanent-magnet motion devices are covered in the following sections.

17.2 Tracking Control of Micromechatronic Systems

In general, mechatronic systems are nonlinear [1–3]. Using the Hamilton–Jacobi theory and Lyapunov concepts, design of control algorithms for nonlinear systems is reported in References 1–3. To introduce

the design, we consider linear and nonlinear models. A linear model is given as

$$x^{sys}(t) = Ax^{sys} + Bu \qquad y = Hx^{sys} \qquad x^{sys}(t_0) = x_0^{sys}, \tag{17.1}$$

where $x^{sys} \in X^{sys} \subset \mathbb{R}^n$ is the state vector with auxiliary conditions; $u \in U \subset \mathbb{R}^m$ is the control vector; $y \in Y \subset \mathbb{R}^b$ is the output vector; and $A^{sys} \in \mathbb{R}^{n\times n}$ and $B^{sys} \in \mathbb{R}^{n\times m}$ are the constant-coefficient matrices. Define the tracking error vector as

$$e(t) = Nr(t) - y(t) = Nr(t) - Hx^{sys}(t). \tag{17.2}$$

Therefore, from Equations 17.1 and 17.2, one finds

$$\dot{e}(t) = N\dot{r}(t) - \dot{y}(t) = N\dot{r}(t) - H\dot{x}^{sys}(t) = N\dot{r}(t) - HA^{sys}x^{sys} - HB^{sys}u. \tag{17.3}$$

Using the expanded state vector $x(t) = \begin{bmatrix} x^{sys}(t) \\ e(t) \end{bmatrix}$, we have

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} \dot{x}^{sys}(t) \\ \dot{e}(t) \end{bmatrix} = \begin{bmatrix} A^{sys} & 0 \\ -HA^{sys} & 0 \end{bmatrix}\begin{bmatrix} x^{sys} \\ e \end{bmatrix} + \begin{bmatrix} B^{sys} \\ -HB^{sys} \end{bmatrix} u + \begin{bmatrix} 0 \\ N \end{bmatrix}\dot{r} \\ &= Ax + Bu + \begin{bmatrix} 0 \\ N \end{bmatrix}\dot{r} \quad y = Hx^{sys}. \end{aligned} \tag{17.4}$$

The *space transformation* method is based on the use of the following z and v vectors: $z = \begin{bmatrix} x \\ u \end{bmatrix}$ and $v = \dot{u}$. Thus, from Equation 17.4, we obtain the system

$$\dot{z}(t) = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} z + \begin{bmatrix} 0 \\ I \end{bmatrix} v = A_z z + B_z v, \quad y = Hx^{sys} \qquad z(t_0) = z_0. \tag{17.5}$$

Minimizing the quadratic performance functional

$$J = \int_{t_0}^{t_f} \left(z^T Q_z z + v^T G_z v \right) dt, \; Q_z \in \mathbb{R}^{(n+m)\times(n+m)}, \; Q_z \geq 0, \; G \in \mathbb{R}^{m\times m} \quad G > 0, \tag{17.6}$$

with respect to the system dynamics (Equation 17.5), the application of the first-order necessary condition for optimality gives

$$v = -G_z^{-1} B_z^T K z. \tag{17.7}$$

The Riccati equation

$$-\dot{K} = KA_z + A_z^T K - KB_z G_z^{-1} B_z^T K + Q_z, \; K(t_f) = K_f \tag{17.8}$$

is solved to find the unknown matrix $K \in \mathbb{R}^{(n+m)\times(n+m)}$. Taking note of Equation 17.7, one has

$$\begin{aligned} \dot{u}(t) &= -G_z^{-1} B_z^T K z = -G_z^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}^T \begin{bmatrix} K_{11} & K_{21}^T \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \\ &= -G_z^{-1} K_{21} x - G_z^{-1} K_{22} u = K_{f1} x + K_{f2} u. \end{aligned} \tag{17.9}$$

Using $\dot{x}(t) = Ax + Bu$, wehave $u = B^{-1}(\dot{x}(t) - Ax) = (\mathrm{BTB})^{-1}\,\mathrm{BT}\,(\dot{x}(t) - Ax)$. From Equation 17.9, one obtains

$$\begin{aligned}\dot{u}(t) &= K_{f1}x + K_{f2}u = K_{f1}x + K_{f2}\left(B^T B\right)^{-1} B^T(\dot{x}(t) - Ax)\\ &= \left[K_{f1} - K_{f2}\left(B^T B\right)^{-1} B^T A\right]x(t) + K_{f2}\left(B^T B\right)^{-1} B^T \dot{x}(t)\\ &= \left(K_{f1} - K_{F1}A\right)x(t) + K_{F1}\dot{x}(t) = K_{F2}x(t) + K_{F1}\dot{x}(t).\end{aligned} \tag{17.10}$$

Using Equation 17.10, and taking note of $x(t) = \begin{bmatrix} x^{\mathrm{sys}}(t) \\ e(t) \end{bmatrix}$, one finally derives the proportional-integral controller in the following form

$$\begin{aligned}u(t) &= K_{F1}x(t) - K_{F1}x_0 + \int K_{F2}x(\tau)d\tau + u_0\\ &= K_{F1}\begin{bmatrix} x^{\mathrm{sys}}(t) \\ e(t) \end{bmatrix} - K_{F1}\begin{bmatrix} x_0^{\mathrm{sys}}(t) \\ e_0(t) \end{bmatrix} + \int K_{F2}\begin{bmatrix} x^{\mathrm{sys}}(\tau) \\ e(\tau) \end{bmatrix} d\tau + u_0.\end{aligned} \tag{17.11}$$

For nonlinear micromechatronic systems, as given by

$$\dot{x}^{\mathrm{sys}}(t) = F(x^{\mathrm{sys}}) + B(x^{\mathrm{sys}})u, \qquad y = Hx^{\mathrm{sys}}, \qquad x^{\mathrm{sys}}(t_0) = x_0^{\mathrm{sys}}, \tag{17.12}$$

the proposed procedure can be straightforwardly used. In Equation 17.12), $F(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{n}}$ and $B(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{n\times m}}$ are the smooth Lipschitz maps; $H \in \mathbb{R}^{\mathrm{b\times c}}$ is the output matrix with constant coefficients.

The proportional-integral controller for Equation 17.12 is given as

$$\dot{u}(t) = -G_z^{-1}B_z^T\frac{\partial V}{\partial z} = -G_z^{-1}\begin{bmatrix} 0 \\ I \end{bmatrix}^T \frac{\partial V(x,u)}{\partial [x\ u]^T}, \tag{17.13}$$

where $V(x, u)$ is the return function.

The control law is bounded, and $u_{\min} \le u \ge u_{\max}$. To design the *admissible* control laws, we minimize

$$J = \int_{t_0}^{t_f}\left[z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v\right]\mathrm{d}t, \tag{17.14}$$

where $\Phi(\cdot)$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}}$ is the bounded, integrable, one-to-one, real-analytic, globally Lipschitz continuous function, $\Phi \in U \subset \mathbb{R}^{\mathrm{m}}$.

Minimizing Equation 17.14 with $u_{\min} \le u \ge u_{\max}$, $u \nmid U$, for linear and nonlinear systems, we have

$$-\frac{\partial V}{\partial t} = \min_{u\in U}\left\{z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v + \frac{\partial V}{\partial z}^T (A_z z + B_z v)\right\}, \tag{17.15}$$

$$\text{and } -\frac{\partial V}{\partial t} = \min_{u\in U}\left\{z^T Q_z z + \int \left(\Phi^{-1}(v)\right)^T G_z \mathrm{d}v + \frac{\partial V}{\partial z}^T [F_z(z) + B_z(z)v]\right\} \tag{17.16}$$

Using the first-order necessary condition for optimality, the minimization of Equations 17.15 and 17.16 results in the *admissible* controllers

$$\dot{u}(t) = -\Phi\left(G_z^{-1}B_z^T\frac{\partial V}{\partial z}\right) \quad u \in U. \tag{17.17}$$

Assuming that the solution of the Hamilton-Jacobi equation is approximated by the quadratic return function, from Equation 17.17, one obtains

$$u(t) = \Phi\left(K_{F1}x(t) + \int K_{F2}x(\tau)d\tau\right) = \Phi\left(K_{F1}\begin{bmatrix} x^{\text{sys}}(t) \\ e(t) \end{bmatrix} + \int K_{F2}\begin{bmatrix} x^{\text{sys}}(\tau) \\ e(\tau) \end{bmatrix} d\tau\right) \quad u \in U. \tag{17.18}$$

The *admissible* control law synthesized is bounded, and the second-order necessary condition for optimality is satisfied. However, the sufficient conditions must be examined. Using the *admissibility* concept, it is demonstrated in References 1–3 that a system with bounded control (Equation 17.18) is stable in $X(X_0, U)$, and robust tracking is guaranteed in the convex and compact set $E(E_0, Y, R)$ if for the reference input $r \in R$ there exists a positive-definite function $V(e,x)$ such that $[\mathrm{d}V(e, x)]/\mathrm{d}t \leq 0$. If the criteria imposed on the Lyapunov pair, $V(e, x) > 0$ and $[\mathrm{d}V(e, x)]/\mathrm{d}t \leq 0$, are guaranteed for all $x_0 \in X_0$, $e_0 \in E_0$, $u \in U$, and $r \in R$, then, the closed-loop system is robustly stable in $X(X_0, U)$, and robust tracking is guaranteed in the convex and compact set $E(E_0, Y, R)$. By computing the derivative of the $V(e, x)$, the unknown feedback gains can be found to satisfy the sufficient conditions for stability.

The designed analog controllers may be implemented using microcontrollers. Therefore, analog controllers are discretized. The feedback gains k_{dp} and k_{di} of the digital control law are found using the proportional, integral, and derivative coefficients of the analog PID controller (k_p and k_i) as well as the sampling period T_s. We have $k_{dp} = k_{\text{p}} \cdot \frac{1}{2}\, k_{\text{di}}$ and $k_{\text{di}} = T_s k_{\text{i}}$. In particular, the controller can be implemented as $u\,(kT_{\text{s}}) = k_{\text{p}} e\,(kT_{\text{s}}) + \frac{1}{2}\, k_{\text{i}} T_{\text{s}} \sum_{i=1}^{k} [e\,((i-1)T_{\text{s}}) + e\,(iT_{\text{s}})]$. The proportional-integral controllers with the state feedbacks are implemented in the similar way, and the state variables are sampled at the specified sampling period T_s.

17.3 Synthesis of Microelectromechanical Motion Devices

The designer synthesizes microelectromechanical motion devices by devising and examining baseline physics and operational principles. Axial and radial topology, ac and dc, electromagnetic and electrostatic, and other actuators are covered in [1–3]. To introduce the synthesis taxonomy, we consider radial topology two-phase permanent-magnet synchronous motion devices as shown in Figure 17.4. The electromagnetic system is *endless*, and different geometries can be utilized. In contrast, the translational (linear) synchronous machines have the *open-ended* electromagnetic system. Thus, the device geometry and electromagnetic systems can be integrated within the synthesis, classification, analysis, design, and optimization taxonomy. In particular, motion devices can have different geometries (plate, spherical, torroidal, conical, cylindrical, and asymmetrical) and electromagnetic systems. Using these distinct features, one can classify motion devices.

The basic types of electromagnetic motion devices are induction, synchronous, while topologies are rotational and translational (linear). That is, devices can be classified using a type classifier as given by $Y = \{y : y \in Y\}$. As illustrated in Table 17.1, electromechanical motion devices are categorized using a geometric classifier (plate *P*, spherical *S*, torroidal *T*, conical *N*, cylindrical *C*, or asymmetrical *A* geometry), and an electromagnetic system classifier (*endless E*, *open-ended O*, or *integrated I)*. The classifier, as documented in Table 17.1, is partitioned into 3 horizontal and 6 vertical strips, and contains 18 sections, each identified by ordered pairs of characters, such as *(E, P)* or *(O, C)*. In each ordered pair, the first entry is a letter chosen from the bounded electromagnetic system set $M = \{E, O, I\}$. The second entry is a letter chosen from the geometric set $G = \{P, S, T, N, C, A\}$. That is, the electromagnetic system—geometric set is

$$M \times G = \{(E, F), (E, S), (E, T), \cdots, (I, N), (I, C), (I, A)\}.$$

In general, we have $M \times G = \{(m, g) : m \in M \text{ and } g \in G\}$.

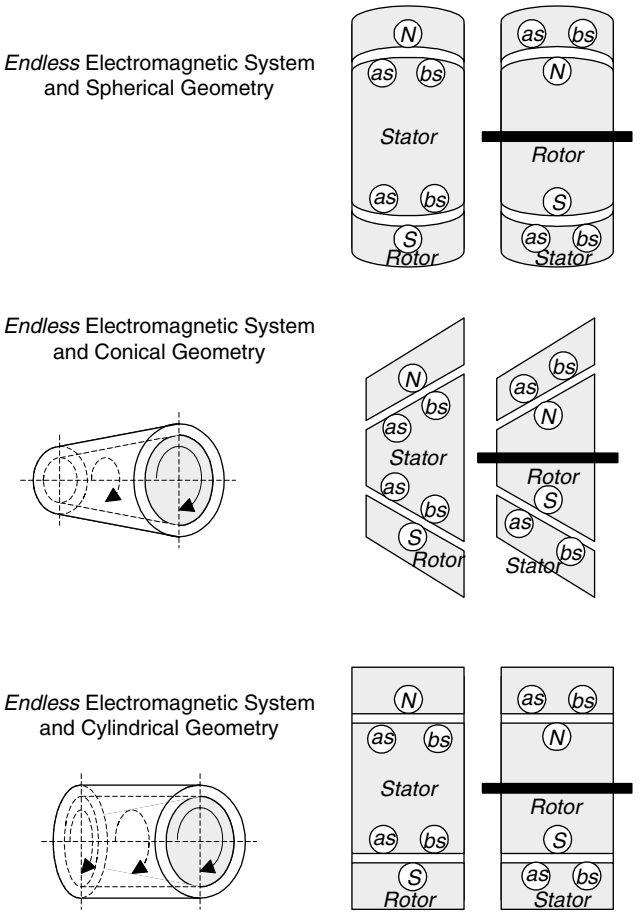


FIGURE 17.4 Radial topology permanent-magnet synchronous motion devices with *endless* electromagnetic system and different geometries.

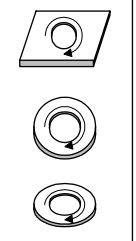
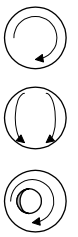
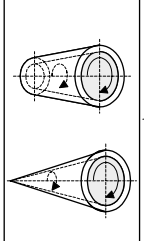
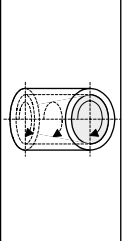
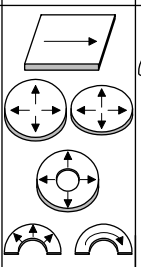
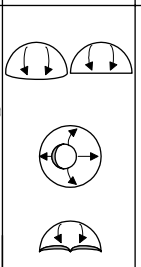
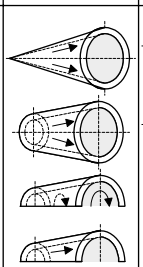
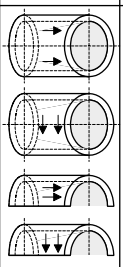
Other categorization can be applied. For example, multiphase motion devices are classified using a phase classifier $H = \{h : h \in H\}$. Therefore, we have

$$Y \times M \times G \times H = \{(y,\ m,\ g,\ h) : y \in Y,\ m \in M,\ g, \in G \text{ and } h \in H\}$$

Topology (radial, axial, or integrated), permanent magnet shapes (strip, arc, disk, rectangular, rhomb, triangular, etc.), permanent magnet characteristics (*BH*demagnetization curve, energy product, hysteresis minor loop, etc.), *electromotive force* distribution, cooling, power, torque, size, torque-speed characteristics, bearing, packaging, as well as other distinct features can be classified. Hence, microelectromechanical motion devices can be devised and classified by an *N*-tuple as: motion device type, electromagnetic system, geometry, topology, phase, winding, sizing, bearing, cooling, fabrication, materials, packaging, and so forth.

By using distinct geometry, electromagnetic systems (*endless*, *open ended*, and *integrated*), topologies, and other features, novel high-performance motion devices can be synthesized by utilizing the Synthesis and Classification Solver. For example, the spherical, conical, and cylindrical geometry of two-phase permanent-magnet synchronous motion devices are illustrated in Figure 17.5. The cross-section of the slotless radial-topology synchronous microactuator, fabricated on the silicon substrate with polysilicon stator (with deposited windings), polysilicon rotor (with deposited permanent magnets), and contact

TABLE 17.1 Classification of Microelectromechanical Motion Devices Using the Electromagnetic System—Geometry Using Synthesis and Classification Solver

M \ G	Plate, P	Spherical, S	Torroidal, T	Conical, N	Cylindrical, C	Asymmetrical,A
Electromagnetic System: *Endless* (Closed), E						
Electromagnetic System: *Open-Ended* (Open), O						
Electromagnetic System: *Integrated*, I						

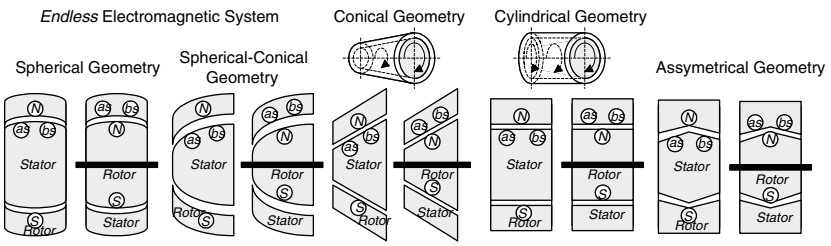


FIGURE 17.5 Two-phase permanent-magnet synchronous motion devices with *endless* electromagnetic system and distinct geometries.

bearing is illustrated in Figure 17.6. The fabrication of this microactuator and the processes are reported in References 1–3.

The major task is to devise high-performance motion devices relaxing fabrication difficulties guarantying affordability. The electrostatic and planar motion devices fabricated and tested to date are found to be inadequate for a wide range of applications owing to specifications imposed on performance [1–3]. Figure 17.7 illustrates the axial topology motion device. The stator is made on the substrate with deposited windings (surface micromachined or printed coils can be made using the fabrication processes as well as using double-sided substrate, applying conventional photolithography and etching processes [1–3]). The bearing post is fabricated on the stator substrate and the bearing hold is a part of the rotor microstructure. The rotor with permanent-magnet thin films rotates because of the electromagnetic torque developed. It is important to emphasize that stator and rotor are made using the conventional well-developed processes and materials.

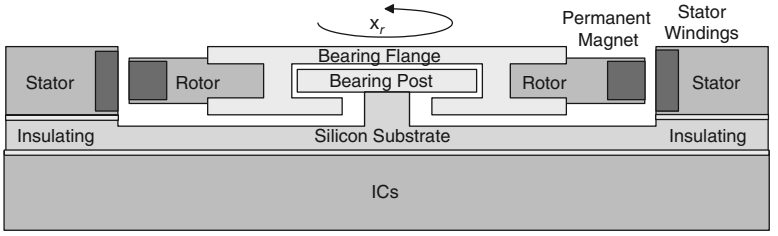


FIGURE 17.6 Cross-section schematics of a slotless radial-topology permanent-magnet brushless microactuator with controlling ICs.

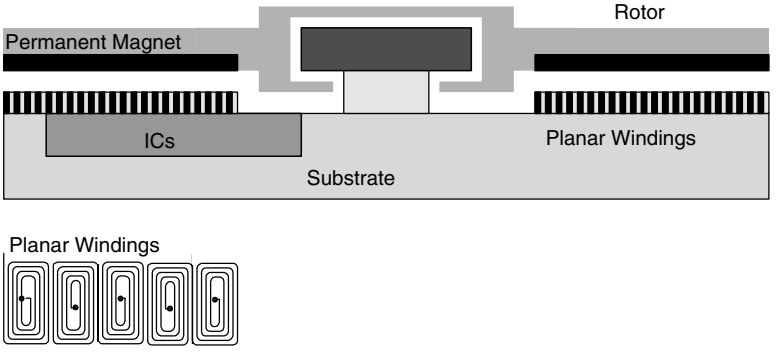


FIGURE 17.7 Cross-section of an axial topology motion device with controlling ICs.

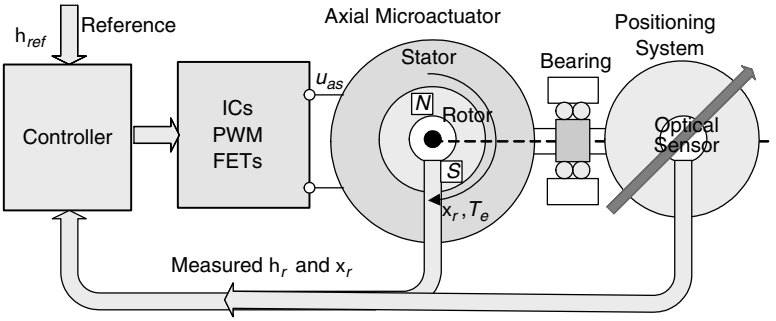


FIGURE 17.8 Schematic diagram of a servosystem: controller; ICs; actuator/sensors; and mechanism.

17.4 Micromechatronic System with an Axial Topology Motion Device

High-torque-density permanent-magnet axial-topology actuators with high switching-frequency power electronics and ICs, controlled by controllers (microcontroller or analog controllers), are found to be a very promising solution. High performance is achieved by utilizing: (1) high-torque-density brushless motion devices; (2) designing tracking controllers implemented utilizing controllers that integrate filters, IO devices, and advanced converters; (3) high-performance two- and four-quadrant power stages. The control laws are designed to satisfy the specifications imposed. Consider a servosystem as depicted in Figure 17.8. The reference signal is the angular displacement. Using the reference θ_{ref} and actual θ_t angular displacements (measured by the high-accuracy optical sensor), the controller develops PWM signals to

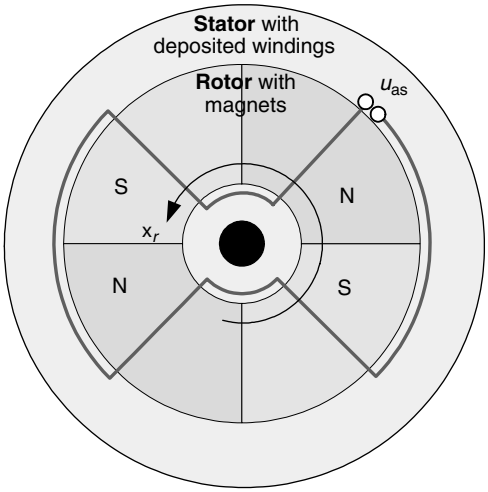


FIGURE 17.9 Axial topology actuator.

drive high-frequency field effect transistors in the power stage. The output voltage u_{as} is controlled by changing the duty cycle.

A limited-angle axial-topology permanent-magnet microactuator is documented in Figure 17.9. This device is used in the studied hard drive reported in Figure 17.8. To rotate the disk and displace the pointer, we utilize axial topology direct-drive synchronous actuators with the segmented array of the permanent-magnet strips. These high-torque-density permanent-magnet actuators exceed performance characteristics achieved by other servomotors [1–3]. The mathematical model must be found and used in the design of controllers as well as numerical analysis. The magnetic field developed by the magnets is approximated as continuous or discontinuous functions, for example, $B(\theta_r) = B_{max}\,\mathrm{sgn}(\sin(\frac{1}{2}\,\mathrm{N}_m\theta_r))$, where B_{max} is the magnetic flux density from the permanent magnets as viewed at the coils; Nm is the number of magnets. For the limited-angle actuator, depending on the geometry, magnetization, and separations of magnets, one can approximate $B(\theta_r)$ as $B(\theta_r) = B_{max}\tanh^q(k\theta_r)$, where q is the technologydependent integer, and usually one finds $q = 1$ or $q = 3$; k is the technology-dependent coefficient.

The electromagnetic torque is derived using the magnetic dipole moment **m**, for example, $T_e = \mathbf{m} \times \mathbf{B}$. That is, for the symmetric two-filament limited angle actuator, for $q = 1$ we obtain $T_e = 2NlB_{max}\,|\tanh(k\theta_r)|\,i_{as}$, where N is the number turns in filaments; l is the equivalent coil active length; i_{as} is the current in the coil.

The mathematical model is found by using Kirchhoff's and Newton's second laws. In particular, the following circuitry and *torsional–mechanical* equations are used.

$$u_{as} = r_s i_{as} + \frac{d\psi_{as}}{dt},$$

$$T_e - B_m\omega_r - T_L = J\frac{d^2\theta_r}{dt^2},$$

where u_{as} is the applied voltage; r_s is coil resistance; ψ_{as} is the flux linkages as viewed from the winding; B_m is the friction coefficient; T_L is the load torque; J is the equivalent moment of inertia.

The flux linkage is expressed as

$$\psi_{as} = L_{as}i_{as} + A_eB(\theta_r),$$

where L_{as} is the self-inductance, and A_e is the equivalent cross-sectional area.

A set of nonlinear differential equations that models an axial topology microactuator is

$$\frac{\mathrm{d}i_{\mathrm{as}}}{\mathrm{d}t} = \frac{1}{L_{\mathrm{as}}}\left(-r_{\mathrm{s}}i_{\mathrm{as}} - \underbrace{A_{\mathrm{e}}\frac{\mathrm{d}B(\theta_{\mathrm{r}})}{\mathrm{d}t}}_{\text{back emf}} + u_{\mathrm{as}}\right) = \frac{1}{L_{\mathrm{as}}}\left(-r_{\mathrm{s}}i_{\mathrm{as}} - A_{\mathrm{e}}B_{\max}k\,\mathrm{sech}^2(k\theta_{\mathrm{r}})\omega_{\mathrm{r}} + u_{\mathrm{as}}\right),$$

$$\frac{\mathrm{d}\omega_{\mathrm{r}}}{\mathrm{d}t} = \frac{1}{J}\left[2NlB_{\max}\left|\tanh(k\theta_{\mathrm{r}})\right| i_{\mathrm{as}} - B_{\mathrm{m}}\omega_{\mathrm{r}} - T_{\mathrm{L}}\right],$$

$$\frac{\mathrm{d}\theta_{\mathrm{r}}}{\mathrm{d}t} = \omega_{\mathrm{r}}.$$

The bounded proportional-integral tracking controller with state feedbacks (Equation 17.18) is designed using the *state transformation* method. We use the expanded state vector as $z = \begin{bmatrix} x \\ u \end{bmatrix}$, $x(t) = \begin{bmatrix} x^{sys}(t) \\ e(t) \end{bmatrix}$, where $x^{\mathrm{sys}}(t) = \begin{bmatrix} i_{\mathrm{as}}(t) \\ \omega_r(t) \\ \theta_r(t) \end{bmatrix}$. Using different weighting matrices Q_z and G_z in Equation 17.14, distinct feedback gains are obtained. Letting $Q_z = I$ and $G_z = 1$, for the bounded controller (Equation 17.18) synthesized we have $k_p = 31$ and $k_i = 24$. The output dynamics for $\theta_{\mathrm{ref}} = 0.349$ rad is shown in Figure 17.10 for $T_s = 0.0014$ s. For $k_p = 31$ and $k_i = 24$ with the state feedbacks, the settling time is 0.027 s, and the overshoot is 27%. It is possible to minimize the overshoot to 13%. The application of proportional-integral controllersmay not ensure the desired transient behavior as reported in Figure 17.10, when $k_p = 20$ and $k_i = 19$. We tested the servo without a return spring that is commonly used to ensure mechanical damping. Thus, the most challenging problem is considered. Pointer acceleration and deceleration are achieved by properly controlling the actuator that operates at very high efficiency and develops high electromagnetic torque.We conclude that optimal performance is achieved.

17.5 Synchronous Micromachines

In micromechatronics systems, rotational and translational motion devices (actuators and sensors), controlled by ICs, are widely used. Among various devices, synchronous permanent-magnet motion devices exhibit superior performance. The advantages of axial topology motion devices are feasibility, efficiency, and reliability. Fabrication simplicity results because: (1) magnets are flat; (2) there are no strict shape requirements imposed on magnets; (3) rotor back ferromagnetic material is not required; (4) it is easy to deposit planar windings on the flat stator. An axial topology permanent-magnet two-phase synchronous motion device is documented in Figure 17.11.

For two-phase actuator, one supplies two phase voltages u_{as} and u_{bs} in order to develop an electromagnetic torque. Assuming that the magnetic flux is constant through the magnetic plane (current loop), the torque on a planar current loop of any size and shape in the uniformmagnetic field is $\mathbf{T} = \boldsymbol{i}\mathbf{s} \times \mathbf{B} = \mathbf{m} \times \mathrm{B}$, where i is the current in the loop (winding); $\mathbf{m}$ is the magnetic dipole moment. The electromagnetic force is given as $\mathrm{F} = \oint id\mathbf{l} \times \mathbf{B}$. The interaction of the current (in windings) and magnets will produce the rotating electromagnetic torque. Microscale synchronous transducers can be used as motors (actuators) and generators (sensors). Microgenerators (velocity and position sensors) convert mechanical energy into electrical energy, while micromotors (actuators) convert electrical energy into mechanical energy. A broad spectrum of synchronous microtransducers can be used as actuators, for example, microscale drives, servos, and power systems.

Distinct electrostatic and electromagnetic motion devices can be designed and fabricated using the surface micromachining technology. However, all high-performance, high-power, and torque density devices are electromagnetic, and these devices use permanent magnets. Therefore, the issues of fabrication and analysis of magnets are of great interest. Different soft and hard magnets can be fabricated using surface micromachining. In particular, Ni, Fe, Co, NiFe, and other magnets can be made. In general, four types of

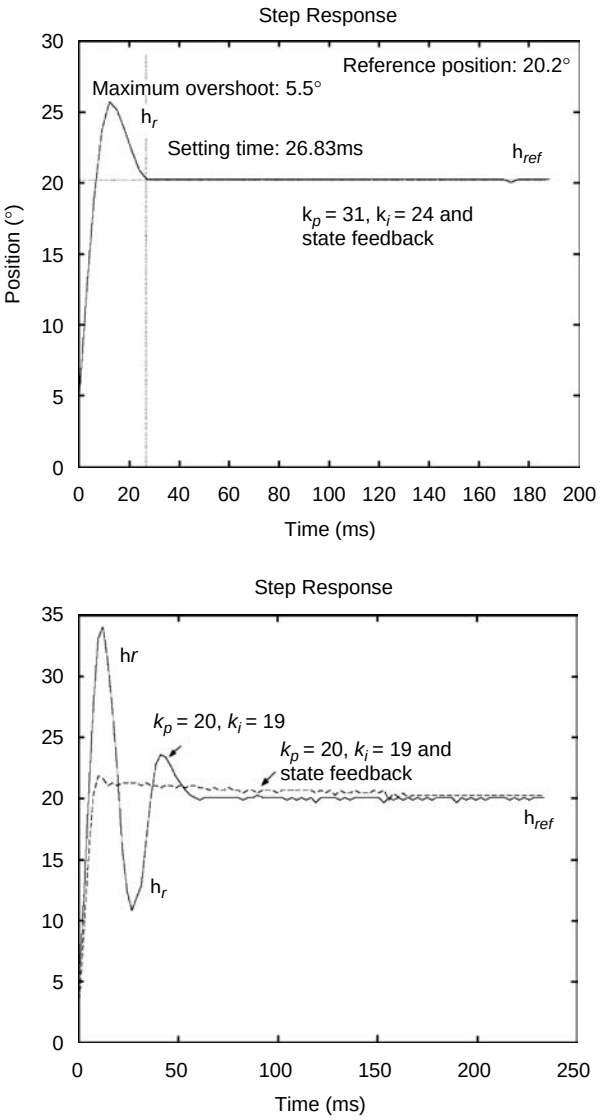


FIGURE 17.10 Transient dynamics of the closed-loop servo with the bounded proportional-integral controller with state feedback.

high-performance magnets have been commonly used in electromechanical motion devices, for example, neodymium iron boron ($Nd_2Fe_{14}B$), samarium cobalt (usually Sm_1Co_5 and Sm_2Co_{17}), ceramic (ferrite), and alnico (AlNiCo). The term soft is used to refer to magnets that have high saturation magnetization and a low coercivity (narrow *BH* curve). Another property of these magnets is their low magnetostriction. The soft magnets have been widely used in magnetic recording heads, and, in particular, NiFe thin films with the desired properties have been utilized. Hard magnets have wide *BH* curves (high coercivity) and, therefore, high-energy storage capacity. These magnets should be used in devices in order to attain high force, torque, and power densities. Unfortunately, limited progress has been made in the fabrication of hard permanent magnets.

The energy density is given as the area enclosed by the *BH* curve, for example, $w_m = \frac{1}{2}\mathbf{B} \cdot \mathbf{H}$. When magnets are used in motion devices, the demagnetization curve (second quadrant of the BH curve) is of

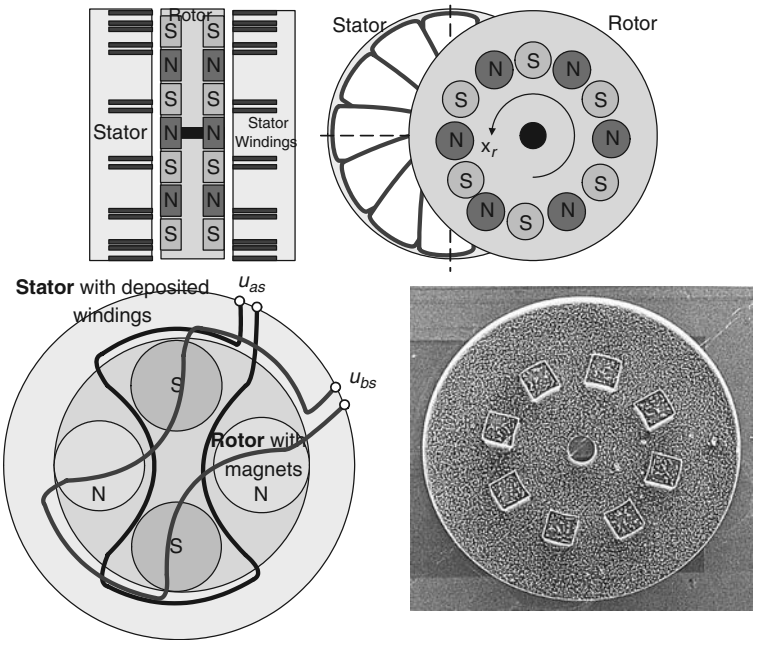


FIGURE 17.11 Axial permanent-magnet synchronous motion device.

interest. Permanent magnets store, exchange, and convert energy. In particular, permanent magnets produce stationary magnetic field without external energy sources or radiation devices.We apply Kirchhoff's and Newton's laws to derive the equations of motion for axial topology permanent-magnet synchronous motion devices. Using an axial permanent-magnet synchronous device documented in Figure 17.11, we assume that this variation of the flux density is sinusoidal, for example, $B(\theta_r) = B\max \sin^n(\frac{1}{2} N\mathrm{m}\theta_r)$, $n = 1$, 3, 5, · · ·, where B_{max} is the maximum flux density in the airgap produced by the magnet as viewed from the winding; n is the integer that is a function of the magnet geometry and the waveform of the airgap B. It should be emphasized that B_{max} depends on the magnets used, airgap length, temperature, and so forth.

The electromagnetic torque, developed by a single-phase axial topology permanent-magnet synchronous actuator, is found using the expression for the coenergy $W_c(i_{as}, \theta_r)$ that is given as $W_c(i_{as}, \theta_r) = NA_eB(\theta_r)i_{as}$. Assuming that the airgap flux density obeys $B(\theta_r) = B_{max} \sin(\frac{1}{2} N_m\theta_r)$, we have $W_c(i_{as}, \theta_r) = NA_eB_{max} \sin(\frac{1}{2} N_m\theta_r)i_{as}$, and the electromagnetic torque is

$$T_e = \frac{\partial W_c(i_{as}, \theta_r)}{\partial \theta_r} = \frac{\partial \left(NA_eB_{max} \sin(\frac{1}{2}N_m\theta_r)i_{as}\right)}{\partial \theta_r} = \frac{1}{2}N_mNA_eB_{max} \cos\left(\frac{1}{2}N_m\theta_r\right) i_{as}.$$

As we fed the following currentwith the magnitude i_M to the micromotorwinding $i_{as} = i_M \cos(\frac{1}{2} N_m\theta_r)$, the electromagnetic torque is $T_e = \frac{1}{2} N_mNA_eB_{max}i_M \cos^2(\frac{1}{2} N_m\theta_r) \neq 0$. The mathematical model of the single-phase permanent-magnet micromotor is found by using Kirchhoff's and Newton's second laws. In particular, one obtains

$$u_{as} = r_s i_{as} + \frac{d\psi_{as}}{dt} \text{ (circuitry equation)},$$

$$T_e - B_m\omega_r - T_L = J\frac{d^2\theta_r}{dt^2} \text{ (torsional–mechanical equation)}.$$

Taking note of the flux linkage equation $\psi_{as} = L_{as} i_{as} + NA_e B(\theta_r)$, we have a set of three first-order nonlinear differential equations that models a single-phase axial topology permanent-magnet synchronous motion device

$$\frac{di_{as}}{dt} = \frac{1}{L_{as}}\left(-r_s i_{as} - \frac{1}{2}N_m N A_e B_{max}\cos\left(\frac{1}{2}N_m\theta_r\right)\omega_r + u_{as}\right),$$
$$\frac{d\omega_r}{dt} = \frac{1}{J}\left(\frac{1}{2}N_m N A_e B_{max}\cos\left(\frac{1}{2}N_m\theta_r\right) i_{as} - B_m\omega_r - T_L\right),$$
$$\frac{d\theta_r}{dt} = \omega_r.$$

Single-phase axial and radial topologies permanent-magnet synchronous actuators have a torque ripple. The torque ripple can be minimized feeding $i_{as} = i_M\left[\cos(\frac{1}{2}N_m\theta_r) + \varepsilon\right]^{-1}$, $|i_{as}| \le i_M$, where $\varepsilon > 0$. To overcome this deficiency (torque ripple and velocity chattering) and increase torque density, two-phase motion devices are used. For two-phase actuators, the coenergy is given as $W_c(i_{as}, \theta_r) = NA_{ag}B_{max}\left(\sin(\frac{1}{2}N_m\theta_r)i_{as} - \cos(\frac{1}{2}N_m\theta_r)i_{bs}\right)$, and, hence, the electromagnetic torque is $T_e = \frac{\partial W_c(i_{as},\theta_r)}{\partial\theta_r} = \frac{1}{2}N_m N A_e B_{max}\left(\cos(\frac{1}{2}N_m\theta_r)i_{as} + \sin(\frac{1}{2}N_m\theta_r)i_{bs}\right)$. Thus, feeding the phase currents i_{as} and i_{bs} as $i_{as} = i_M\cos(\frac{1}{2}N_m\theta_r)$ and $i_{bs} = i_M\sin(\frac{1}{2}N_m\theta_r)$, we maximize the electromagnetic torque because $T_e = \frac{1}{2}N_m N A_e B_{max} i_M$.

17.6 Fabrication Aspects

Mini and microscale electromechanical motion devices can be fabricated by creating (etching or depositing) of conductors (coils and windings), ferromagnetic core, magnets, insulating layers, as well as other microstructures (radiating energy devices, movable and stationarymembers, and their components including bearings and posts). The subsequent order of the processes, sequential steps, and materials are different depending on the devised, designed, analyzed, and optimized devices. Several textbooks [1–6] provide the reader with the basic fabrication features and processes. In partcilular, the electrostatic devices are covered in References 1–6, while the electromagnetic motion devices are reported in References 1–3. Complementary metal oxide semiconductor (CMOS), high aspect ratio (LIGA and LIGA-like), and surface micromachining technologies are key features for fabrication of microdevices and structures. The LIGA (Lithography–Galvanoforming–Molding, or in German Lithografie–Galvanik–Abformung) technology allows one to make three-dimensional microstructures with the aspect ratio (depth versus lateral dimension is more than 100). The LIGA technology is based on the x-ray lithography that ensures short wavelength (from few to 10 Å), which leads to negligible diffraction effects and larger depth of focus compared with photolithography. The major processes in the machine's microfabrication are diffusion, deposition, patterning, lithography, etching, metallization, planarization, assembling, and packaging. Thin film fabrication processes were developed and used for polysilicon, silicon dioxide, silicon nitride, and other different materials (e.g., metals, alloys, composites, etc.). The basic steps are lithography, deposition of thin films and materials (electroplating, chemical vapor deposition, plasma-enhanced chemical vapor deposition, evaporation, sputtering, spraying, screen printing, etc.), removal of material (patterning) by wet or dry techniques, etching (plasma etching, reactive ion etching, laser etching, etc.), doping, bonding (fusion, anodic, and other), and planarization.

To fabricate motion and radiating energy microscale structures and devices, different fabrication technologies are used [1–6]. New processes were developed and novel materials were applied modifying the CMOS, surface micromachining, and LIGA technologies. Currently, the state-of-the-art technology in microfabrication has been progressed to the nanometer minimal features. Motion devices and structures (stator, rotor, bearing, coils, etc.) are defined photographically, and the high-resolution photolithography is applied to define two- (planar) and three-dimensional shapes (geometry). Deep ultraviolet lithography processes were developed to decrease the feature sizes of microstructures to 0.1 $f\hat{E}$m and less. Different

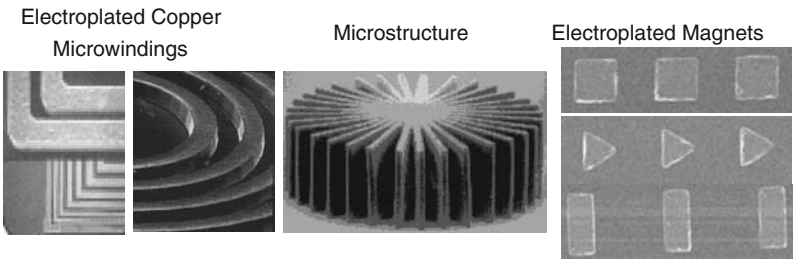


FIGURE 17.12 Deposited copper microwindings, microstructure, and permanent magnets.

exposure wavelengths λ (435, 365, 248, 193 nm) are used. Using the Rayleigh model for image resolution, the expressions for image resolution i_R and the depth of focus d_F are $i_R = k_i \frac{\lambda}{N_A}$ and $d_F = k_d \frac{\lambda}{N_A^2}$, where k_i and k_d are the lithographic process constants; λ is the exposure wavelength; N_A is the numerical aperture coefficient, and for high-numerical aperture we have $N_A = 0.5$–0.6.

The g- and i-line IBM lithography processes (with wavelengths of 435 and 365 nm, respectively) allow one to attain minimal features in the range of 0.35 μm. The deep ultraviolet light sources (mercury source or excimer lasers) with 248-nm wavelength enable the industry to achieve 0.25 μm resolution, leading to the minimal features in the range of 0.13 μm. The changes to short exposure wavelength possess challenges and present new highly desired possibilities. Using advanced lithography, 90-nm features were achieved. Different lithography processes commonly applied are photolithography, screen printing, electron-beam lithography, x-ray lithography (high-aspect ratio technology), and so forth. Althoughmachine topologies and configurations vary, magnetic and insulating materials, magnets, and windings are used in all motion devices. Figure 17.12 illustrates the electroplated 10 μm wide and thick with 10 μm spaced insulated copper microwindings (deposited on ferromagnetic cores), microstructures, and permanent magnets (electroplated NiFe alloy). Finally, it must be emphasized that usually the size of microelectromechanical motion devices is defined by the torque and power densities specified. Those requirements, as well as physical limits, define the device dimensionality.

Acknowledgments

The author sincerely acknowledges the support from the NSF under the NSF awards DUE 0311588 and EEC 0407281. *Disclaimer*—Any opinion, findings, and conclusions or recommendations expressed in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

1. V. Giurgiutiu and S. E. Lyshevski, *Micromechatronics: Modeling, Analysis and Design with MATLAB*, CRC Press, Boca Raton, FL, 2003.
2. S. E. Lyshevski, *MEMS and NEMS: Systems, Devices, and Structures*, CRC Press, Boca Raton, FL, 2001.
3. S. E. Lyshevski, *Nano- and Micro-Electromechanical Systems: Fundamental of Micro- and Nano-Engineering*, CRC Press, Boca Raton, FL, 2005.
4. S. A. Campbell, *The Science and Engineering of Microelectronic Fabrication*, Oxford University Press, New York, 2001.
5. G. T. A. Kovacs, *Micromachined Transducers Sourcebook*,WCB McGraw-Hill, Boston, MA, 1998.
6. M. J. Madou, *Fundamentals of Microfabrication*, CRC Press, Boca Raton, FL, 2001.

18

Introduction to Computers and Logic Systems

Kevin C. Craig
Fred Stolfi
Rensselaer Polytechnic Institute

18.1 Introduction: The Mechatronic Use of Computers

Mechatronics is the synergistic combination of mechanical engineering, electronics, control systems, and computers. The key element in mechatronics is the integration of these areas through the design process. Synergism and integration in design set a mechatronic system apart from a traditional, multidisciplinary system. In a mechatronic system, computer, electronic, and control technology allow changes in design philosophy, which lead to better performance at lower cost: accuracy and speed from controls, efficiency and reliability from electronics, and functionality and flexibility from computers. Automotive engine-control systems are a good example. Here a multitude of sensors measure various temperatures, pressures, flow rates, rotary speeds, and chemical composition and send this information to a microcomputer. The computer integrates all this data with preprogrammed engine models and control laws and sends commands to various valves, actuators, fuel injectors, and ignition systems so as to manage the engine's operation for an optimum combination of acceleration, fuel economy, and pollution emissions.

In mechatronics, balance is paramount. The essential characteristic of a mechatronics engineer and the key to success in mechatronics design is a balance between two sets of skills:

- Modeling (physical and mathematical), analysis (closed-form and numerical simulation), and control design (analog and digital) of dynamic physical systems
- Experimental validation of models and analysis and understanding the key issues in hardware implementation of designs

In mechatronic systems, computers play a variety of roles. First, computers are used to model, analyze, and simulate mechatronic systems and mechatronic system components and, as such, are useful for control design. Second, computers, as part of measurement systems, are used to measure the performance

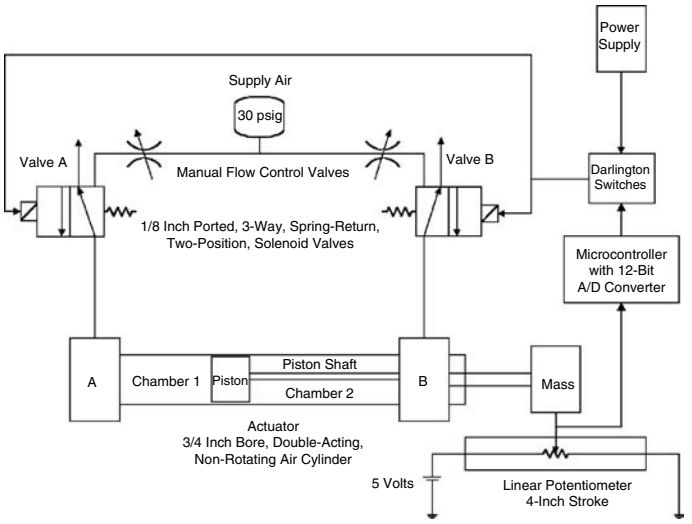


FIGURE 18.1 Pneumatic servomechanism.

of mechatronic systems, to determine the value of component parameters, and to experimentally validate models. Finally, computers or microcomputers form the central component in digital control systems for mechatronic designs. Thus, computers play an essential role in the two essential characteristics of the mechatronics balance and comprise a key component to mechatronic system designs. This is illustrated by the following example.

Consider the schematic of a pneumatic servomechanism, a computer-controlled, closed-loop positioning system, shown in Figure 18.1. Pneumatic servomechanisms have the advantages of low cost, high power-to-weight ratio, ease of maintenance, cleanliness, and a readily-available and cheap power source. However, the disadvantages are high, nonlinear friction forces, deadband due to stiction, and dead time due to the compressibility of air. The design goal is to implement a fast, accurate, and inexpensive pneumatic-actuator system using inexpensive on/off solenoid valves, rather than expensive continuously-variable servo valves. To accomplish this task, one must completely understand the physical system, develop a physical model on which to base analysis and design, and experimentally determine and/or validate model parameters. One must then develop a mathematical model of the system, analyze the system, and compare the results of the analysis to experimental measurements to validate the model. One must then design a closed-loop position control system utilizing on/off, modified on/off, or pulse-width modulated control. Finally, one must implement the control system and experimentally validate its predicted performance.

A MatLab/Simulink model of this system is shown in Figure 18.2. The mathematical model is highly nonlinear, as are the various control schemes. A computer numerical simulation is needed to understand the behavior of the system and the various control schemes. A data acquisition system is needed to take measurements of the various system inputs and outputs and validate the numerical simulation. And, a computer (a microcontroller in this case) is needed for the real-time implementation of the various control schemes. There are a variety of computer numerical simulation tools available, some requiring the detailed mathematical model while others enable virtual prototyping where the various system components are assembled on the computer screen with the component mathematical models given hidden in the background. There are also a variety of computer platforms on which to run the control algorithm, e.g., high-end PC using a DSP board and a real-time control-code generator; a microcontroller programmable in C or Basic with an analog-to-digital (A/D) converter and numerous digital input/output (I/O) ports; and a microchip implementation needed for product development.

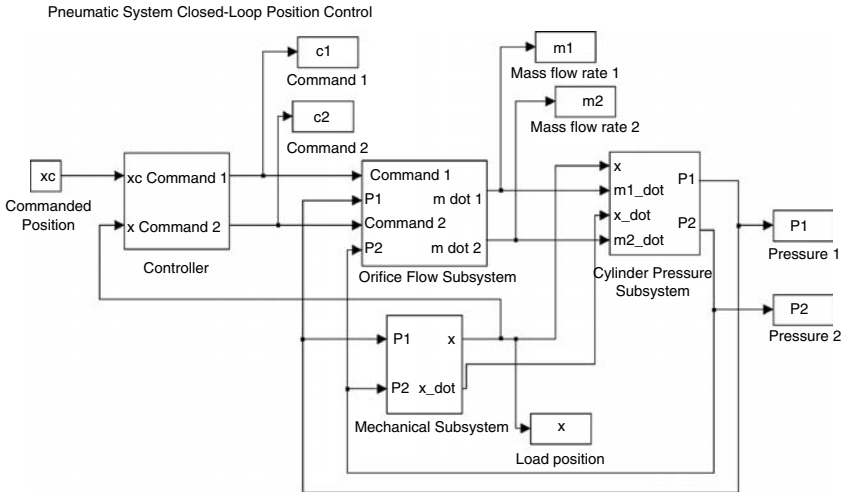


FIGURE 18.2 MatLab/Simulink model of the pneumatic servomechanism.

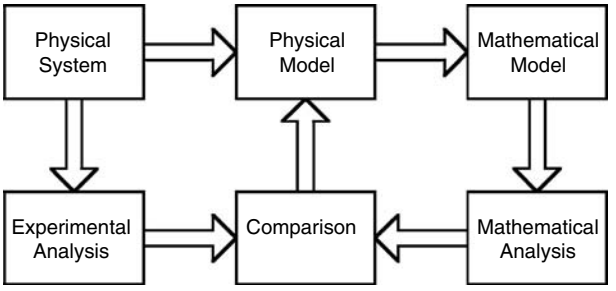


FIGURE 18.3 Dynamic system investigation process.

18.2 Mechatronics and Computer Modeling and Simulation

In design, balance is the key to success, i.e., balance between theory and practice and balance between modeling/analysis skills and hardware-implementation/measurement skills. Figure 18.3 illustrates the steps in a dynamic system investigation, which is the process that would be utilized to design a mechatronic system. The distinction between physical modeling and mathematical modeling is emphasized, as is the importance of both analytical and numerical solutions to the model equations. To generate a physical model, approximations must be made to the actual physical system. Small effects are neglected. The influence of the environment is ignored. Elements are assumed to be lumped instead of distributed. The dynamics are assumed to be linear. Parameters are assumed to be constant. Noise and uncertainty is ignored. These approximations have a direct influence on the mathematical model. Neglecting small effects limits the number of equations. Environmental independence reduces the complexity of the equations. Other approximations result in linear ordinary differential equations with constant coefficients. Neglecting uncertainty avoids the use of statistics in the model. In most cases, a design consideration is to develop the simplest model which adequately depicts the complexity of the system dynamics.

The predicted dynamic behavior of the model is only half the story, for these results, without experimental verification, are at best questionable, and at worst useless. Comparing the predicted dynamic behavior with the actual measured dynamic behavior is the key step in the dynamic system investigation process.

The steps in the dynamic system investigation process should be applied not only when an actual physical system exists (as in reverse engineering) and one desires to understand and predict its behavior, but also when the physical system is a concept in the design process that needs to be analyzed and evaluated. After recognizing a need for a new product or service, one uses past experience (personal and vicarious), awareness of existing hardware, understanding of physical laws, and creativity to generate design concepts. *The importance of modeling and analysis in the design process has never been more important than in this situation.* These design concepts can no longer be evaluated by the build-and-test approach because it is too costly and time consuming. Validating the predicted dynamic behavior in this case, when no actual physical system exists, becomes even more dependent on one's past hardware and experimental experience.

In physical modeling, one first specifies the physical system to be studied along with the system boundaries, input variables, and output variables. In modeling dynamic systems, we use engineering judgment and simplifying assumptions to develop a physical model. The complexity of the physical model depends on the particular need, e.g., system design iteration, control system design, control design verification, physical understanding. The intelligent use of simple physical models requires that we have some understanding of what we are missing when we choose the simpler model over the more complex model. The astuteness with which these approximations are made at the onset of an investigation is the very crux of engineering analysis. A variety of engineering models may be developed based on the particular need. Always ask the question: "Why am I modeling the physical system and what is the range of operation that I wish my model to be valid for?" If the need is system-design iteration or control-system design, then a "*design model*" is needed, i.e., a physical model whose mathematical model is a linear ordinary differential equation with constant coefficients and, therefore, useful with a broad, highly-developed assortment of linear design techniques. If the need is design verification before actual hardware implementation, then a "*truth model*" is needed, i.e., a physical model that is as close to reality as possible; with nonlinear simulation tools available, almost any mathematical model can now be simulated. Iterations can then be performed using, as a starting point, the results of the work performed with the design model. Models only need to be valid for the particular range of operation of interest; low-order models then can often represent very complex, higher-order models very effectively. In practice, you may need a hierarchy of models of varying complexity: a very detailed truth model for final performance evaluation before hardware implementation, several less complex truth models for use in evaluating particular effects, and one or more design models.

18.3 Mechatronics, Computers, and Measurement Systems

Measurement systems or data acquisition systems may be used for a variety of purposes, and a computer plays an integral role in each.

1. *Monitoring of Processes and Operations.* Certain applications of measuring instruments may be characterized as having essentially a monitoring function, e.g., thermometers, barometers, and water, gas, and electric meters.
2. *Control of Processes and Operations.* An instrument can serve as a component of a control system. To control any variable in a feedback control system, it is first necessary to measure it. A single control system may require information from many measuring instruments, e.g., industrial machine and process controllers, aircraft control systems.
3. *Experimental Engineering Analysis.* In solving engineering problems, two general methods are available: theoretical and experimental. Many problems require the application of both methods and theory and experiment should be thought of as complementing each other. Further, all models need validation, and measurement systems offer a means to collect the data required for model validation.

The distinction among monitoring, control, and analysis functions is not clear-cut; the category that a given application may fit may depend somewhat on the engineer's point of view and the apparent looseness of the classifications should not cause any difficulty. Rather it should be realized that computers,

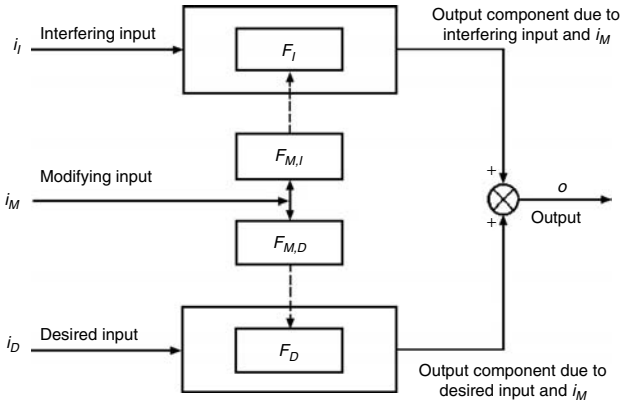


FIGURE 18.4 Input–output configuration of a measurement system.

as general purpose processing elements, can serve many functions in the processing of measured parameters from mechatronic systems and that these processing functions can be related to or unrelated to the modeling and control of such systems. Special purpose digital signal processing electronics are also used in measurement systems. High-speed digital signal processors (DSPs), for example, are used to collect input and output signals in the determination of transfer functions for mechatronic systems. The high speed allows the processing of simultaneous samples of the input and output for minimal phase error. The primary application for DSPs in mechatronic systems, however, is real-time control, discussed below.

Figure 18.4 is the input–output configuration of a measurement system. Input quantities are classified into three categories:

1. *Desired Inputs.* These are quantities that the instrument is specifically intended to measure.
2. *Interfering Inputs.* These are quantities to which the instrument is unintentionally sensitive. F_D and F_I are input–output relations, i.e., the mathematical operations necessary to obtain the output from the input. They represent different concepts depending on the particular input–output characteristic being described, e.g., a constant, a mathematical function, a differential equation, a statistical distribution function.
3. *Modifying Inputs.* These are quantities that cause a change in the input–output relations for the desired and interfering inputs, i.e., they cause a change in F_D and/or F_I. $F_{M,I}$ and $F_{M,D}$ represent the specific manner in which i_M affects F_I and F_D, respectively.

There are several methods for canceling or reducing the effects of spurious inputs. One method which relies upon computer processing of the signals is the method of calculated output corrections. This method requires one to measure or estimate the magnitudes of the interfering and/or modifying inputs and to know quantitatively how they affect the output. Then it is possible to calculate corrections, which may be added to or subtracted from the indicated output so as to leave (ideally) only that component associated with the desired input. Since many measurement systems today can afford to include a computer to carry out various functions, if sensors for the spurious inputs are provided, the computer can implement the method of calculated output corrections on an automatic basis.

18.4 Mechatronics and the Real-Time Use of Computers

We turn to the field of closed-loop control using a digital computer as the controller. Several comments are in order. First, a mechatronic system typically involves continuous variables. Elements rotate or translate in space. Fluids or gasses flow. Heat or energy is transferred. Computers are, by their nature, digital elements. Variables are represented in a computer by discrete values or simply by collections of zeroes and ones. For a computer to be used as the controller for a mechatronic system, therefore, the

continuous variables must be converted to discrete variables for processing and then back again to continuous variables. This might seem obvious. What is not so apparent is that the computer algorithm forms an inherent separation between the processing of the signals and the signals themselves, which is not true of other mechatronic system components. Even if digital logic elements are used (as discussed in this chapter) the signals are converted to discrete form, but the flow of information is still continuous through the elements. When a computer is used for the control element, this information flow is broken and buried in the computer algorithm. As an example, computer algorithms sometimes mimic continuous proportional-integral-derivative (PID) control laws. When the execution of this algorithm is analyzed, even if the effects of sampling and quantization are included, it is assumed that the signals are processed just as if they were being determined by continuous processing elements. In reality, if the computer code is examined at the machine level (i.e., not in the high level language in which it may be written), it would bear very little resemblance to a differential equation representation of the PID algorithm. This has practical implications both for modeling the exact operation of the computer as a control element and for validating that the computer code actually produces the desired response to signals.

Other issues are involved when the mechatronic system controller is implemented in software. Software execution is often asynchronous to the other time constants in the system (i.e., the software execution and system response are often not synchronized). Software can be made synchronous by syncing it to the sampler period, but this typically limits performance and is difficult if the computer is to be used for other tasks than control. Once a computer is contained as an element in a mechatronic system, there is a tendency to use some of the processing power to provide additional functionality or ease of use for the product. This additional code can affect, sometimes adversely, the operation of the real time controller execution. Testing of the code and safety of the code are also issues. The engineer has to determine that his system operates deterministically and safely for all possible combinations of input signals and for all possible states in the execution of the algorithm. For real-time systems, execution order for the code is often not predictable since it can be dependent on the particular combination of input signals. Simplicity of the code, providing for testability of the code, using established software quality assurance practices, and developing extensive documentation are ways to achieve system determinism and safety. Often, a hardware interlock, that is, a safety system utilizing electronic or mechanical hardware, is often included in software controlled systems.

Code operation has to be further verified as the code is modified and as the code is reused for systems other than that for which it was developed. Unlike other controllers, computer code is portable, but this requires more thought for its possible reuse. Using standard software packages, standard processors, modular code, and commercial real-time environments increases the possibility for reuse.

Besides the issues inherent in using computer code as the controller, there are issues involved whenever a digital processing component is incorporated into a mechatronic system. Further, there are considerations that must be taken into account whenever digital signals are processed. Figure 18.5 shows a configuration useful for this discussion. The computer is important, but the computer "component" of many mechatronic machines and processes is often not the critical system element in terms of either technical or economic factors. Rather, components external to the computer, the actuators and sensors, the sampling system, and the anti-aliasing filter are more often the limiting factors in the system design.

Since both continuous (analog) and digital signals exist in computer-controlled systems, the signals in such a system can be classified as shown in the table below.

Signal Classification	Discrete in Time	Continuous in Time
Discrete in amplitude	D-D (digital)	D-C
Continuous in amplitude	C-D	C-C (analog)

For analog signals, the precise value of the quantity (voltage, rotation angle, etc.) carrying the information is significant, meaning that the specific waveform of input and output signals is of vital importance. Conversely, digital signals are binary (on/off) in nature, and variations in numerical value are associated

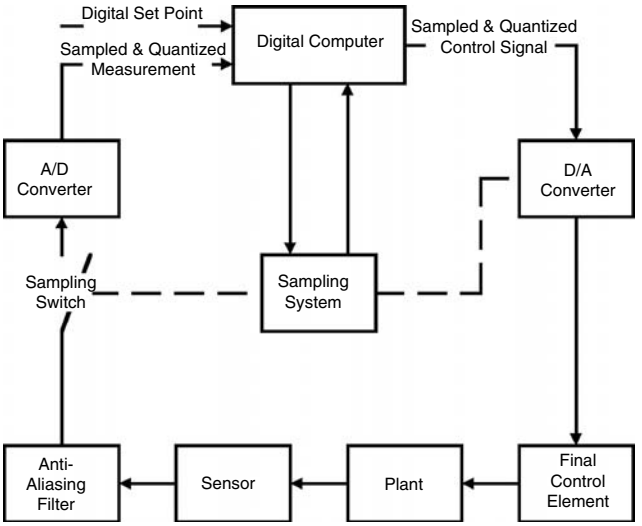


FIGURE 18.5 General computer-control configuration.

with changes in the logical state (true/false) of some combination of switches, e.g., +2 V to +5 V represents ON state, 0 V to +0.8 V represents OFF state.

In digital devices, it is simply the presence (logical 1) or absence (logical 0) of a voltage within some wide range that matters; the precise value of the signal is of no consequence. Digital devices are therefore very tolerant of noise voltages and need not be individually very accurate, even though the overall system can be extremely accurate. When combined analog/digital systems are used, the digital portions need not limit system accuracy; these limitations generally are associated with analog portions and/or the analog-to-digital (A/D) conversion devices. Since most mechatronic systems are analog in nature, it is necessary to have both A/D converters and digital-to-analog (D/A) converters, which serve as translators that enable the computer to communicate with the outside analog world.

In most cases, the sensor and the final control element are analog devices, requiring, respectively, A/D and D/A conversion at the computer input and output. There are, of course, exceptions, e.g., stepper motor and optical encoder. In most cases, however, the sensors can be thought of as providing analog voltage output and the final control element will accept an analog voltage input.

The current trend toward using dedicated, computer-based, and often decentralized (distributed) digital control systems in mechatronic applications can be rationalized in terms of the major advantages of digital control:

- Digital control is less susceptible to noise or parameter variation in instrumentation because data can be represented, generated, transmitted, and processed as binary words, with bits possessing two identifiable states.
- Very high accuracy and speed are possible through digital processing. However, hardware implementation is usually faster than software implementation. Determining the time required to develop a system in software is notoriously difficult to estimate.
- Digital control can handle repetitive tasks extremely well, through programming.
- Complex control laws and signal conditioning methods that might be impractical to implement using analog devices can be programmed. Very sophisticated algorithms can be implemented digitally.
- High product reliability can be achieved by minimizing analog hardware components and through decentralization using dedicated computers for various control tasks.

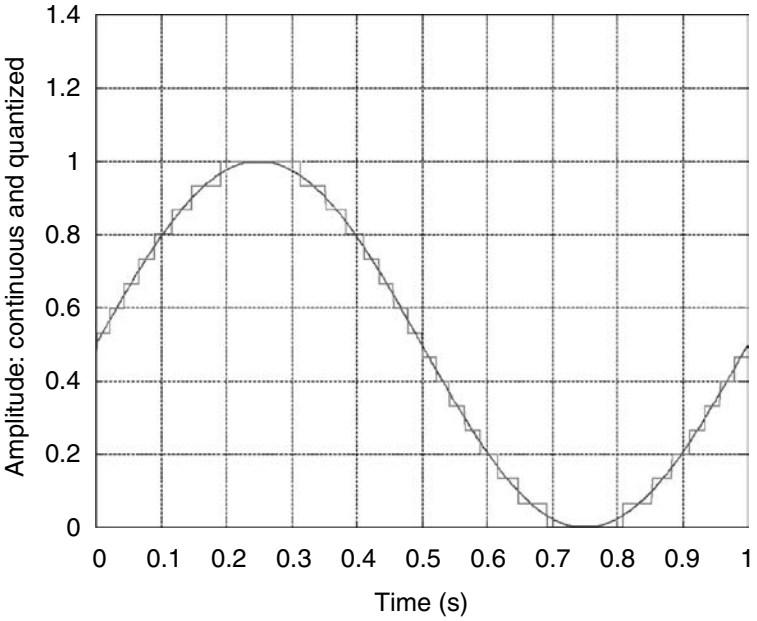


FIGURE 18.6 Simulation of a continuous and 4-bit quantized signal.

- Digital systems are more easily "programmed" and offer the ability to time-share a single processing unit among a number of different functions.
- Large amounts of data can be stored using compact high-density data storage methods.
- Data can be stored or maintained for very long periods of time without drift and without being affected by adverse environmental conditions. Digital control has easy and fast data retrieval capabilities.
- Fast data transmission is possible over long distances without introducing dynamic delays, as in analog systems.
- Digital processing uses low operational voltages (e.g., 0–12 V DC).
- Digital control has low overall component cost.

Further, from the standpoint of the mechatronic product, the inclusion of a computer means that additional system functions can be provided. The user can select from a range of operations. Additional features can be included. A user interface providing indications of operation can be added with minimal cost.

In a real sense, some of the problems of analysis and design of digital control systems (beyond the issues associated with software) are concerned with taking into account the effects of the sampling period, T, and the quantization size, q. If both T and q are extremely small (i.e., sampling frequency 50 or more times the system bandwidth with a 32-bit word size), digital signals are nearly continuous, and continuous methods of analysis and design can be used. It is most important to understand the *effects of all sample rates*, fast and slow, and the *effects of quantization* for large and small word sizes. Lower cost computers are typically slower and have a smaller word size. Figure 18.6 shows the effects of having too few quantization levels, i.e., too small a word size. The signal that will be processed by the controller has large errors over the original analog signal.

Figure 18.7 shows the effects of sampling. It is worthy to note that the *single most important impact* of implementing a control system digitally is often the delay associated with the D/A converter, i.e., $T/2$. This pure delay results in a substantial phase shift in the closed-loop feedback system and often limits the control operation.

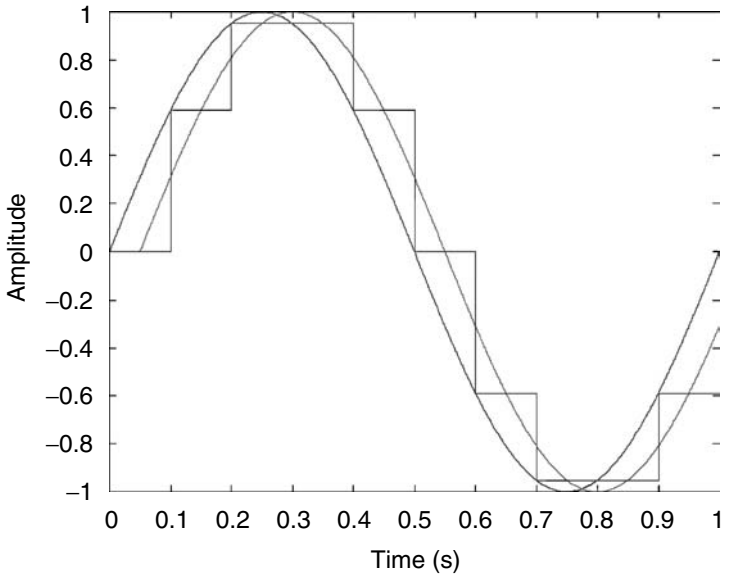


FIGURE 18.7 Continuous and D/A converter output.

In a feedback system, the analog signal coming from the sensor contains useful information related to controllable disturbances (relatively low frequency), but also may often include higher frequency "noise" due to uncontrollable disturbances (too fast for control system correction), measurement noise, and stray electrical pickup. Such noise signals cause difficulties in analog systems and low-pass filtering is often needed to allow good control performance. The phase shift from this filter also adversely affects control system stability.

Finally, in digital systems, a phenomenon called *aliasing* introduces some new aspects to the area of noise problems. If a signal containing high frequencies is sampled too infrequently, the output signal of the sampler contains low-frequency ("aliased") components not present in the signal before sampling. This is illustrated in Figure 18.8. If the higher frequency signal is sampled too infrequently, the result will be exactly the same values as the low frequency signal. From the standpoint of the controller, there is no way for the system to distinguish which signal is present. If we base our control actions on these false low-frequency components, they will, of course, result in poor control. The theoretical absolute minimum sampling rate to prevent aliasing is two samples per cycle; however, in practice, rates of about 10 are more commonly used. A high-frequency signal, inadequately sampled, can produce a reconstructed function of a much lower frequency, which cannot be distinguished from that produced by adequate sampling of a low-frequency function.

In all of the above, the word computer was used for the digital processing element. In electronics literature, a distinction is usually drawn between a microprocessor, microcomputer, DSP, and computer. There is no standard for what each of these terms can mean, but some insight can be gained by examining Figure 18.9, which is a general block diagram for a computer. All computers have a means of getting input, a means of generating output, a means of controlling the flow of signals and operations, memory for data storage, and an arithmetic logic unit (ALU) which executes the instructions. The ALU and control elements are often called the central processing unit (CPU). Small computers, which just contain a CPU, are often called microprocessors. Memory for these computers is often attached to the microprocessor but in distinct electronic packages. Input and output to the microprocessor is often handled by electronics called peripherals. If the memory is included in the same package, the computer is called either a microcomputer or computer depending on its physical size. CPU and memory on a single electronics chip is often

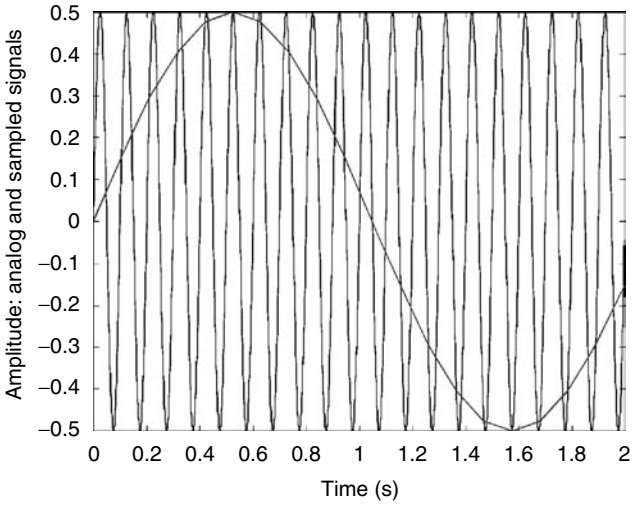


FIGURE 18.8 Simulation of continuous and sampled signal: aliasing.

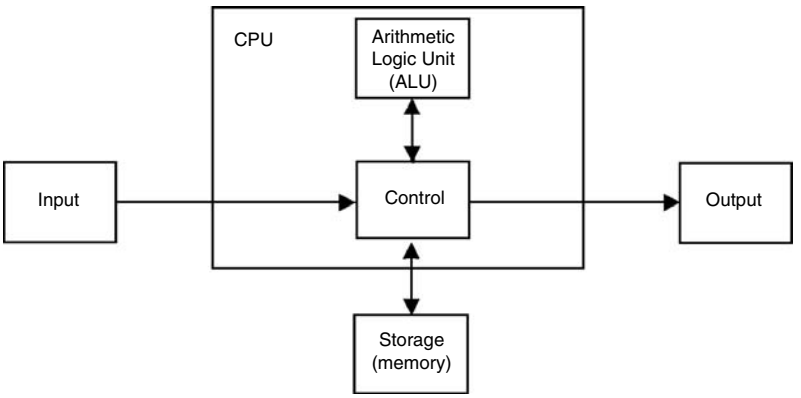


FIGURE 18.9 Elements of a computer.

called a microcomputer. The reader should be aware that a single electronics package can contain many "chips," which are connected by fine wires within the package. The overall package is still called a chip. Finally, if the A/D and D/A functions are provided in the same package, the computer is often called a DSP. However, these functions can also be contained in something which is called a microcomputer. DSPs are also computers which have a special instruction in the ALU called a multiply-accumulate (MAC) instruction even if the A/D and D/A are not present. Digital signal processing algorithms often involve MAC instructions and a computer, which can execute this instruction very effectively (in one instruction cycle of the computer), and are often called DSPs. To further complicate the situation, electronic devices called application specific integrated circuits (ASICs) exist. These devices can be custom made to perform a specific operation (such as a PID algorithm). ASICs can contain a CPU or memory or peripheral functions or even a MAC cell as part of its makeup. If the reader is thoroughly confused by this explanation, he probably has the proper grasp of the situation. However, he should be aware that diagrams like the one shown in Figure 18.9 often accompany the electronic component so the internal capabilities can be determined.

Before leaving computers, one final point will be made. Memory in a computer can often be divided between program space and data space, as shown in Figure 18.10. This representation is meant to be

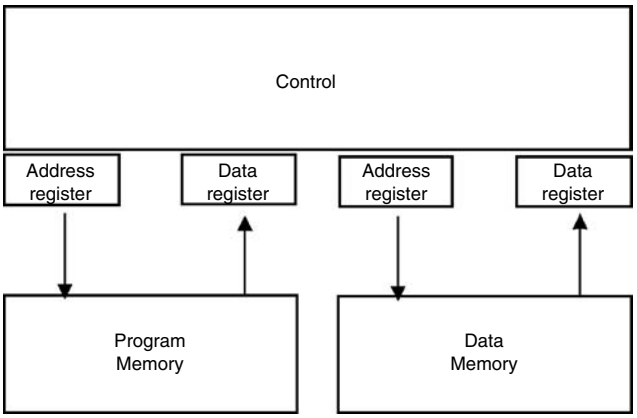


FIGURE 18.10 Computer memory organization.

pictorial rather than to define a specific computer architecture. In a von Neumann architecture, for example, the program memory and data memory share the same space and information busses. Whereas in a Harvard architecture, program memory and data memory are distinct (looking more like the figure). In either case, for a mechatronic system, one can think of the program (in program memory) as the set of instructions which tells the CPU how to manipulate data (in data memory) to produce an output. This view should emphasize the earlier point that the flow of signals in a mechatronic system becomes confused if a computer is to be used for real-time control.

Because of the low cost of modern microcomputers, the use of logic elements as discrete components in a mechatronic system has diminished. Microcomputers are often programmed to perform logic functions, which has the advantage that the operation can be altered in software rather than requiring electronic hardware changes. In analyzing this logic, of course, any of the traditional methods can be employed. The logic can be minimized via Karnaugh maps, for example. The only difference lies in the implementation of the algorithm. ASICs are also used to implement logic functions.

18.5 The Synergy of Mechatronics

As stated at the beginning of this section, mechatronics is the synergistic combination of mechanical engineering, electronics, control systems, and computers and the key element in mechatronics is the integration of these areas through the design process. The use of computers and logic elements as components in mechatronic systems will produce successful designs only if this synergy is achieved. The system must be designed as a system. Computers should never be an add-on component included when the design is complete. When computers are synergistically incorporated in the system, the power of the mechatronics approach to design is realized.

19

Digital Logic Concepts and Combinational Logic Design

George I. Cohn
California State University

19.1 Introduction

Digital logic deals with the representation, transmission, manipulation, and storage of digital information. A digital quantity has only certain discrete values in contrast with an analog quantity, which can have any value in an allowed continuum. The enormous advantage digital has over analog is its immunity to degradation by noise, if that noise does not exceed a tolerance threshold.

19.2 Digital Information Representation

Information can be characterized as qualitative or quantitative. Quantitative information requires a number system for its representation. Qualitative does not. In either case, however, digitalized information is represented by a finite set of different characters. Each character is a discrete quanta of information. The set of characters used constitutes the alphabet.

TABLE 19.1 Notation for Numbers

	Juxtaposition	Polynomial
Integer	$N = N_{n-1}N_{n-2} \ldots N_1N_0$	$N = \sum_{k=0}^{n-1} N_k R^k$
Fraction	$F = F_{-1}F_{-2} \ldots F_{-m+1}F_{-m}$	$F = \sum_{k=-m}^{-1} F_k R^k$
Real	$X = X_{n-1}X_{n-2} \ldots X_1X_0 \cdot X_{-1}X_{-1} \ldots X_{-m+1}X_{-m}$	$X = \sum_{k=-m}^{n-1} X_k R^k$

19.3 Number Systems

Quantitative information is represented by a number system. A character that represents quantitative information is called a **digit**. The number of different values which a digit may have is called the **radix**, designated by *R*. The symbols that designate the different values a digit can have are called numeric characters. The most conventionally used numeric characters are 0, 1, 2,…, etc., with 0 representing the smallest value. The largest value that a digit may have in a number system is the **reduced radix**, $r = R - 1$. Different radix values characterize different number systems: with *R* different numeric character values the number system is *R*nary, with 2 it is binary, with 3 it is ternary, with 8 it is octal, with 10 it is decimal, and with 16 it is hexadecimal.

Any value that can be expressed solely in terms of digits is an **integer**. A negative integer is any integer obtained by subtracting a positive integer from a smaller integer. Any number obtained by dividing a number by a larger number is a **fraction**. A number that has both an integer part and a fraction part is a **real number**.

All of the digits in a *number system* have the same radix. The radix is the **base** of the number system. Presumably, the possession of 10 fingers has made the decimal number system the most convenient for humans to use. The characters representing the 10 values a decimal digit can have are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The binary number system is the most natural for digital electronic systems because a desired reliability for a system can be most economically achieved using elements with two stable states. The characters normally used to represent the two values a binary digit may have are 0 and 1. The hexadecimal number system (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, *C*, *D*, *E*, and *F*; $R = 16$) is of importance because it shortens by a factor of four the string of digits representing the binary information stored and manipulated in digital computers.

19.4 Number Representation

Numbers that require more than one digit can be represented in different formats, as shown in Table 19.1. Different formats facilitate execution of different procedures. Arithmetic is most conveniently done with the juxtaposition format. Theoretical developments are facilitated by the polynomial format.

19.5 Arithmetic

The most common arithmetic processes, addition, subtraction, multiplication, and division are conveniently implemented using multidiget notation. Development of formulation procedures is facilitated using the polynomial notation. Since the numbers are digital representations, the logic used to maniputate the numbers is *digital logic*. However, this is different than the logic of boolean algebra, which is what is usually meant by the term digital logic. The logic of the former is implemented in hardware by using the logic of the latter. The four basic arithmetic operations can be represented as functional procedures in equation form or in arithmetic manipulation form, as shown in Table 19.2.

The arithmetic processes in the binary system are based on the binary addition and multiplication tables given in Table 19.3.

Table 19.4 gives binary examples for each of the basic arithmetic operations.

TABLE 19.2 Arithmetic Operations

	Algebraic Form	Arithmetic Form
Addition	Sum = Augend + Addend	Augend + Addend Sum
Subtraction	Difference = Minuend–Subtrahend	Minuend – Subtrahend Difference
Multiplication	Product = Multiplicand × Multiplier	Multiplicand × Multiplier Product
Division	Dividend/Divisor = Quotient + Remainder/Divisor	Quotient Remainder Divisor ⌊Dividend

TABLE 19.3 Single Digit Binary Arithmetic Table

(a) Addition

	0	1
0	0	1
1	1	10

(b) Multiplication

	0	1
0	0	0
1	0	1

TABLE 19.4 Binary Arithmetic Operation Examples

Addition:

```
 1 1 0 0   carries
 1 1 1 0 0  augend
+ 1 1 0 1  addend
-----------
1 0 1 0 0 1  sum
```

Multiplication:

```
     1 1 0 1   multiplicand
   × 1 1 0     multiplier
   ---------
     0 0 0 0   partial product 1
   1 1 0 1     partial product 2
 1 1 0 1       partial product 3
 -------------
 1 0 0 1 1 1 0  product
```

Subtraction, borrow method:

```
  10
 /101   borrows
 - 10   minuend
 ----   subtrahend
   11   difference
```

Division, with fraction remainder:

```
                 quotient      remainder
                   1 0 0 1 0     1 1 1
                 -------------   -----
1 0 1 1 | 1 1 0 0 1 1 0 1       dividend
divisor   1 0 1 1
          -------
            0 1 1 1 0
              1 0 1 1
              -------
                0 1 1 1
```

Subtraction, payback method:

```
   10
    101   borrows
  - 1     minuend
    - 10  payback
  ------  subtrahend
      11  difference
```

Division, with remainder in quotient:

```
                 quotient
                 ---------------------
1 0 1 1 | 1 1 1.0 0 0 0 0 0 0 0  dividend
          |
divisor     1 0 1 1
          -------
            1 1 0 0
            1 0 1 1
            -------
              0 1 0 0 0 0
                1 0 1 1
```

19.6 Number Conversion from One Base to Another

The method of using series polynomial expansions for converting numbers from one base to another is illustrated in Table 19.5.

Evaluation of polynomials is more efficiently done with the *nested form*. The nested form is obtained from the series form by successive factoring of the variable from all terms in which it appears as shown in Table 19.6. The number of multiplications to evaluate the nested form increases linearly with the order of the polynomial, whereas the number of multiplications to evaluate the series form increases with the square of the order.

Conversion of integers between bases is more easily done using the lower order polynomials, Table 19.6b, obtained by nesting. The least significant digit of the number in the new base is the remainder obtained after dividing the number in the old base by the new radix. The next least significant digit is the remainder obtained by dividing the first reduced polynomial by the new radix. The process is repeated until the most significant digit in the new base is obtained as the remainder, when the new radix no longer fits into the last reduced polynomial. This is more compactly represented with the arithmetic notation shown in Table 19.7 along with the same examples used to illustrate the polynomial series method.

TABLE 19.5 Series Polynomial Method for Converting Numbers between Bases

Method	Sample Conversion From	
	A Lower to a Higher Base	A Higher to a Lower Base
1. Express number in polynomial form in the given base	$101.1_2 = 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1}$	$19.5_{10} = 3\times 10^1 + 6\times 10^0 + 5\times 10^{-1}$
2. Convert radix and coefficients to the new base	$= 1\times 4 + 0\times 2 + 1\times 1 + 1\times 0.5$	$= 11\times 1010^1 + 110\times 1010^0 + 101\times 1010^{-1}$
3. Evaluate terms in the new base	$= 4 + 0 + 1 + 0.5$	$= 11\times 1010 + 110 + 101/1010$ $= 11110 + 110 + .1$
4. Add the terms	$101.1_2 = 5.5_{10}$	$19.5_{10} = 100100.1_2$

TABLE 19.6 Nested Polynomials

(a) Nested Polynomial via Iterated Factoring	(b) Lower Order Polynomials
$N = N_{n-1}R^{n-1} + N_{n-2}R^{n-2} + \cdots + N_2R^2 + N_1R + N0$	$N = N^{(1)}R + N_0$
$N = (N_{n-1}R^{n-2} + N_{n-2}R^{n-3} + \cdots + N_2R + N_1)R + N0$	$N^{(1)} = N^{(2)}R + N_1$
$N = ((N_{n-1}R^{n-3} + N_{n-2}R^{n-4} + \cdots + N_2)R + N_1)R + N0$	$N^{(2)} = N^{(3)}R + N_2$
$\vdots$	$\vdots$
	$N^{(n-2)} = N^{(n-3)}R + N_{n-2}$
$N = (\cdots(N_{n-1}R + N_{n-2})R + \cdots + N_2)R + N_1)R + N0$	$N^{(n-1)} = N_{n-1}$

TABLE 19.7 Radix Divide Method for Converting Numbers between Bases

Method		Sample Conversion From	
		A Higher to a Lower Base	A Lower to a Higher Base
$R\,\big/\,N$		$2\,\big/\,36_{10}$	
$R\,\big/\,N^{(1)}$	N_0	$2\,\big/\,18$ 0	
$R\,\big/\,N^{(2)}$	N_1	$2\,\big/\,9$ 0	$12\,\big/\,2012_3$
$R\,\big/\,N^{(3)}$	N_2	$2\,\big/\,4$ 1	$12\,\big/\,102$ 11
$\vdots$		$2\,\big/\,2$ 0	$12\,\big/\,2$ 1
		$2\,\big/\,1$ 0	$12\,\big/\,0$ 2
$R\,\big/\,N^{(n-1)}$	N_{n-2}	0 1	
$R\,\big/\,0$	N_{n-1}	$36_{10} = 100100_2$	$2012_3 = 214_5$

Conversion of a fraction from one base to another can be done by successive multiplications of the fraction by the radix of the number system to which the fraction is to be converted. Each multiplication by the radix gives a product that has the digits shifted to the left by one position. This moves the most significant digit of the fraction to the left of the radix point, placing that digit in the integer portion of the product, thereby isolating it from the fraction. This process is illustrated in algebraic form in the left column of Table 19.8 and in arithmetic form in the next column. Two sample numeric conversions are shown in the next two columns of Table 19.8.

Table 19.8 deals only with terminating fractions, that is, the remaining fractional part vanishes after a finite number of steps. For a nonterminating case the procedure is continued until a sufficient number of digits have been obtained to give the desired accuracy. A nonterminating case is illustrated in Table 19.9. A set of digits, which repeat ad infinitum, are designated by an underscore, as shown in Table 19.9.

TABLE 19.8 Radix Multiply Number Convertion Method (Terminating Case)

Formalism		Sample Conversion between Bases	
		Higher to Lower	Lower to Higher
Algebraic	Arithmetic		
$F = F_{-1}F_{-2}F_{-3}\cdots F_{-m}$	$F_{-1}F_{-2}F_{-3}\cdots F_{-m}$		$0.100101_{\,2}$
	$\times R$		$\times 1010$
		$0.125_{\,10}$	$101\,\vert\,11001$
$R^*F = F_{-1}\cdot F_{-2}F_{-3}\cdots F_{-m}$	$F_{-1}\,\vert\,\cdot F_{-2}F_{-3}\cdots F_{-m}$	$\times 2$	$\times 1010$
$= F_{-1}\cdot F^{(1)}$	$\times R$	$0\,\vert\,25$	$111\,\vert\,1101$
		$\times 2$	$\times 1010$
$R^*F^{(1)} = F_{-2}\cdot F_{-3}F_{-4}$	$F_{-2}\,\vert\,\cdot F_{-3}\cdots F_{-m}$	$0\,\vert\,5$	$1000\,\vert\,.001$
$\cdots F_{-m} = F_{-2}\cdot F^{(2)}$	$\times R$	$\times 2$	$\times 1010$
		$1\,\vert\,0$	$1\,\vert\,.10$
$R^*F^{(2)} = F_{-3}\cdot F_{-4}F_{-5}$	$\vdots$		$\times 1010$
$\cdots F_{-m} = F_{-2}\cdot F^{(3)}$			$10\,\vert\,.1$
$\vdots$			$\times 1010$
$R^*F^{(m-2)} = F_{-m+1}\cdot F_{-m}$	$F_{-m+1}\,\vert\,\cdot F_{-m}$		$101\,\vert\,.$
$= F_{-m+1}\cdot F^{(m-1)}$	$\times R$		
$R^*F^{(m-1)} = F_{-m}$	$F_{-m}\,\vert\,\cdot$	$.125_{\,10} = .001_{\,2}$	$.100101_{\,2} = .578125_{\,10}$

TABLE 19.9 Nonterminating Fraction Conversion Example

$0.1_{10} = 0.000110011\ldots_2$

$\times 2$
$0\,\vert\,.2$
$\times 2$
$0\,\vert\,.4$
$\times 2$
$0\,\vert\,.8$
$\times 2$
$1\,\vert\,.6$
$\times 2$
$1\,\vert\,.2$
$\times 2$
$0\,\vert\,.4$
$\times 2$
$0\,\vert\,.8$
$\times 2$
$1\,\vert\,.6$

or more compactly

$0.1_{10} = 0.\underline{00011}_{\,2}$

TABLE 19.10 Conversions between Systems Where One Base Is an Integer Power of the Other Base

(a) Conversion from high base to lower base
*B*2. $C5_{16} = 1011\quad 0010\quad .1100\quad 0101_2$
$62.75_8 = 110\quad 010\quad .111\quad 101_2$

(b) Conversion from lower base to high base
$11\quad 0010\quad 0100.0001\quad 1100\quad 01_2 = 324.1C4_{16}$
$10\quad 110\quad 001.011111\quad 01_2 = 261.372_8$

Conversion to base 2 from a base, which is an integer power of 2, can be most simply accomplished by independent conversion of each successive digit, as illustrated in Table 19.10a. Inversely, conversion from base 2 to a base 2^k can be simply accomplished by grouping the bits into sets of k bits, each starting with the least significant bit for the integer portion and with the most significant bit for the fraction portion, as shown by the examples in Table 19.10b.

19.7 Complements

Each number system has two conventionally used **complements**:

$$\text{Radix complement of } N = N^{RC} = R^n - N$$
$$\text{Reduced radix complement of } N = N^{rC} = N^{RC} - 1$$

where R is the radix and n is the number of digits in the number N. These equations provide complements for numbers having the magnitude N.

A positive number can be represented by a code in the two character machine language alphabet, 0 and 1, which is simply the positive number expressed in the base 2, that is, the code for the number is the number itself. A negative number requires that the sign be coded in the binary alphabet. This can be done by separately coding the sign and the magnitude or by coding the negative number as a single entity. Table 19.11 illustrates four different code types for negative numbers. Negative numbers can be represented in the sign magnitude form by using the leftmost digit as the code for the sign (0 for + and 1 for −) and the rest of the digits as the code for the magnitude. Complements and biasing provide the means for coding the negative number as a single entity instead of having a discrete separate coding for the sign. The use of complements provides for essentially equal ranges of values for both positive and negative numbers. The biased representation can also provide essentially equal ranges for positive and negative values by choosing the biasing value to be essentially half of the largest binary number that could be represented with the available number of digits. The bias code is obtained by subtracting the biasing value from the code considered as a positive number, as shown in the rightmost column of Table 19.11.

Complements enable subtraction to be done by addition of the complement. If the result fits into the available field size the result is automatically correct. A diagnostic must be provided to show that the result is incorrect if **overflow** occurs, that is, the number does not fit in the available field. Table 19.12 illustrates arithmetic operations with and without complements. The two rightmost columns illustrate cases where the result overflows the 3-b field size for the magnitude. The overflow condition can be represented in terms of two carry parameters:

- C_0, the output carry from the leftmost digit position
- C_1, the output carry from the second leftmost digit position (the output carry from the magnitude field if sign magnitude representation is used)

If both of these carries are coincident (i.e., have the same value) the result fits in the available field and, hence, is correct. If these two carries are not coincident, the result is incorrect.

TABLE 19.11 Number Representations

Available Codes	Positive Numbers	Signed Numbers Having the Specified Codes: Sign Magnitude	One's Complement	Two's Complement	111 bias
1111	1111	−111	−000	−001	+1000
1110	1110	−110	−001	−010	+111
1101	1101	−101	−010	−011	+110
1100	1100	−100	−011	−100	+101
1011	1011	−011	−100	−101	+100
1010	1010	−010	−101	−110	+011
1001	1001	−001	−110	−111	+010
1000	1000	−000	−111	−1000	+001
0111	0111	+111	+111	+111	000
0110	0110	+110	+110	+110	−001
0101	0101	+101	+101	+101	−010
0100	0100	+100	+100	+100	−011
0011	0011	+011	+011	+011	−100
0010	0010	+010	+010	+010	−101
0001	0001	+001	+001	+001	−110
0000	0000	+000	+000	+000	−111

TABLE 19.12 Comparison of Arithmetic with and without Complements

Sample Illustrations	$N = 7 - 5 = 2$	$N = 5 - 7 = -2$	$N = 5 + 7 = 12$	$N = -5 - 7 = -12$
Pencil and paper arithmetic (without complements)	111 −101 10	(−111) −(−101) −10	101 +111 1100	(−101) +(−111) −1100
Computer arithmetic with 2's complement	0111 +1011 /10010	0101 +1001 1110	0101 +0111 1100	1011 +1001 10100
4-binary digit working field (accommodates 3-b magnitude)	↓ Designates positive number	↓ Designates negative number	↓ Designates negative number	↓ Designates positive number
Result Veracity	+010 True	−010 True	−100 False	−100 False
Significant carries	$C_0 = 1$ $C_1 = 1$	$C_0 = 0$ $C_1 = 0$	$C_0 = 0$ $C_1 = 1$	$C_0 = 1$ $C_1 = 0$
Veracity condition or equivalently	$C_0 \equiv C_1$ $C_0 \odot C_1 = 1$	$C_0 \equiv C_1$ $C_0 \odot C_1 = 1$	$C_0 \not\equiv C_1$ $C_0 \odot C_1 = 0$	$C_0 \not\equiv C_1$ $C_0 \odot C_1 = 0$

19.8 Codes

Various types of **codes** have been developed for serving different purposes. There are codes that enable characters in an alphabet to be individually expressed in terms of codes in a smaller alphabet. For example, the alphabet of decimal numeric symbols can be expressed in terms of the binary alphabet by the **binary-coded decimal** (**BCD**) or 8421 code shown in Table 19.13. The 8421 designation represents the weight given to each of the binary digits in the coding process.

There are codes that facilitate doing arithmetic. The 2421 code can also be used to represent the decimal numeric symbols. The 2421 code has the advantage that the code for the reduced radix complement is

TABLE 19.13 Sample Codes

Decimal	BCD		2-out-	Parity		Gray		
Digits	8421	2421	of-5	Even	Odd	1-bit	2-bit	3-bit
0	0000	0000	00011	0000 0	0000 1	0	00	000
1	0001	0001	00101	0001 1	0001 0	1	01	001
2	0010	0010	00110	0010 1	0010 0		11	011
3	0011	0011	01001	0011 0	0011 1		10	010
4	0100	0100	01010	0100 1	0100 0			110
5	0101	1011	01100	0101 0	0101 1			111
6	0110	1100	10001	0100 1	0110 0			101
7	0111	1101	10010	0111 1	0111 0			100
8	1000	1110	10100	1000 1	1000 0			
9	1001	1111	11000	1001 0	1001 1			

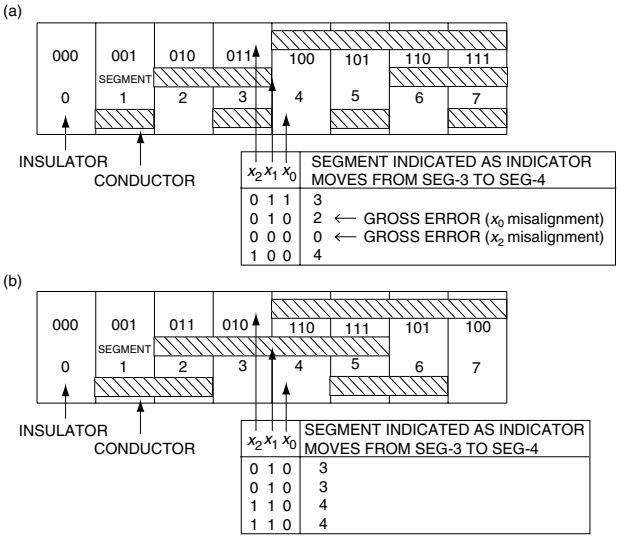


FIGURE 19.1 Eight-segment position sensor with slightly misaligned contacts: (a) binary code physical configuration, (b) gray code physical configuration.

the same as the reduced radix complement of the code, and this is not true of the BCD code. Thus, the 2421 code facilitates arithmetic with multiple individually coded digit numbers.

There are codes designed to detect errors that may occur in storage or transmission. Examples are the even and odd parity codes and the 2-out-of-5 code shown in Table 19.13. The 2-out-of-5 error detection code is such that each decimal value has exactly two high digit values. Parity code attaches an extra bit having a value such that the total number of high bits is odd if odd parity is used, and the total number of high bits is even if even parity is used. An even number of bit errors are not detectable by a single bit parity code. Hence, single bit parity codes are adequate only for sufficiently low bit error rates. Including a sufficient number of **parity bits** enables the detection and correction of all errors.

There are codes designed to prevent measurement misrepresentation due to small errors in sensor alignment. **Gray codes** are used for this purpose. A Gray code is one in which the codes for physically adjacent positions are also **logically adjacent**, that is, they differ in only one binary digit. Gray codes can be generated for any number of digits by reflecting the Gray code for the case with one less digit, as shown in Table 19.13, for the case of 1, 2, and 3-bit codes. The advantage of a Gray scale coded lineal position sensor is illustrated in Figure 19.1 for the eight-segment case.

19.9 Boolean Algebra

Boolean algebra provides a means to analyze and design binary systems and is based on the seven postulates given in Table 19.14. All other Boolean relationships are derived from these seven postulates. Expressed in graphical form, called *Venn diagrams*, the postulates appear more natural and logical. This benefit results from the two-dimensional pictorial representation freeing the expressions from the one-dimensional constraints imposed by lineal language format.

The OR and AND operations are normally designated by the arithmetic operator symbols + and · and referred to as sum and product operators in basic digital logic literature. However, in digital systems that perform arithmetic operations this notation is ambiguous and the symbols $\vee$ for OR and Λ for AND eliminates the ambiguity between arithmetic and boolean operators. Understanding the conceptual meaning of these boolean operations is probably best provided by set theory, which uses the union operator $\cup$ for OR and the intersection operator $\cap$ for AND. An element in a set that is the union of sets is a member of one set OR another of the sets in the union. An element in a set that is the intersection of sets is a member of one set AND a member of the other set in the intersection.

A set of theorems derived from the postulates facilitates further developments. The theorms are summarized in Table 19.15. Use of the postulates is illustrated by the proof of a theorem in Figure 19.2.

TABLE 19.14 Boolean Postulates

Postulate	Name	Meaning	Forms (a)	Forms (b)
1	De nition	$\exists$ aset $\{K\} = \{a,b,...\}$	**OR**	**AND**
		of two or more elements	+	·
		and two binary operators	$\vee$	Λ
		$\ni\{K\} = \{a, b, a+b, a\cdot b, \ldots\}$	$\cup$	$\cap$
2	Substitution Law	$\text{expression}_1 = \text{expression}_2$ If one replaced by the other does not alter the value		
3	Identity Element	$\exists$ identity elements for each operator	$a + 0 = a$	$a \cdot 1 = a$
4	Commutativity	For every a and b in K	$a + b = b + a$	$a \cdot b = b \cdot a$
5	Associativity	For every a, b, and c in K	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
6	Distributivity	For every a, b, and c in K	$a + (b \cdot c) = (a + b) \cdot (a + c)$	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
7	Complement	For every a in K $\exists$ acomplement in $K\ni$	$a + \bar{a} = 1$	$a \cdot \bar{a} = 0$

TABLE 19.15 Boolean Theorems

Theorem		Forms (a)	Forms (b)
8	Idempotency	$a + a = a$	$a \cdot a = a$
9	Complement Theorem	$a + 1 = 1$	$a \cdot 0 = 0$
10	Absorption	$a + ab = a$	$a(a + b) = a$
11	Extra Element Elimination	$a + \bar{a}b = a + b$	$a(\bar{a} + b) = ab$
12	De Morgan's Theorem	$a + b = \bar{a} \cdot \bar{b}$	$\overline{ab} = \bar{a} + \bar{b}$
13	Concensus	$ab + \bar{a}c + bc = ab + \bar{a}c$	$(a + b)(\bar{a} + c)(b + c) = (a + b)(\bar{a} + c)$
14	Complement Theorem 2	$ab + a\bar{b} = a$	$(a + b)(a + \bar{b}) = a$
15	Concensus 2	$ab + a\bar{b}c = ab + ac$	$(a + b)(a + \bar{b} + c) = (a + b)(a + c)$
16	Concensus 3	$ab + \bar{a}c = (a + c)(\bar{a} + b)$	$(a + b)(\bar{a} + c) = ac + \bar{a}b$

1)	$x + x = x + x$	IDENTITY
2)	$= (x + x) \cdot 1$	P3b IDENTITY ELEMENT EXISTENCE
3)	$= (x + x) \cdot (x + \bar{x})$	P7a COMPLEMENT EXISTENCE
4)	$= x + x \cdot \bar{x}$	P6a DISTRIBUTIVITY
5)	$= x + 0$	P7b COMPLEMENT EXISTENCE
6)	$= x$	P3a IDENTITY ELEMENT EXISTENCE

FIGURE 19.2 Proof of Theorem 8: Idempotency (a): $x + x = x$.

(a) $f(A, B, C) = AB + A\bar{C} + \bar{A}\bar{C}$

(b) $f(0, 0, 1) = 0 \cdot 0 + 0 \cdot \bar{1} + \bar{0} \cdot 1$
$= 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1$
$= 0 + 0 + 1$
$= 1$

(c)

A	B	C	f(A, B, C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

FIGURE 19.3 Example of forms for defining boolean functions: (a) boolean expression definition, (b) boolean expression evaluation, (c) truth table definition.

TABLE 19.16 Number of Different Boolean Functions

Variables n	Arguments 2^n	Functions 2^{2^n}
0	1	2
1	2	4
2	4	16
3	8	256
4	16	65,536
5	32	4,194,304
⋮	⋮	⋮

19.10 Boolean Functions

Boolean functions can be defined and represented in terms of boolean expressions and in terms of **truth tables** as illustrated in Figure 19.3a,c. Each form can be converted into the other form. The function values needed for the construction of the truth table can be obtained by evaluating the function as illustrated in Figure 19.3b. The reverse conversion will be illustrated subsequently.

For a given number of variables there are a finite number of boolean functions. Since each boolean variable can have two values, 0 or 1, a set of n variables has 2^n different values. A boolean function has a specific value for each of the possible values that the independent variables can have. Since there are two possible values for each value of the independent variables there are 2^{2^n} different boolean functions of n variables. The number of functions increases very rapidly with the number of independent variables, as shown in Table 19.16.

The 16 different boolean functions of the two independent variables are defined in algebraic form in Table 19.17 and in truth table form in Table 19.18.

19.11 Switching Circuits

Boolean functions can be performed by digital circuits. Circuits that perform complicated boolean functions can be subdivided into simpler circuits that perform simpler boolean functions. The circuits that perform the simplest boolean functions are taken as basic elements, called *gates* and are represented

TABLE 19.17 Functions of Two Variables De ned as Boolean Expressions

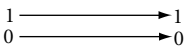
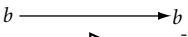
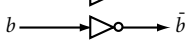
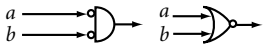
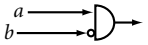
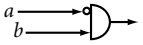
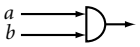
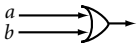
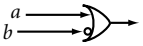
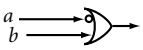
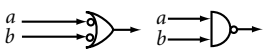
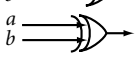
	Name	Expression	Circuit Representation
0	ALWAYS	1	
1	NEVER	0	
2	1st Var	a	
3	2nd Var	b	
4	NOT 1st Var	$\bar{a}$	
5	NOT 2nd Var	$\bar{b}$	
6	MIN-0/NOR	$\bar{a}\Lambda\bar{b} = a \downarrow b$	
7	MIN-1	$a\Lambda\bar{b}$	
8	MIN-2	$\bar{a}\Lambda b$	
9	MIN-3/AND	$a\Lambda b$	
10	MAX-0/OR	$a\vee b$	
11	MAX-1	$a\vee\bar{b}$	
12	MAX-2	$\bar{a}\vee b$	
13	MAX-3/NAND	$\bar{a}\vee\bar{b} = a \uparrow b$	
14	EXOR	$A \oplus b = a\Lambda\bar{b}\vee\bar{a}\Lambda b$	
15	COIN	$a \ominus b = \overline{a \oplus b}$	

TABLE 19.18 Truth Tables for Two Variable Functions

		NOR			AND	OR			NAND								
a	b	m_0	m_1	m_2	m_3	M_0	M_1	M_2	M_3	XOR	COIN	a	b	$\bar{a}$	$\bar{b}$	LO	HI
0	0	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1
0	1	0	1	0	0	1	0	1	1	1	0	0	1	1	0	0	1
1	0	0	0	1	0	1	1	0	1	1	0	1	0	0	1	0	1
1	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1

by specialized symbols. The circuit symbols for the gates that perform functions of two independent variables are shown in Table 19.17. The gates are identified by the adjective representing the operation they perform. The most common gates are the AND, OR, NAND, NOR, *X*OR, and COIN gates. The only nontrivial single input gate is the invertor or NOT gate. Gates are the basic elements from which more complicated digital logic circuits are constructed.

A logic circuit whose steady-state outputs depend only on the present steady-state inputs (and not on any prior inputs) is called a *combinational logic circuit*. To depend on previous inputs would require memory, thus a combinational logic circuit has no memory elements.

Boolean algebra allows any combinational logic circuit to be constructed soley with AND, OR, and NOT gates. Any combinational logic circuit may also be constructed solely with NAND gates, as well as solely with NOR gates.

19.12 Expansion Forms

The **sum of products** (**SP**) is a basic form in which all boolean functions can be expressed. The **product of sums** (**PS**) is another basic form in which all boolean functions can be expressed. An illustrative example is given in Figures 19.4b,c for the example given in Figure 19.4a.

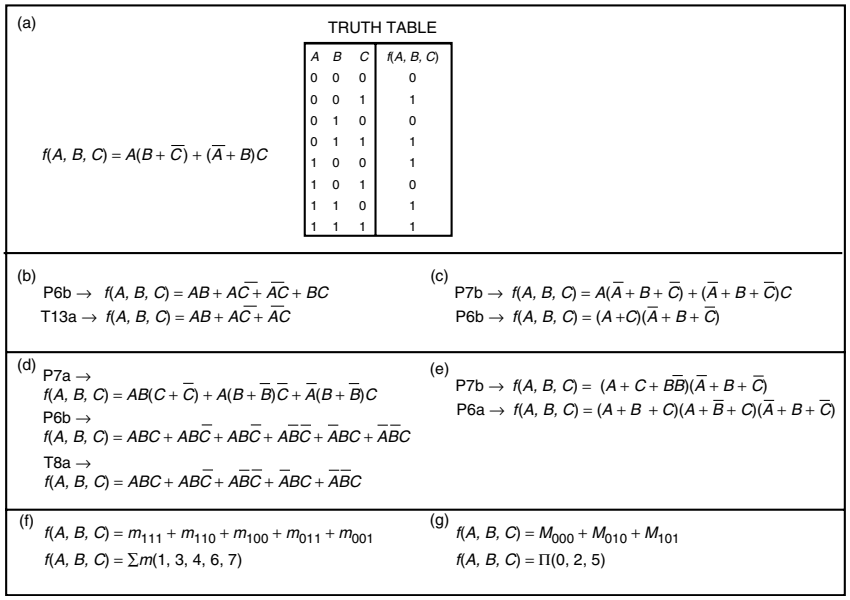


FIGURE 19.4 Examples of converting boolean functions between forms: (a) given example, (b) conversion to SP form, (c) conversion to PS form, (d) conversion to canonical SP form, (e) conversion to canonical PS form, (f) minterm notation/canoncial SP form, (g) maxterm notation/canonical PS form.

Minterms are a special set of functions, none of which can be expressed in terms of the others. Each minterm has each of the variables in the complemented or the uncomplemented form ANDed together. An SP expansion in which only minterms appear is a canonical SP expansion. Figure 19.4d shows the development of the canonical SP expansion for the previous example. The canonical SP expansion may also be simply expressed by enumerating the minterms as shown in Figure 19.4f. Comparison of the truth table with the minterm expansion shows that each function value of 1 represents a minterm of the function and vice versa. All other function values are 0.

Maxterms are a special set of functions, none of which can be expressed in terms of the others. Each maxterm has each of the variables in the complemented or the uncomplemented form ORed together. A PS expansion in which only maxterms appear is a canonical PS expansion. Figure 19.4e shows the development of the canonical PS expansion for the previous example. The canonical PS expansion may also be simply expressed by enumerating the maxterms as shown in Figure 19.4g. Comparison of the truth table with the maxterm expansion shows that each function value of 0 represents a maxterm of the function and vice versa. All other function values are 1.

19.13 Realization

The different types of boolean expansions provide different circuits for implementing the generation of the function. A function expressed in the SP form is directly realized as an AND–OR **realization**, as illustrated in Figure 19.5a. A function expressed in the PS form is directly realized as an OR–AND realization as illustrated in Figure 19.5b. By using involution and deMorgan's theorm the SP expansion can be expressed in terms of NAND–NAND and the PS expansion can be expressed in terms of NOR–NOR, as shown in Figures 19.5c,d. The variable inversions specified in the inputs can be supplied by either NAND or NOR gates, as shown in Figures 19.5g,h, which then provide the NAND–NAND–NAND and the NOR–NOR–NOR circuits shown in Figures 19.5i,j.

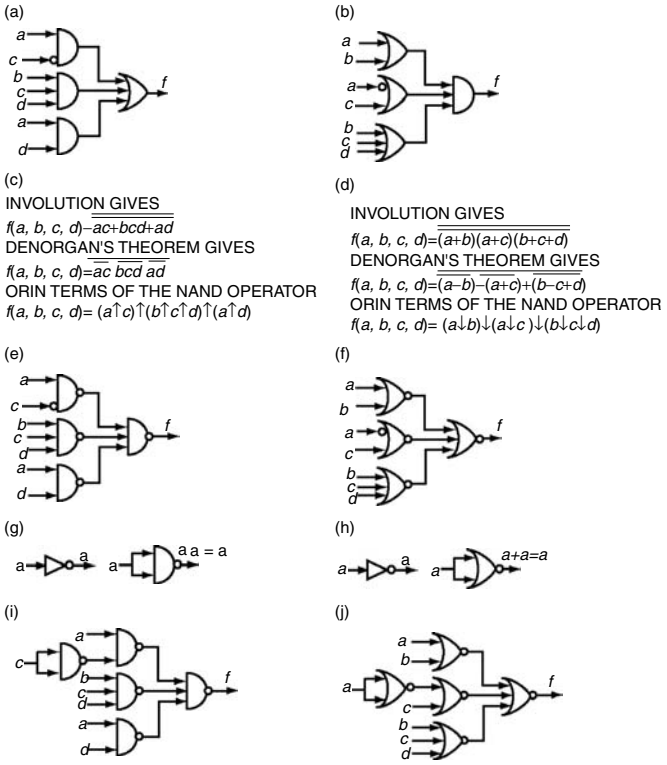


FIGURE 19.5 Examples of realizations based on various expansion forms: (a) AND-OR realization of $f(a, b, c, d) = a\bar{c} + bcd + ad$, (b) OR-AND realization of $f(a, b, c, d) = (a + b)(\bar{a} + c)(b + c + d)$, (c) AND-OR conversion to NAND-NAND, (d) OR-AND conversion to NOR-NOR, (e) NAND-NAND realization of $f(a, b, c, d) = a\bar{c} + bcd + ad$, (f) NOR-NOR realization of $f(a, b, c, d) = (a + b)(\bar{a} + c)(b + c + d)$, (g) NAND gate realization of NOT gate, (h) NOR gate realization of NOT gate, (i) NAND-NAND-NAND realization of $f(a, b, c, d) = a\bar{c} + bcd + ad$, (j) NOR-NOR-NOR realization of $f(a, b, c, d) = (a + b)(\bar{a} + c)(b + c + d)$.

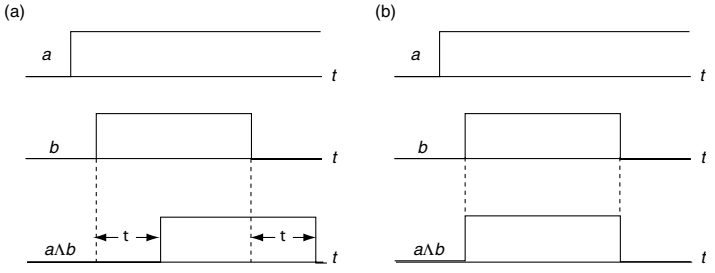


FIGURE 19.6 Timing diagrams for the AND gate circuit: (a) microtiming diagram, (b) macrotiming diagram.

19.14 Timing Diagrams

Timing diagrams are of two major types. A **microtiming diagram** has a time scale sufficiently expanded in space to display clearly the gate delay, such as shown in Figure 19.6a for an AND gate. A **macrotiming diagram** has a time scale sufficiently contracted in space so that the gate delay is not noticeable, as shown in Figure 19.6b for an AND gate.

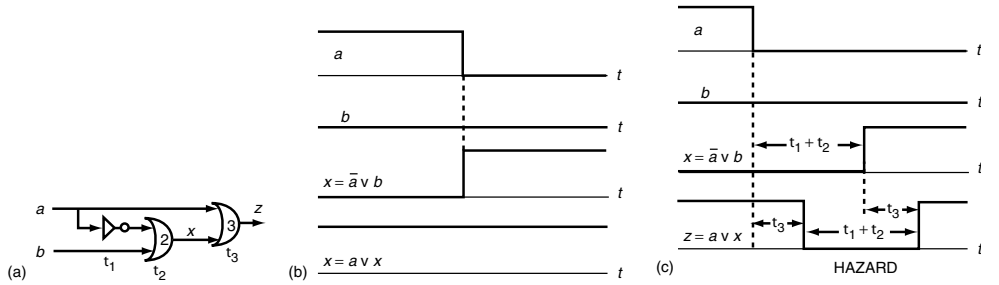


FIGURE 19.7 Example of a hazard (output variation) caused by unequal delay paths: (a) circuit for illustrating a hazard, (b) ideal case, no delays, $\tau_1 = \tau_2 = \tau_3 = 0$, no hazard introduced; (c) signal paths with different delays, $\tau_1 + \tau_2 + \tau_3 > \tau_3$, hazard introduced.

The advatange of the macrotiming diagram is that larger time intervals can be represented in a given spatial size and they can be more quickly developed. The disadvantage is that they do not display the information required for speed limitation considerations.

19.15 Hazards

The variation in signal delays through different circuit elements in different paths may cause the output signal to fluctuate from that predicted by non-time-dependent truth tables for the elements. This fluctuation can cause an undesired result and, hence, is a *hazard*. This is illustrated in Figure 19.7.

19.16 *K*-Map Formats

In a truth table the values of a boolean function are displayed in a one-dimensional array. A ***K*-map** contains the same information arranged in as many effective dimensions as there are independent variables in the function. The special form of the representation provides a simple procedure for minimizing the expression and, hence, the number of components required for realizing the function in a given form. The function is represented in a space that in a Venn diagram is called the *universal set*. The *K*-map is a special form of Venn diagram. The space is divided into two halves for each of the independent variables. The division of the space into halves is different for each independent variable. For one independent variable the space is divided into two different identical size regions, each of which represents a minterm of the function. For n independent variables the space is divided into 2^n different identical size regions, one for each of the 2^n minterms of the function. This and associated considerations are illustrated in a sequence of figures from Figures 19.8–19.15.

Figure 19.8 shows one-variable *K*-map formats. Figure 19.8a shows the space divided into two equal areas, one for each of the two minterms possible for a single variable. The squares could also be identified by the variable placed external to the space to designate the region that is the domain of the variable, with the unlabeled space being the domain of the complement of the variable, as shown in Figure 19.8b. Another way of identifying the regions is by means of the minterm number the area is for, as shown in Figure 19.8c. Still another way is to place the values the variable can have as a scale alongside the space, as shown in Figure 19.8d. The composite labeling, shown in Figure 19.8e, appears redundant but is often useful because of the different modes of thought used with the different label types. Putting the actual minterm expressions inside each square is too cluttering and is rarely used except as an aid in teaching the theory of *K*-maps. The use of minterm numbers, although widely used, also clutters up the diagram, and the methodology presented here makes their use superfluous once the concepts are understood.

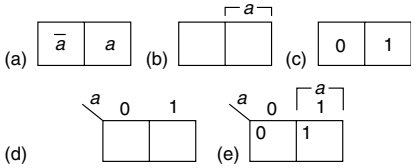


FIGURE 19.8 One vartiable *K*-map forms: (a) internal minterm labels, (b) external domain label, (c) internal minterm number labels, (d) external scale label, (e) composite labeling.

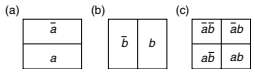


FIGURE 19.9 Two-variable *K*-map format construction: (a) domains for ***a***, (b) domains for ***b***, composite.

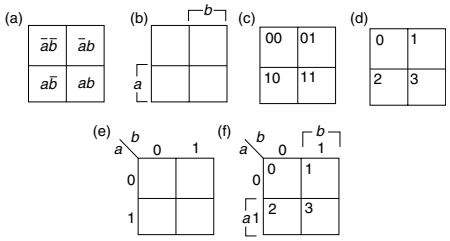


FIGURE 19.10 Two variable *K*-map formats: (a) minterm labels, (b) domain labels, (c) minterm binary labels, (d) minterm decimal labels, (e) scale labels, (f) composite labeling.

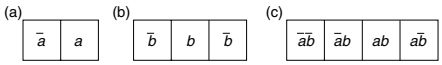


FIGURE 19.11 Two-variable *K*-map alternate format construction: (a) domains for ***a***, (b) domains for ***b***, (c) composite.

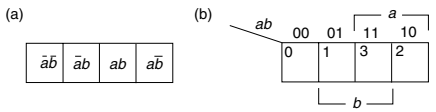


FIGURE 19.12 Two-variable *K*-map alternate format: (a) minterm labels, (b) composite labeling.

The organization of a two-variable *K*-map format is illustrated in Figure 19.9. The space is subdivided vertically into two domains for the variable ***a*** and its complement, and is subdivided horizontally for the variable ***b*** and its complement, as shown in Figures 19.9a,b. The composite subdivisions for both variables together with the expressions for the two-variable minterms are shown in Figure 19.9c.

The different formats for identifying the areas for the two-variable case are shown in Figure 19.10. Of particular interest is the comparison of the binary and decimal minterm number labels. The binary minterm number is simply the **catenation** of the vertical and horizontal scale number for the position. It is the use of this identity that makes internal labels in the square totally superfluous.

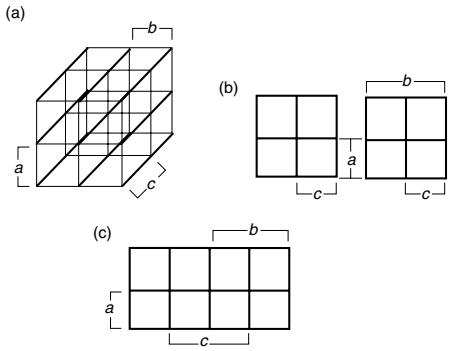


FIGURE 19.13 Three-variable *K*-map formats: (a) three-dimensional, (b) three-dimensional left and right halves, (c) two-dimensional.

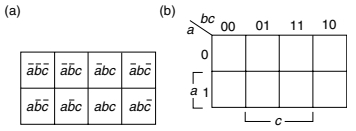


FIGURE 19.14 Three variable two-dimensional *K*-map formats: (a) minterm labels, (b) composite scale labels.

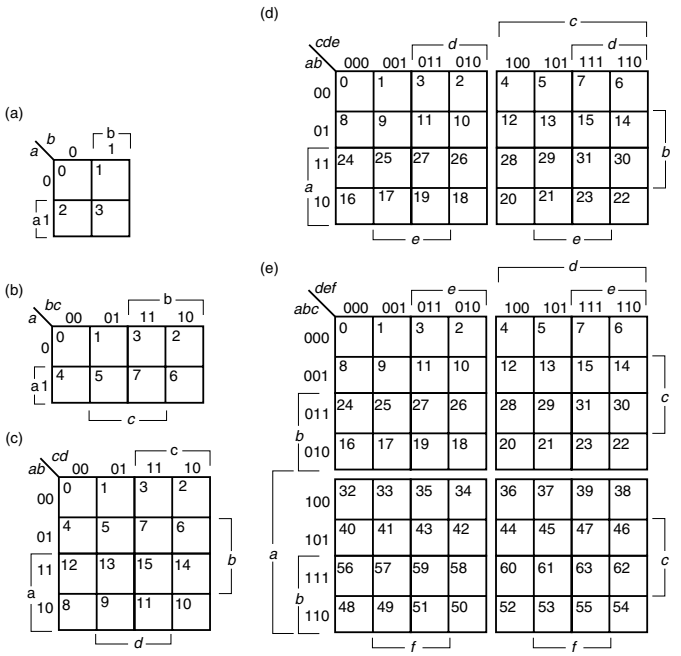


FIGURE 19.15 Formats for *K*-maps for functions of 2–6 independent variables with conformal coordinate scales: (a) two-variable case $f(a, b)$, three-variable case $f(a, b, c)$; (c) four-variable case $f(a, b, c, d)$; (d) five-variable case $f(a, b, c, d, e)$; (e) six-variable case $f(a, b, c, d, e, f)$.

An alternate way of subdividing the space for the two-variable case is illustrated in Figure 19.11 and labeling alternatives in Figure 19.12. The configuration employed in Figure 19.9 uses two dimensions for two variables, whereas the configuration employed in Figure 19.11 uses one-dimension for two variables. The two-dimensional configuration appears to be more logical and is more convenient to use than the one-dimensional configuration for the case of two variables. For the case of a larger number of variables the configuration in Figure 19.12 offers special advantages, as will be shown.

The organization of three-variable *K*-map formats is illustrated in Figure 19.13. It is logical to introduce an additional space dimension for each additional independent variable as depicted in Figure 19.13a; however, the excessive inconvenience of working with such formats makes them impractical. To make the mapping process practical it must be laid out in two dimensions. This can be done in two ways. One way is to take the individual slices of the three-dimensional configuration and place them adjacent to each other as illustrated in Figure 19.13b. Another way is to use the one-dimensional form for two variables, illustrated in Figure 19.12, as shown in Figure 19.13c. For the case of three and four independent variables the format given in Figure 19.13c is more convenient and for further independent variables that of Figure 19.13b is convenient. These are all illustrated in Figure 19.15. Labeling for three independent variables is given in Figure 19.14.

The independent boolean variables in **conformal coordinate scales** have exactly the same order as in the boolean function argument list, as depicted in Figure 19.15. Conformal assignment of the independent variables to the *K*-map coordinate scales makes the catenated position coordinates for a minterm (or maxterm) identical to the minterm (or max-term) number. Utilization of this identity eliminates the need for the placement of minterm identification numbers in each square or for a separate position identification table. This significantly decreases the time required to construct *K*-maps and makes their construction less error prone. The minterm number, given by the catenation of the vertical and horizontal coordinate numbers, is obvious if the binary or octal number system is used.

19.17 *K*-Maps and Minimization

A function is mapped into the *K*-map format by entering the value for each of the minterms in the space for that minterm. The function values can be obtained in various ways such as from the truth table for the function, the Boolean expression for the function, or from other means by which the function may be defined. An example is given for the truth table given in Figure 19.4a, which is repeated here in Figure 19.16a and whose *K*-map is shown in various forms in Figures 19.16b–d.

The function can also be mapped into a conformally scaled *K*-map directly from the canonical expansion, this being essentially the same process as entering the minterms (or maxterms) from the truth table. The function may also be directly mapped from any PS or SP expansion form. Another means of obtaining the *K*-map is to formulate it as a function table, as illustrated for the multiplication of 1- and 2-b numbers in Figure 19.17.

(a)

A	*B*	*C*	*F*(*A*, *B*, *C*)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(b)

A \ *BC*	00	01	11	10
0	0	1	1	0
1	1	0	1	1

B (11, 10); *C* (01, 11)

(c)

A \ *BC*	00	01	11	10
0		1	1	
1	1		1	1

B (11, 10); *C* (01, 11)

(d)

A \ *BC*	00	01	11	10
0	0			0
1		0		

B (11, 10); *C* (01, 11)

FIGURE 19.16 Three-variable *K*-map example: (a) truth table, (b) *K*-map with all values shown, (c) minterm *K*-map, (d) maxterm *K*-map.

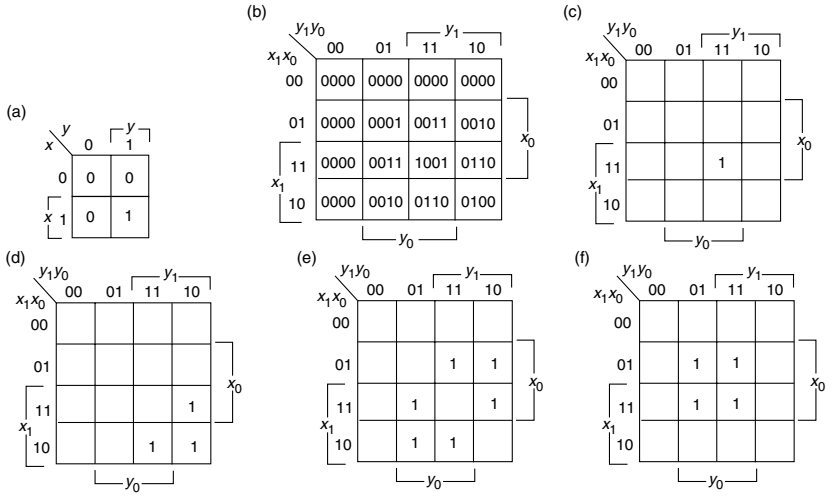


FIGURE 19.17 Examples of *K*-maps formulated as function tables: (a) *K*-map for the product of two 1-b numbers $P = x * y$; (b) composite *K*-map for the product of two 2-b numbers, $P_3\ P_2\ P_1\ P_0 = x_1\ x_0\ *\ y_1\ y_0$; (c) *K*-map for the digit P_3 of the product of two 2-b numbers; (d) *K*-map for the digit P_2 of the product of two 2-b numbers; (e) *K*-map for the digit P_1 of the product of two 2-b numbers; (f) *K*-map for the digit P_0 of the product of two 2-b numbers.

19.18 Minimization with *K*-Maps

The key feature of *K*-maps that renders them convenient for minimization is that minterms, which are spatially adjacent in the horizontal or vertical directions, are logically adjacent. Logically adjacent minterms are identical in all variables except one. This allows the two minterms to be combined into a single terms with one less variable. This is illustrated in Figure 19.18. Two adjacent minterms combine into a first-order **implicant**. A first-order implicant is the combination of all of the independent variables but one. In this example, the first-order implicant expressed in terms of minterms contains eight literals but the minimized expression contains only three literals. The circuit realization for the OR combination of the two minterms has two AND gates and one OR gate, whereas the realization for the equivalent implicant requires only a single AND gate.

The combination of minterms into first-order implicants can be represented more compactly by using the single symbol minterm notation with the subscript that identifies the particular minterm expressed in binary, as illustrated in Figure 19.18d.

Two adjacent first-order implicants can be combined into a second-order implicant as illustrated in Figure 19.19. A second-order implicant contains all of the independent variables except two. In general, an *n*th-order implicant contains all of the variables except *n* and requires an appropriately grouped set of 2^n minterms.

Minterms that are at opposite edges of the same row or column are logically adjacent since they differ in only one variable. If the plane space is rolled into a cylinder with opposite edges touching, then the logically adjacent edge pairs become physically adjacent. For larger numbers of variables using *K*-maps with parallel sheets, the corresponding positions on different sheets are logically adjacent. If the sheets are considered as overlaid, the corresponding squares are physcally adjacent. The minimized expression is obtained by covering all of the minterms with the fewest number of largest possible implicants. A minterm is a zero-order implicant. Figure 19.20 illustrates a variety of examples. A **don't care** is a value that never occurs or if it does occur it is not used, and hence, it does not matter what its value is. Don't cares are also included to illustrate that they can be used to simplify expressions by taking their values to maximize the order of the implicants.

Maxterm *K*-maps can also be utilized to obtain minimized expressions by combining maxterms into higher order implicants, as illustrated for the example in Figure 19.21.

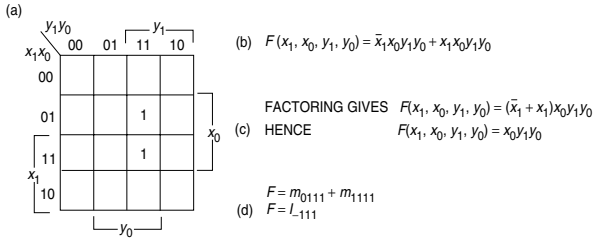


FIGURE 19.18 Example of minimization with a *K*-map: (a) sample *K*-map, (b) expression in minterms of function definition in (a), (c) simplification of the expression in (b), (d) simplification of the expression in (b) using single symbol minterm and implicant notation.

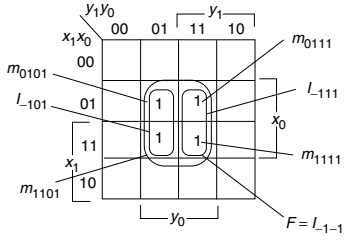


FIGURE 19.19 Example of minimization with *K*-map.

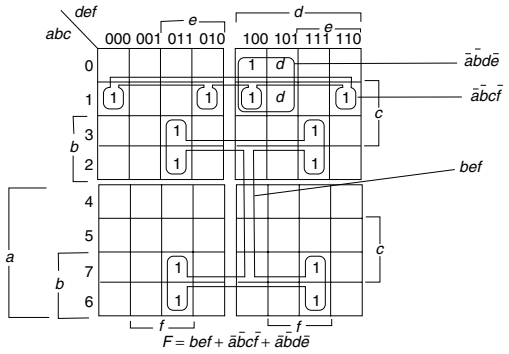


FIGURE 19.20 Example of minimization with six-variable *K*-map.

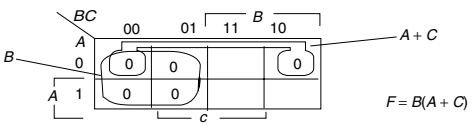


FIGURE 19.21 Three-variable maxterm *K*-map example.

GIVEN $F(A, B, C, D) = \Sigma m(2, 6, 7, 8) + d(0, 4, 5, 12, 13)$ and $G(A, B, C, D) = \Sigma m(2, 4, 5) + d(6, 7, 8, 10)$

ZERO-ORDER IMPLICANT LIST

NO. OF 1s	MIN-TERM	CODE *ABCD*	FLAGS	PI
0	0	0000	F-	√
	2	0010	FG	√
1	4	0100	FG	√
	8	1000	FG	√
	5	0101	FG	√
2	6	0110	FG	√
	10	1010	-G	√
	12	1100	F-	√
	7	0111	FG	√
3	13	1101	F-	√

FIRST-ORDER IMPLICANT LIST

IMPLICANTS	CODE *ABCD*	FLAGS	PI
0, 2	00-0	F-	√
0, 4	0-00	F-	√
0, 8	-000	F-	√
2, 6	0-10	FG	5
2, 10	-010	-G	6
4, 5	010-	FG	√
4, 6	01-0	FG	√
4, 12	-100	F-	√
8, 10	10-0	-G	7
8, 12	1-00	F-	√
5, 7	01-1	FG	√
5, 13	-101	F-	√
6, 7	011-	FG	√
12, 13	110-	F-	√

SECOND-ORDER IMPLICANT LIST

IMPLICANTS	CODE *ABCD*	FLAGS	PI
0, 2, 4, 6	0-0	F-	1
0, 4, 8, 12	-00	F-	2
4, 5, 6, 7	01-	FG	3
4, 5, 12, 13	-10-	F-	4

PRIME IMPLICANT CHART

	m / PI	F 2	F 6	F 7	F 8	G 2	G 4	G 5
	1	×	×					
*	2				×			
*	3		×	×			×	×
	4							
**	5	×	×				×	
	6						×	
	7							

MINIMUM SP EXPANSIONS

$F = PI_2 + PI_3 + PI_5$
$= \overline{C}\overline{D} + \overline{A}B + \overline{A}C\overline{D}$

$G = PI_3 + PI_5$
$= \overline{A}B + \overline{A}C\overline{D}$

FIGURE 19.22 Illustration of the Quine–McClusky method of simultaneous minimization.

19.19 Quine–McCluskey Tabular Minimization

The *K*-map minimization method is too cumbersome for more than six variables and does not readily lend itself to computerization. A tabular method, which can be implemented for any number of variables and which lends itself to computer program implementation, consists of the following steps:

1. List all the minterms in the boolean function (with their binary code) organized into groups having the same number of 1s. The groups must be listed in consecutive order of the number of 1s.
2. Construct the list of first-order implicants. Use flags to indicate which minterms, don't cares, or implicants go with which functions. (Only minterms in adjacent groups have the possibility of being adjacent and, hence, this ordering method significantly reduces the labor of compiling the implicants.)
3. Construct the list of second-order implicants and the lists of all higher order implicants, until no higher order implicants can be constructed.
4. Construct the *prime implicant chart*. The prime implicant chart shows what prime implicants cover which minterms.
5. Select the minimum number of largest prime implicants that cover the minterms.

This procedure is illustrated in Figure 19.22 for the simultaneous minimization of two boolean functions.

Defining Terms

Base: The number of different values a single digit may have. The number a digit must be multiplied by to move it one digit to the left, also called the radix.

Binary-coded decimal (BCD): Each decimal digit is expressed individually in binary form.

Catenation: Symbols strung together to form a larger sequence, as the characters in a word and the digits in a number.

Code: The representation in one alphabet of something in another alphabet.

Complement: The quantity obtained by subtracting a number from the largest quantity that can be expressed in the specified number of digits in a given number system.

Conformal: The same arrangment of a set of quantities in two different contexts.
Digit: A character that represents quantitative information.
Don't care: A value that can be represented either as a minterm or a maxterm.
Fraction: Any number divided by a larger number.
Gray code: A set of codes having the property of logical adjacency.
Implicant: A first-order implicant is a pair of logically adjacent minterms. A second-order implicant is a set of logically adjacent first-order implicants and so on.
Integer: Any number that can be expressed solely in terms of digits.
***K*-map:** An arrangment of space into equal size units, each of which represents a minterm (or maxterm) such that each physically adjacent square is also logically adjacent.
Logically adjacent: Any two codes having the same number of digits for which they differ in the value of only one of the digits.
Macrotiming diagram: A graphical display showing how the waveforms vary with time, but with a time scale that does not have sufficient resolution to display the delays introduced by the individual basic elements of the digital circuit.
Maxterm: A function of a set of boolean variables that has a low value for only one combination of variable values and has a high value for all other combinations of the variable values.
Microtiming diagram: A graphical display showing how the waveforms vary with time, but with a time scale that has sufficient resolution to display clearly the delays introduced by the individual basic elements of the digital circuit.
Minterm: A function of a set of boolean variables that has a high value for only one combination of variable values and has a low value for all other combinations of the variable values.
Overflow: That part of a numerical operation result that does not fit into the allocated field.
Parity bit: An extra bit catenated to a code and given a value such that the total number of high bits is even for even parity and odd for odd parity.
Prime implicant: An implicant that is not part of a larger implicant.
Product of sums (PS): The AND combination of terms, which are OR combinations of boolean variables.
Radix: The number of different values that a digit can have in a number system.
Realization: A circuit that can produce the value of a function.
Real number: A number that has a fractional part and an integer part.
Reduced radix: The largest value a digit can have in a number system. It is one less than the radix.
Sum of products (SP): The OR combination of terms, which are AND combinations of Boolean variables.
Truth table: The table of values that a boolean function can have for which the independent variables considered as a multidigit number are arranged in consecutive order.

References

Hayes, J.P. 1993. *Introduction of Digital Logic Design.* Addison-Wesley, Reading, MA.
Humphrey, W.S., Jr. 1958. *Switching Circuits with Computer Applications.* McGraw-Hill, New York.
Hill and Peterson. 1974. *Introduction to Switching Theory and Logical Design,* 2nd ed. Wiley, New York.
Johnson and Karim. 1987. *Digital Design a Pragmatic Approach.* Prindle, Weber and Schmidt, Boston.
Karnaugh, M. 1953. The map method for synthesis of combinational logic circuits. *AIEE Trans. Comm. Elec.* 72 (Nov.): 593–599.
Mano, M.M. 1991. *Digital Design.* Prentice-Hall, Englewood Cliffs, NJ.
McClusky, E.J. 1986. *Logic Design Principles.* Prentice-Hall, Englewood Cliffs, NJ.
Mowle, F.J. 1976. *A Systematic Approach to Digital Logic Design.* Addison-Wesley, Reading, MA.
Nagle, Carrol, and Irwin. 1975. *An Introduction to Computer Logic,* 2nd ed. Prentice-Hall, Englewood Cliffs, NJ.
Pappas, N.L. 1994. *Digital Design* West, St. Paul, MN.
Roth, C.H., Jr. 1985. *Fundamentals of Logic Design,* 3rd ed. West, St. Paul, MN.
Sandige, R.S. 1990. *Modern Digital Design.* McGraw-Hill, New York.

Shaw, A.W. 1991. *Logic Circuit Design*. Saunders, Fort Worth, TX.
Wakerly, 1990. *Digital Design Principles and Practices*. Prentice-Hall, Englewood Cliffs, NJ.

Further Information

Further information on basic logic concepts and combinational logic design can be found in occasional articles in the following journals:

Lecture Notes in Computer Science (annual)
International Journal of Electronics (monthly)
IEEE Transactions on Education (quarterly)
IEEE Transactions on Computers (monthly)
IEEE Transactions on Software Engineering (monthly)
IEEE Transactions on Circuits and Systems 1. Fundamental Theory and Applications (monthly)

20

System Interfaces

Michael J. Tordon
Jayantha Katupitiya
The University of New South Wales

This chapter deals with asynchronous serial interfaces described by interface standards RS-232, RS-422, and RS-485 and with the general-purpose parallel interface bus described by IEEE-488 standard. The chapter also provides background information, terminology and parameters, which are important in the design of system interfaces for mechatronic systems.

20.1 Background

Modern mechatronic systems comprise a number of subsystems, which rely heavily on digital data communications. Different levels of complexity of these systems means that the requirement for data communications range from a simple communication between two devices to systems with a large number of subsystems, where each subsystem communicates directly or indirectly with other subsystems using a communication network. Depending on the proximity of subsystems, different requirements are placed on data communication channels, the physical implementation of channels, and interfaces between these devices. Figure 20.1 shows a schematic diagram of a simple data communication system connecting two devices.

A data source creates the data to be transmitted to the destination system and may convert the data into a specific form. The originating system usually does not create the data in a form suitable for transmission over transmission lines. This is left to the transmitter, which transforms the data into a signal suitable for transmission over a specific type of transmission line. The transmission line is generally implemented using electrical wiring but can involve a variety of physical medium including radio frequency, infrared, and sound

FIGURE 20.1 A schematic diagram of a simple data communication system.

signals. A transmission line provides a physical medium connecting the two systems. A receiver accepts the signal and converts it to the data form suitable to be passed onto the destination system. A data destination processes the data in order to recover the original information. From the previous information, it follows that even in the case of a simple data communication system a number of subsystems is involved in the communication task.

20.1.1 Terminology and Definitions

Interface: The common boundary between two subsystems is called an interface, and as can be seen in Figure 20.1, a number of interfaces can be involved even in a simple communication system.

Bit: The simplest form of data is one bit, which can take one of the two values 0 or 1, and hence is called binary data. All information in modern digital computers is stored in binary form.

Byte: A fixed number of bits (usually 8), which can be treated by a computer as a unit.

Character: Historically, the information is expressed in terms of characters. A character is a member of a character set. An example of a character set is the set of characters in the English language.

Character code: Individual characters from the selected character set are encoded in digital computers as binary numbers. One of the most widely used character set codes is the American Standard Code for Information Interchange (ASCII).

20.1.2 Serial versus Parallel

The basic unit of information to be transferred between subsystems is usually a character. For short distances, multiple parallel lines can be used to carry out simultaneous transmission of all the bits of a character. For the transmission of data over long distances, the cost of multiple data lines is often prohibitive and it is normal to serialize the data so that it can be passed over a single data path as a stream of bits.

20.1.3 Bit Rate versus Baud Rate

The speed of data transmission is usually expressed as a number of data bits transmitted per second and is called an effective bit rate with a unit bps. Larger units like kbps (1,000 bps) and Mbps (1,000,000 bps) are commonly used. The baud rate is a signaling rate and is expressed as a number of times per second that the signal transmitted over a data transmission line changes state. For systems using only two states, the signaling bit rate is equivalent to the baud rate. Distinction should be made between the effective data transmission bit rate and the signaling bit rate. In asynchronous serial communications, the effective data transmission bit rate can be significantly lower than the signaling bit rate because of the inclusion of start, stop, and parity bits. To maximize the transmission speed over a serial line, modern communication systems use signals with more than two states, thus achieving higher signaling bit rates. For example, if the transmission signal uses 16 states, then the signaling bit rate is four times higher than the baud rate. The terms baud rate, signaling bit rate, and effective data transmission rate are often used interchangeably which leads to confusion.

20.1.4 Synchronous versus Asynchronous

For both parallel and serial interface, the problem of synchronization must be solved. The communication over a transmission line can be done either in synchronous or asynchronous communication mode. In synchronous communication mode, the transmission of data is synchronized with a clock; thus, the transmission is occurring at regular time intervals. Since the data transmission takes place at fixed times, the

completion of data transfer does not have to be acknowledged. In asynchronous transmission mode, the two systems are using clocks, that are not synchronized and may run at frequencies slightly out of step. Thus, for asynchronous systems, data validation requires a separate scheme called handshaking.

20.1.5 Data Flow-Control

Another problem in asynchronous communication systems is the speed of data processing. If one system is significantly slower in processing the data, a flow-control must be implemented to avoid data loss. Data flow-control may require additional handshaking. Similar problems may arise in multitasking systems in which, due to other tasks, the system is unable to handle incoming data during the period of high workload.

20.1.6 Handshaking

In order to ensure efficient transmission of data without errors, the sending system will use a separate signal to indicate that valid data has been presented to the interface. Because the instant at which the receiving device can process the data is not known, the sending device must wait for an acknowledgment signal before presenting new data to the interface. The handshaking can be implemented in either hardware or software.

20.1.7 Communication Protocol

Operation of a communication system is governed by a set of rules which must ensure reliable data transfer without errors and data loss. Such a set of rules is called a communication protocol.

20.1.8 Error Handling

Data transmitted over a communication line are subjected to noise and can thus be corrupted. Since it is essential to maintain the integrity of data, a number of different schemes for error detection have been developed. The simplest remedy after error detection is retransmission of the corrupted data. More sophisticated communication protocols can involve complex error correction schemes implemented at protocol level.

20.1.9 Simplex, Half-Duplex, Full-Duplex

In its simplest form, communication can be established with a single pair of wires. The data transmission mode, in which data can pass in one direction only, is called simplex or unidirectional channel. In most applications it is required that the communication takes place in both directions. If the cost of the data transmission line is high, it can be arranged that signals can pass in either direction over a single transmission line using additional circuitry on both ends of the transmission line but only in one direction at a time. This type of data communication mode is called half-duplex. Additional handshaking is required to implement the time sharing of the transmission line.

If signals can pass in either direction over a single transmission line simultaneously, the data communication mode is called full-duplex. An example of a full-duplex is a telephone line where the two channels are created as separate frequency bands. Cost permitting, two separate transmission lines can be established in which case the full-duplex communication is conducted over two simplex channels. This requires duplication of all the functions of a simple data communication system as shown in Figure 20.1.

20.1.10 Unbalanced versus Balanced Transmission

Implementation of the electrical transmission line can take two basic forms, unbalanced (single-ended) or balanced (differential). For unbalanced operation, a single conductor is used to carry the signal voltage, which is referenced to a signal ground. The signal ground is usually common return for all signals in the interface. Figure 20.2 shows an example of an unbalanced data transmission system with two channels and three wires. Symbol D represents driver and symbol R receiver. Unbalanced data transmission is

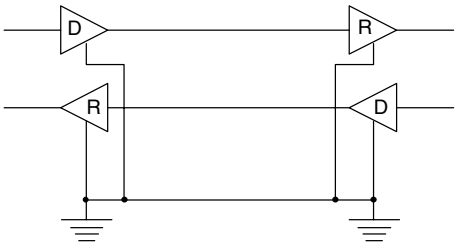


FIGURE 20.2 Example of an unbalanced data transmission.

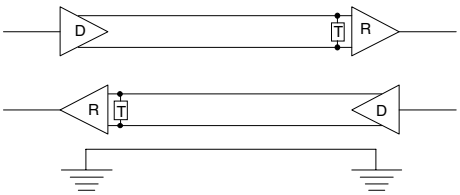


FIGURE 20.3 Example of a balanced data transmission.

relatively inexpensive because, for multiple signal lines, only one common line is required; however, this type of interface is susceptible to induced and ground noise and is not suitable for high-speed communication over long distances. The ground noise is associated with voltage drop in a common return line, while the induced noise comes from interfering electromagnetic fields. Both types of noise can come from external sources or from neighboring transmission circuits. A remedy can be the use of coaxial cable, shielded cable, and/or the use of separate return lines for individual signals. These additional measures tend to increase the cost of the interface.

The balanced (differential) transmission mode has much better noise immunity than the unbalanced mode. Two complementary signal lines carry the data signal. The implementation often involves two single-ended drivers driving a twisted-pair transmission line. Figure 20.3 shows an example of balanced data transmission with two channels and five wires. As in Figure 20.2, symbol D represents driver and symbol R receiver. Symbol T represents termination resistor. Use of a termination resistor at the receiver end of the transmission line is critical for high-speed communications over long distances as unterminated transmission lines can cause severe distortion of signals. Both induced and ground noises appear on both conductors as common-mode signals that are rejected by the differential receiver. The differential signals carrying data are amplified while the common-mode noise signals are suppressed. As a result, the balanced data transmission lines can be used for longer distances with higher transmission rates. Both unbalanced and balanced interfaces shown in Figures 20.2 and 20.3 represent two simplex interfaces, which can form one full-duplex point-to-point (see below) communication channel.

A good source of information on individual drivers and receivers is provided in the data sheets and application notes of semiconductor manufacturers [1,2].

20.1.11 Point-to-Point versus Multi-Point

If communication takes place between two devices, we call such a communication link a point-to-point link. In mechatronic systems, it is often required for the master system to communicate with a number of subsystems. Cost permitting, a number of point-to-point data transmission lines can be implemented. In a point-to-point arrangement, the master system has a point-to-point connection to each individual subsystem, i.e., there is a separate port and communication line for each subsystem. This type of arrangement is shown in Figure 20.4. The connection can also be arranged as a multi-point connection in which

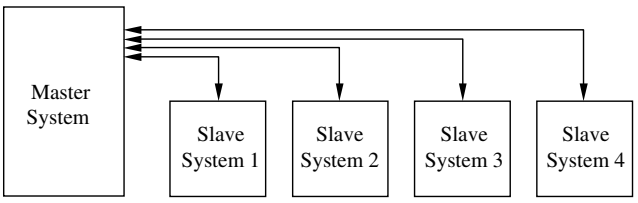


FIGURE 20.4 A point-to-point communication system with four subsystems.

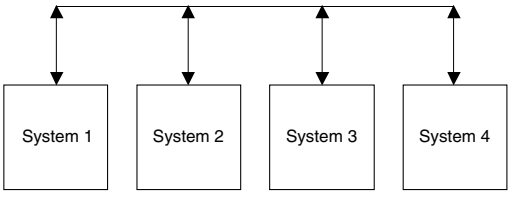


FIGURE 20.5 A multi-point communication system.

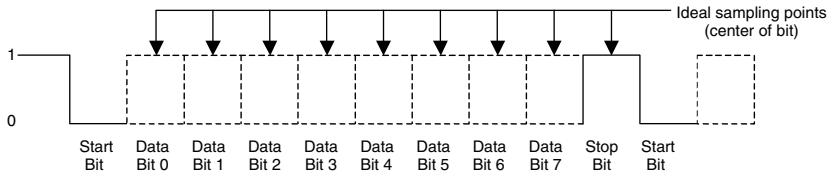


FIGURE 20.6 Asynchronous serial data format.

all devices are connected to a single transmission line, as shown in Figure 20.5. This arrangement is a data communication network arrangement where data can be transmitted from any device to any other device on the network. All devices on the network must be equipped with a receiver and a transmitter. Transmitters must have a tri-state (high output impedance) capability so that they do not provide additional load to the line. When transmitters are not transmitting, they are virtually disconnected from the transmission line. Complex communication protocol is required to manage individual transmitters on the network. The major advantage of a multi-point arrangement is usually the lower cost of the network compared to the individual communication links. The disadvantage is a more complex communication protocol (which must deal with the identity of the transmitting and receiving devices) and a more complex interface.

20.1.12 Serial Asynchronous Communications

In asynchronous serial communications, the data are transmitted at irregular intervals as a bit stream. Individual characters coded as binary numbers are converted to serial data streams, which are framed with start and stop bits. Optionally, a parity bit is added to the stream. In general, a computer represents information in parallel form such as bytes and words while the majority of communications with external devices takes place serially. The task of the parallel-to-serial and serial-to-parallel conversion is performed by a special integrated circuit called a universal asynchronous receiver transmitter (UART) as described later (see Figure 20.7).

Figure 20.6 shows an example of a typical data stream for asynchronous transmission. During idle time the line is in logical state 1 (for historical reasons also called "MARK"). The start of the data stream

is always indicated by the start bit, which has logical value 0 (also called "SPACE"). The start bit is followed by 5–8 data bits representing a character. The data bits are followed by an optional parity bit. The stream is terminated by one or two stop bits with logical value 1, which can be followed by idle line or the start bit of the next character. The idle line corresponds to logical state 1. A parity bit is an extra bit inserted after the data bits and before the stop bit(s). It is set according to the parity information of the data in the stream. For example, if an even parity is used, the parity bit is set such that total number of ones in the data stream including the parity bit is even. The parity bit is used by the receiver for error checking. The task of the receiver is to detect the start of the data stream and to correctly sample individual bits in the stream. After the detection of the start bit the receiver should sample individual bits, ideally at the mid point of each bit, as shown in Figure 20.6. In the case of an ideal sampling, as shown in Figure 20.6, the receiver is said to have distortion tolerance of 50%. In practice, the receiver of a UART is sampling incoming signals using the Baud Rate Generator frequency, which is 16 times higher than the corresponding baud rate used for transmission. The uncertainty in the detection of the start bit will reduce the distortion tolerance by 6.25% (1/16) to 43.75% [3].

If, for example, the receiver clock is 1% slower than the clock of the corresponding transmitter, the sampling time of the first data bit will be delayed by 1.5% of the bit time and the sampling time of the stop bit will be delayed by 9.5%. In this case, the distortion tolerance would be further reduced to 34.25%. If the receiver clock is slower by 5%, then the receiver may detect the start bit of the next character instead of the stop bit of the current character. This results in a framing error. The above example shows the significance of the accuracy of the clock speed and the reason why the data stream must be kept short in asynchronous transmission.

Other factors affecting the error-free communications include length and type of transmission line, speed of communications, parameters of line drivers, termination of transmission line, and the level of noise in the communication system.

20.1.13 The Universal Asynchronous Receiver Transmitter

The basic function of the UART is to facilitate parallel-to-serial and serial-to-parallel data conversion. The UART usually contains one transmitter and one receiver. The receiver and transmitter can operate simultaneously and independently. The UART can operate in full-duplex or half-duplex mode.

Parallel data from the host computer are converted to an asynchronous serial bit stream. The UART automatically adds a start bit, an optional parity bit, and the programmed number of stop bits, and sends the stream out through the transmitter serial data output (TxD) output pin. The parallel data are converted to a serial stream with the least significant bit shifted out first. Figure 20.7 shows a typical arrangement for UART. As can be seen, the UART uses TTL (transistor transistor logic) compatible interface. The TIA/EIA (see later) transmission line drivers and receivers are specific to a particular interface; thus, changing system interface means changing the transmission medium and the relevant drivers and receivers. The use of UART is independent of the transmission medium.

Serial data received on the receiver serial data input (RxD) pin is converted to parallel data. In the process the UART checks the start bit, parity bit (if any), and stop bit and reports any error conditions. Note that the UART is capable of generating all signals required for successful bit-serial asynchronous communications.

The UART can also report a number of error conditions, including receiver overrun, parity error, framing error, and break error. Receiver overrun error occurs when the bytes are received faster than the computer

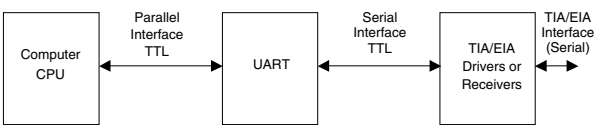


FIGURE 20.7 Typical arrangement for the UART.

processes them. The parity error is indicated if the parity of the bit stream changed during the communication process. Framing error is reported if the sampled stop bit is not at logic 1 level. The break error is reported if the communication line is idle for the time equivalent to the duration of at least one character.

Older types of UART devices such as 8250, 16450 had only one byte FIFO (first in first out) buffer and thus it was easy to overrun the receiver buffer. More recent devices are equipped with larger buffers providing more efficient communications. For example, device 16550D from National Semiconductor has a 16-byte receiver buffer and a 16-byte transmitter buffer and can operate at speeds up to 1.5 Mbps. Modern UARTs can also automatically handle tasks pertaining to multi-drop systems on a network.

20.2 TIA/EIA Serial Interface Standards

20.2.1 RS-232 Serial Interface

The RS-232 (Recommended Standard) was originally developed in 1962 by the Electronic Industries Association (EIA) as an interface between a computer and communication equipment. It is now jointly maintained by the Telecommunication Industries Association (TIA) and the EIA. The current version is designated as TIA/EIA 232-F (sixth revision) [4]. The Consulting Committee for International Telegraphs and Telephones (CCITT) issues recommendations that cover interfaces equivalent or similar to those issued by TIA/EIA.

Rapid development of computers created a demand for computer-to-computer communications over long distances. The switched public telephone network provided a readily available infrastructure for the communication task. Because computers generate digital data while the telephone network was designed for the transmission of voice signal, the digital signals from the computer had to be converted to a modulated signal which can be transmitted over the analog network. Modems (modulator/demodulator) are used to convert the digital signal into a modulated analog signal that is transmitted over the telephone line and converted back to digital signal by the modem at the other end of the telephone line. The RS-232 was designed as an interface between a computer and a modem. The formal name of the RS-232 standard is "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange," in which the Data Terminal Equipment (DTE) represents the computer and the Data Communication Equipment (DCE) represents the modem. Figure 20.8 shows an example of the RS-232 interface in the system providing computer-to-computer communication over the switched telephone network. The computers at each end represent DTE and the modems represent DCE.

The RS-232 interface standard specifies mechanical, electrical, and functional characteristics of the DTE/DCE interface. The CCITT V.24 interface describes equivalent functional characteristics and relies on other standards for mechanical and electrical characteristics of the interface. The RS-232 standard is widely used in applications where it provides a direct point-to-point connection between two computers or computers and field elements of mechatronic systems in which case we are dealing with DTE to DTE interface. As this is a situation where a modem is not required, the cable used to connect a DTE to another DTE is called a "null modem" cable, which has internal built-in connections to fake the presence of a modem.

The mechanical characteristic is concerned with the actual physical connection of the DTE and DCE and involves specification of pin assignments and genders of the connectors. The RS-232 standard does not specify a connector type, but it is customary to use either 25-pin D-type (DB-25) connector, which

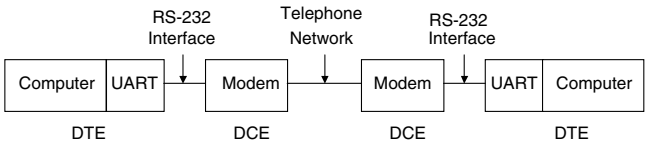


FIGURE 20.8 Data communication over a telephone network.

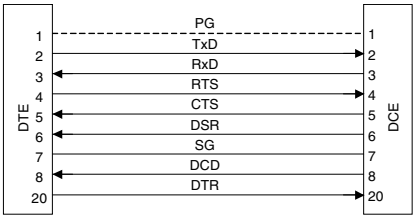


FIGURE 20.9 Pin assignment between a DTE and DCE.

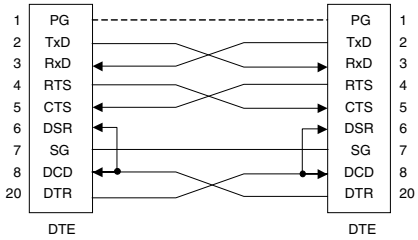


FIGURE 20.10 Example of a null modem cable pin assignment.

can accommodate all 25 pins, listed in the standard. In practice, a smaller number of pins are used; thus, as an alternative, a 9-pin D-type connector (DB-9) is often used. Please note that the pin assignment for DB-9 connector is not specified by RS-232 and is different from DB-25 pin assignment.

Figure 20.9 shows DB-25 connector pin assignments and the interconnection of selected circuits between a DTE and a DCE. Figure 20.10, on the other hand, shows an example of the DB-25 connector pin assignments and interconnection of selected circuits between two computers, i.e., two DTEs.

20.2.2 Functional Description of Selected Interchange Circuits

A full description of all signals as specified by the RS-232 standard is beyond the scope of this chapter. The reader is referred to the relevant standard [4]. We will describe the most common signals used in DTE/DCE and DTE/DTE interface. Please note that with the exception of the Protective Ground circuit and the Signal Ground circuit the circuits carry signals unidirectionally, as shown by arrows in Figure 20.9. Functional characteristics specify the functions that are performed by individual interchange circuits.

Protective Ground (PG). This line ensures that the chassis of the DTE and DCE are on the same potential.

Transmitted Data (TxD). Transmission line-signal originating from the DTE propagates to DCE.

Received Data (RxD). Receiver line-signal originating from the DCE propagates to DTE.

Request to Send (RTS). This signal is used to condition DCE for data transmission. On a half-duplex channel the signal controls the direction of data transmission of the DCE (transmit or receive). On a one-way-only channel (simplex) and on the full-duplex channels this signal controls the transmit state of the DCE (transmit or nontransmit state). A signal originating from the DTE propagates to DCE.

Clear to Send (CTS). This signal indicates that the DCE is ready to receive and is the response to the asserted RTS signal. The signal is originating from the DCE and propagates to DTE.

Data Set Ready (DSR). This signal indicates that the DCE is ready to operate. The signal is originating from the DCE and propagates to DTE.

Signal Ground (SG). This line is a common ground return line for all other signals.

Data Carrier Detect (DCD). This signal indicates that the DCE is receiving a valid modulated signal from the DCE at the other end. The signal is originating from the DCE and propagates to the DTE.

Data Terminal Ready (DTR). This signal indicates that the DTE is powered up and ready to operate. This signal is originating from the DTE and propagates to DCE.

The RS-232 specifies unbalanced, unidirectional, point-to-point interface. The interconnection is done over a set of wires referred to as interchange circuits. The electrical characteristics specify voltage levels of signals, rate of change of signals, and line impedance of interchange circuits. The standard specifies Nonreturn to Zero (NRZ) coding of digital signals.

The standard requires that the drivers be designed such that for the terminator load resistance between 3 and 7 kΩ the drivers should be capable of delivering high-level voltages between +5 and +15 V and low voltages between −5 and −15 V. The electrical signals are designed to provide a 2 V margin in signaling levels. The receiver signals are defined as +3 to +15 V for high voltage and as −3 to −15 V for low voltage.

It should be noted that for the data interchange circuit the high level voltage is defined as logic 0 (SPACE), while the low level voltage is defined as logic 1 (MARK). For control signals, on the other hand, the high level voltage defines the ON state while the low level voltage defines the OFF state. The maximum rate of change of signal allowed on both data and signal lines is 30 V/μs.

The original standard has also specified the maximum length of cable as 15 m. This specification was replaced by the specification of the maximum allowed capacitive load of 2500 pF in the EIA/TIA-232-D. The maximum cable length is determined by the capacitance of the cable per unit length; thus, this parameter now defines indirectly the length of the interface cable. The RS-232 interface is rated at signaling rates in the range from 0 to 20 kbps. It should be noted that in practice a good design would allow greater distances and greater data rates than the ones specified by the standard.

20.2.3 RS-422 and RS-485 Interfaces

The TIA/EIA-422-B standard "Electrical Characteristics of Balanced Voltage Digital Interface Circuits" [5] defines electrical characteristics of RS-422 interface. The RS-422 specifies a unidirectional, single driver, terminated balanced interface. The standard allows multiple receivers (up to 10) on one line. Figure 20.3 illustrates a typical point-to-point application of RS-422. As a result of improved noise immunity the RS-422 interface supports data rates up to 10 Mbps and cable length up to 1200 m, although not simultaneously. The maximum data rate of 10 Mbps is supported on a cable length up to 12 m, while a cable length of 1200 m supports data rates up to 100 kbps. Observe that the product of cable length and data rate is a limiting parameter of the interface. The transmission medium is a twisted-pair transmission line.

The TIA/EIA-485-A standard "Standard for Electrical Characteristics of Generators and Receivers for Use in Digital Multipoint Systems" [6] defines electrical characteristics of RS-485 interface. The RS-485 is a unique standard, which allows multiple nodes to communicate bidirectionally over a single twisted-pair transmission line. The RS-485 standard defines a low cost, multipoint balanced interface with electrical characteristic, supported cable types, cable length and data rates equivalent to those specified by the RS-422 standard. RS-485 parts are backward compatible and interchangeable with their equivalent RS-422 parts; however, RS-422 parts should not be used in RS-485 systems. RS-422 is usually used in point-to-point full-duplex communication systems, while RS-485 is used in multipoint half-duplex communication systems. The distinguishing feature of RS-485 drivers is their TRI-STATE capability, which allows the use of multiple drivers. The RS-485 has improved driver capability and input voltage range and supports up to 32 devices (drivers and/or receivers) on a single transmission line. Figure 20.11 shows a typical RS-485 multipoint application.

It should be noted that both RS-422 and RS-485 are electrical standards only. They do not specify mechanical or functional requirements.

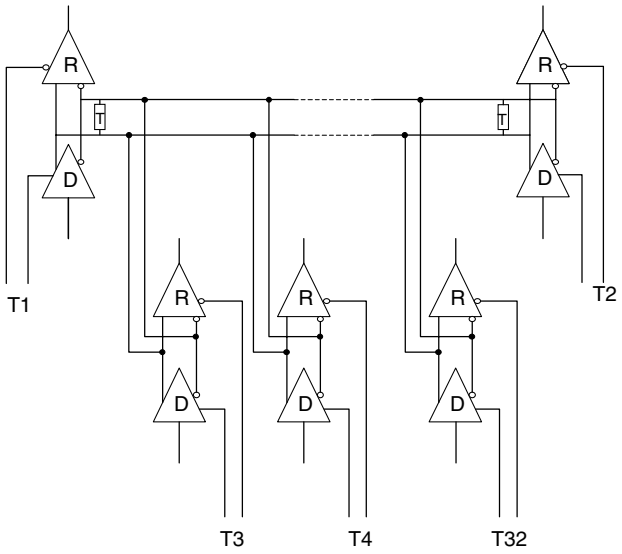


FIGURE 20.11 Example of an RS-485 multipoint application.

As mentioned earlier, data sheets and application notes of component suppliers provide an excellent source of information [1,2]. An excellent overview of practical data communications and interfacing for instrumentation and control is provided in reference 7. Detailed discussion of design aspects of serial communications and interfacing based on RS-232 and RS-485 is given in reference 8 It is recommended that for the full specification the designer should consult the relevant standards [4–6]. Additional information including design recommendations can be found in [9–11]. A good introduction and background theory to data communication and computer networks can be found in reference 12.

20.3 IEEE 488—The General Purpose Interface Bus

20.3.1 Introduction

The interface described by IEEE 488 standard, which will be referred to as general purpose interface bus (GPIB) in this chapter, is used to connect instruments to test and measurement systems. Examples of such instruments are digital voltmeters, storage oscilloscopes, printers, and plotters. In general, these instruments are called GPIB devices. These devices operate under the coordination of a controller. Most modern systems consist of a cluster of such devices connected to one or more computers. In such a system, one of the computers will become the controller.

Historically, the interface was developed by Hewlett–Packard in 1965. At that time, the interface was called HPIB, and a general standard did not exist. In 1975, it was formulated as IEEE 488 and was called IEEE Standard Digital Interface for Programmable Instrumentation. The standard specified the electrical, mechanical, and hardware aspects, i.e., the signals, their functioning, and purpose. Instrument manufacturers used the interface freely without adhering to a standard protocol in communicating with instruments. Instruments meant for the same purpose, yet manufactured by different manufacturers required widely varied commands. Some instruments made measurements in response to a command, while some other instruments of similar type made measurements without a command at all. Further, there were no agreed data formats between instruments sending data and instruments receiving data. This situation led to the development of an extension to the IEEE 488 standard. The new standard was published in 1987 and was

called IEEE 488.2 Standard Codes, Formats, Protocols, and Common Commands for Use with IEEE 488.1 (1987) [13], where IEEE 488.1 is the new name for the original IEEE 488 standard.

The IEEE 488.2 compliant devices must present data through data formats and codes specified in the standard. The standard also specifies a minimum set of mandatory control sequences or commands and suggests a few other optional commands. It also provides a standard status-reporting model that must be implemented by the instrument manufacturers so that determining the status of instruments will be easier for the instrument programmers.

Although not yet a standard, The Standard Commands for Programmable Instrumentation (SCPI) put together in 1990 agrees upon a standard set of commands for various instrument categories. Accordingly, all digital voltmeters manufactured by different manufacturers will respond to the same GPIB command.

20.3.2 GPIB Hardware

This section describes the electrical and mechanical specifications of the GPIB interface as well as the signal description and their purpose.

All GPIB devices are connected using a special cable with each end having the male as well as the female ends of the connector. This permits piggyback connections of cables. The devices can be connected either in a chained manner (i.e., device B connected to device A, device C connected to device B, etc.) or in a star configuration (i.e., device A, B, C, etc. connected to a common node). The connection configurations are shown in Figure 20.12.

A maximum of 15 devices can be connected to the bus. The maximum separation between two devices is 4 m with an average separation of not more than 2 m. At least two-thirds of the devices connected must be powered on.

The GPIB cable consists of 24 wires. Eight of these lines are data lines, while three lines are used for handshaking. Another five lines are used for interface management and the remaining eight lines are ground lines. Among the ground lines are a cable shield line, a signal ground line, three ground return lines for the handshaking signals, and three other ground return lines for three of the interface management lines. All signals used are standard TTL signal levels with negative logic. The handshake lines and interface management lines are given in Table 20.1.

The operation of the individual lines is not important to the average user or programmer as their usage is taken care of by the controllers and the instruments that comply with IEEE 488.2 standard.

TABLE 20.1 Handshaking and Interface Management Lines

Handshaking Lines		Interface Management Lines	
NRFD	Not ready for data	ATN	Attention
NDAC	Not data accepted	IFC	Interface clear
DAV	Data valid	REN	Remote enable
		SRQ	Service request
		EOI	End of identify

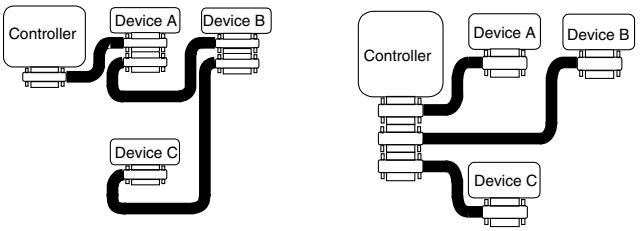


FIGURE 20.12 Linear and star configurations of connecting GPIB devices to a controller.

20.3.3 Controllers, Talkers, and Listeners

The controller carries out the general management of the bus. While there can be many controllers connected to the GPIB network, there can be only one controller-in-charge (CIC) which manages the bus at that given time. All information sent out by the controller on the data lines are called "commands" and all information sent out by other devices are termed "data." The GPIB devices that send data any time are called "talkers" and the devices that receive data are called "listeners." While there can be more than one listener operating at any given time, there can be only one talker operating at any given time. A system can have permanent talkers and permanent listeners; however, if the capability exists, a GPIB device can be a listener at one time and a talker at another time. A brief explanation of the signal lines is given below, as it would enhance our understanding of the operation of GPIB interface.

20.3.4 Interface Management Lines

20.3.4.1 Attention (ATN)

The ATN line is controlled by the CIC. When asserted, the signals on the data lines constitute a command signal and all devices must listen. When unasserted, the signals on the data lines represent data and are generally sent by a talker to one or more listeners.

20.3.4.2 Interface Clear (IFC)

The IFC line is asserted by the CIC to reset the GPIB bus. Upon receipt of this signal, all GPIB devices on the bus will initialize themselves.

20.3.4.3 Remote Enable (REN)

GPIB devices can be controlled either locally or remotely. The CIC asserts the REN line to bring all GPIB devices under remote programming mode. Thus, for example, the change of scale of a DVM can be carried out by a GPIB command instead of a front panel control.

20.3.4.4 Service Request (SRQ)

Any device other than a controller can asynchronously assert the SRQ line requesting service from the controller. The controller monitors the SRQ line and polls all devices to determine the device or devices requiring service.

20.3.4.5 End of Identity (EOI)

The EOI signal is used by a talker to indicate the end of the data message of the talker. It indicates to the listener(s) the end of the receiving data record.

20.3.5 Handshake Lines

In general, a data transfer with complete handshake gets through three stages: request or preparedness, data transfer, and acknowledgment. On some systems, where the stability of data on the data bus is questionable, a data valid signal may also be provided. On the GPIB bus, when a talker has to send data to a listener, the controller must address a device and instruct it to be the talker and then address one or more other devices and instruct them to be listeners. See Section 20.3.7 for addressing of GPIB devices.

20.3.5.1 Not Ready for Data (NRFD)

The NRFD line is controlled by the controller when sending commands or by the talker when sending data. A device that has been instructed to be a listener will unassert NRFD to indicate to the talker that it is ready to receive data. Of all the listeners, the slowest device will be the last to unassert NRFD and thus control the speed of data transfer.

20.3.5.2 Data Valid (DAV)

When all listeners have indicated their readiness to receive data by unasserting NRFD, the talker (or the controller when sending commands) will assert a DAV signal to indicte to all listeners that the data on the data lines DIO1-DIO8 are stable and may be read by the listeners. In response to a DAV signal, the listeners

may assert NRFD to halt any further data transmission by talkers until the data already transmitted has been received.

20.3.5.3 Not Data Accepted (NDAC)

The NDAC line driven by all listeners is the acknowledgment signal. When data has been received by all listeners, the NDAC line will be unasserted. The talker can then remove the data and unassert the DAV signal.

20.3.6 Data Lines DIO1–DIO8 (8 lines)

The data lines are controlled by the controller when issuing commands or by the talker. As soon as the controller instructs a particular device to be a talker, that device will place data on the lines DIO1–DIO8 and will wait for at least T1 seconds.

20.3.7 Addressing of GPIB Devices

All GPIB devices connected to a GPIB bus must have a unique GPIB address. A device can have a primary address as well as a secondary address. Most devices use a primary address only. The addresses are in the range 0–30 decimal. In general, the addresses on the instruments as well as the controller are set using switches. The controller instructs a device with a particular address to be a talker or a listener by sending a bit pattern on the data bus. The bit pattern is formed according to Table 20.2. The data bits are numbered from D7 to D0. The value of each bit is listed straight underneath. The letter A signifies 0 or 1. The five bits D4–D0 will form a bit pattern representing the address of the device. The letter X signifies a "don't care" bit, which is not used. TA will be set to 1 if the controller is instructing the device to be a talker. LA will be set to 1 if the controller is instructing the device to be a listener. For example, if a particular device has the address 15 (decimal) and if the controller is instructing that device to be the talker, then the controller must send the following bit pattern over the data lines DIO8-DIO1.

Device with address 15 be the talker 0 1 0 0 1 1 1 1 = 4F (hex)

Similarly,

Device with address 0 be the listener 0 0 1 0 0 0 0 0 = 20 (hex)

"Untalk" the current talker 0 1 0 1 1 1 1 1 = 5F (hex)

"Unlisten" all listeners 0 0 1 1 1 1 1 1 = 3F (hex)

Note that Untalk and Unlisten commands look very similar to Talk and Listen commands; however, the address 31 (decimal) does not exist. Therefore, the address 31 is used to affect Untalk and Unlisten.

For the controllers, IEEE 488.2 Standard provides the Required and Optional Control Sequences. All controllers that comply with IEEE 488.2 must support all mandatory commands. The standard also provides the "Controller Protocols." Protocols are formed by combining a set of control sequences. For example, FINDLSTN command will issue a set of control sequences to determine the existing listeners.

For the instruments, IEEE 488.2 specifies a set of mandatory commands and queries. For example, when the command "*RST" is received by IEEE 488.2 compliant instruments, they all must carry out an instrument reset. Similarly, upon receipt of "*STB?" command the instrument will send the status byte to the controller.

TABLE 20.2 Talker/Listener Addressing Commands

D7	D6	D5	D4	D3	D2	D1	D0
X	TA	LA	A	A	A	A	A

All mandatory common commands and queries, all required and optional control sequences as well as the controller protocols can be found in references 14 and 15.

References

1. Goldie, J., Summary of well known interface standards, Application note AN-216, National Semiconductor, 1998, www.national.com.
2. Goldie, J., Comparing EIA-485 and EIA-422-A line drivers and receivers in multipoint applications, Application note AN-759, National Semiconductor, 1998.
3. McNamara, J.E., *Technical Aspects of Data Communication,* 3rd ed., Digital Press, 1988.
4. TIA/EIA-232-F, Interface between data terminal equipment and data communication equipment employing serial binary data interchange, TIA, EIA, 1997.
5. TIA/EIA-422-B, Electrical characteristics of balanced voltage digital interface circuits, TIA, EIA, 1995.
6. TIA/EIA-485-A, Standard for electrical characteristics of generators and receivers for use in digital multipoint systems, TEI, EIA, 1998.
7. Mackay, S.G., et al., *Data Communications for Instrumentation and Control,* IDC Techbooks, 2000.
8. Axelson, J., *Serial Port Complete,* Lakeview Research, Madison, 1998.
9. Goldie, J., Ten ways to bulletproof RS-485 interfaces, Application note AN-1057, National Semiconductor, 1996.
10. RS-422 and RS-485 Application Note, B&B Electronics Manufacturing Co., 1997, www.bb-elec.com.
11. DALLAS SEMICONDUCTOR, Application Note 83, Fundamentals of RS-232 Serial Communications, 1998.
12. Stallings, W., *Data and Computer Communications,* 6th ed., Prentice-Hall, Upper Saddle River, NJ, 2000.
13. ANSI/IEEE 488.1-1987 IEEE standard digital interface for programmable instrumentation. institution of Electrical and Electronic Engineers, New York, 1987.
14. ANSI/IEEE 488.2-1987 IEEE standard codes, formats, protocols and common commands. institution of Electrical and Electronic Engineers, New York, 1987.
15. ANSI/IEEE 488.2-1992 IEEE standard codes, formats, protocols and common commands, and standard commands for programmable instruments. Institution of Electrical and Electronic Engineers, New York, 1992.

21

Communications and Computer Networks

Mohammad Ilyas
Florida Atlantic University

The field of communications and computer networks deals with efficient and reliable transfer of information from one point to another. The need to exchange information is not new but the techniques employed to achieve information exchange have been steadily improving. During the past few decades, these techniques have experienced an unprecedented and innovative growth. Several factors have been and continue to be responsible for this growth. The Internet is the most visible product of this growth and it has impacted the life of each and every one of us. This chapter describes salient features and operational details of communications and computer networks.

The contents of this chapter are organized in several sections. Section 21.1 describes a brief history of the field of communications. Section 21.2 deals with the introduction of communication and computer networks. Section 21.3 describes operational details of computer networks. Section 21.4 discusses resource allocation mechanisms. Section 21.5 briefly describes the challenges and issues in communication and computer networks that are still to be overcome. Finally, Section 21.6 summarizes the chapter.

21.1 A Brief History

Exchange of information (communications) between two or more entities has been a necessity since the existence of human life. It started with some form and shape of human voice that one entity can create and other(s) can listen to and interpret. Over a period of several centuries, these voices evolved into languages. As the population of the world grew, more and more languages were born. For a long time, languages were used for face-to-face communications. If there were ever a need to convey some information (a message) over a distance, someone would be briefed and sent to deliver the message to a distant site. Gradually, additional methods were developed to represent and exchange the information. These methods included symbols, shapes, and eventually alphabets. This development facilitated information recording and use of nonvocal means for exchanging information. Hence, preservation, dissemination, sharing, and communication of knowledge became easier.

Until about 150 years ago, all communication was via wireless means and included smoke signals, beating of drums, and use of reflective surfaces for reflecting light signals (optical wireless). Efficiency of

these techniques was heavily influenced by environmental conditions. For instance, smoke signals were not very effective in windy conditions. In any case, as we will note later, some of the techniques that were in use centuries ago for conveying information over a distance were similar to the techniques that we currently use. The only difference is that the implementation of those techniques is exceedingly more sophisticated now than it was centuries ago.

As the technological progress continued and electronic devices started appearing on the surface, the field of communication also started making use of the innovative technologies. Alphabets were translated into their electronic representations so that information could be electronically transmitted. Morse code was developed for telegraphic exchange of information. Further developments led to the use of the telephone. It is important to note that in earlier days of technological masterpieces, users would go to a common site where one could send a telegraphic message over a distance or could have a telephonic conversation with a person at a remote location. This was a classic example of resource sharing. Of course, human help was needed to establish a connection with remote sites.

As the benefits of the advances in communication technologies were being harvested, electronic computers were also emerging and making the news. Earlier computers were not only expensive and less reliable, they were also huge in size. For instance, the computers that used vacuum tubes were of the size of a large room and used roughly about 10,000 vacuum tubes. These computers would stop working if a vacuum tube burned out, and the tube would need to be replaced by using a ladder. On average, those computers would function for a few minutes before another vacuum tube's replacement was necessary. A few minutes of computer time was not enough to execute a large computer program. With the advent of transistors, computers not only became smaller in size and less expensive, but also more reliable. These aspects of computers resulted in their widespread applications. With the development of personal computers, there is hardly any side of our lives that has not been impacted by the use of computers. The field of communications is no exception and the use of computers has escalated our communication capabilities to new heights.

21.2 Introduction

Communication of information from one point to another in an efficient and reliable manner has always been a necessity. A typical communication system consists of the following components as shown in Figure 21.1:

- Source that generates or has the information to be transported
- Transmitter that prepares the information for transportation
- Transmission medium that carries the information from one end to the other
- Receiver that receives the information and prepares it for delivering to the receiver
- Destination that takes the information from receiver and utilizes it as necessary

The information can be generated in analog or digital form. Analog information is represented as a continuous signal that varies smoothly in time. As one speaks in a microphone, an analog voice signal is generated. Digital information is represented by a signal that stays at some fixed level for some duration of time followed by a change to another fixed level. A computer works with digital information that has two levels (binary digital signals). Figure 21.2 shows an example of analog and digital signals. Transmission of

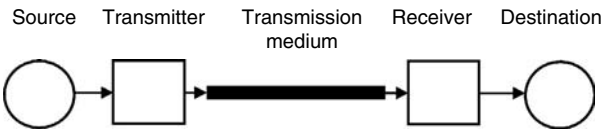


FIGURE 21.1 A typical communication system.

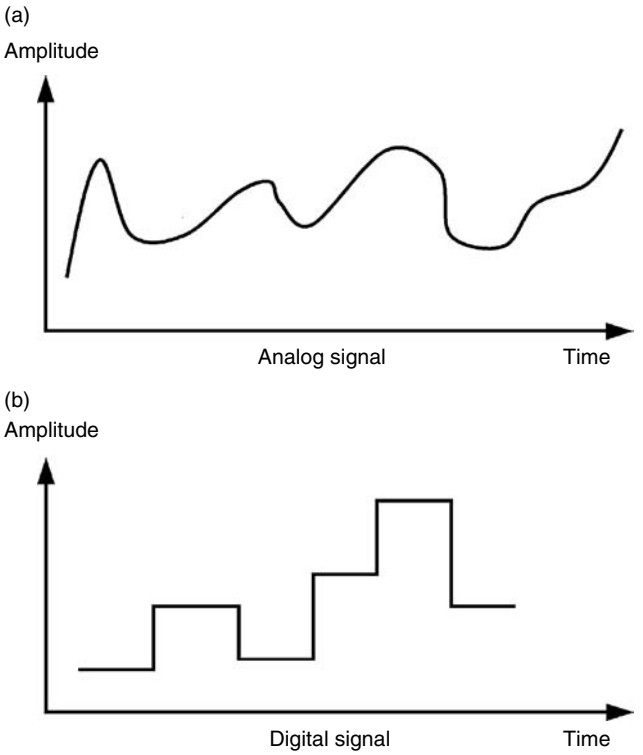


FIGURE 21.2 Typical analog (a) and digital (b) signals.

information can also be in analog or digital form. Therefore, we have the following four possibilities in a communication system [21]:

- Analog information transmitted as an analog signal
- Analog information transmitted as a digital signal
- Digital information transmitted as an analog signal
- Digital information transmitted as a digital signal

There may not be a choice regarding the form (analog or digital) of information being generated by a device. For instance, a voice signal as one speaks, a video signal as generated by a camera, a speed signal generated by a moving vehicle, and an altitude signal generated by the equipment in a plane will always be analog in nature; however, there is a choice regarding the form (analog or digital) of information being transmitted over a transmission medium. Transmitted information could be analog or digital in nature and information can be easily converted from one form to another.

Each of these possibilities has its pros and cons. When a signal carrying information is transmitted, it looses its energy and strength and gathers some interference (noise) as it propagates away from the transmitter. If the energy of the signal is not boosted at some intermediate point, it may attenuate beyond recognition before it reaches its intended destination. That will certainly be a wasted effort. In order to boost energy and strength of a signal, it must be amplified (in case of analog signals) and rebuilt (in case of digital signals). When an analog signal is amplified, the noise also becomes amplified and that certainly lowers expectations about receiving the signal at its destination in its original (or close to it) form. On the other hand, digital signals can be processed and reconstructed at any intermediate point and, therefore, the noise can essentially be filtered out. Moreover, transmission of information in digital form has many

other advantages including processing of information for error detection and correction, applying encryption and decryption techniques to sensitive information, and many more. Thus, digital information transmission technology has become the dominant technology in the field of communications [9,18].

As indicated earlier, communication technology has experienced phenomenal growth over the past several decades. The following two factors have always played a critical role in shaping the future of communications [20]:

- Severity of user needs to exchange information
- State of the technology related to communications

Historically, inventions have always been triggered by the severity of needs. It has been very true for the field of communications as well. In addition, there is always an urge and curiosity to make things happen faster. When electricity was discovered and people (scattered around the globe) wanted to exchange information over longer distances and in less time, the telegraph was invented. Morse code was developed with shorter sequences (of dots and dashes) for more frequent alphabets. That resulted in transmission of messages in a shorter duration of time. The presence of electricity and the capability of wires to carry information over longer distances led to the development of devices that converted human voice into electrical signal, and thus led to the development of telephone systems. Behind this invention was also a need/desire to establish full-duplex (two-way simultaneous) communication in human voice. As use of the telephone became widespread, there was a need for a telephone user to be connected to any other user, and that led to the development of switching offices. In the early days, the switching offices were operated manually. As the state of the technology improved, the manual switching offices were replaced by automatic switching offices. Each telephone user was assigned a telephone number for identification purposes and a user able to dial the number for the purpose of establishing a connection with the called party. As the computer technology improved and the computers became easier to afford and smaller in size, they found countless uses including their use in communications. The computers not only replaced the automatic (electromechanical) switching offices, they were also employed in many other aspects of communication systems. Examples include conversion of information from analog to digital and vice versa, processing of information for error detection and/or correction, compression of information, and encryption/decryption of information.

As computers became more powerful, there were many other applications that surfaced. The most visible application was the amount of information users started sharing among themselves. The volume of information being exchanged among users has been growing exponentially over the last three decades. As users needed to exchange such a mammoth amount of information, new techniques were invented to facilitate the process. There was not only a need for users to exchange information with others in an asynchronous fashion, there was also a need for computers to exchange information among themselves. The information being exchanged in this fashion has different characteristics than the information being exchanged through the telephone systems. This need led to the interconnection of computers with each other and that is what is called computer networks.

21.3 Computer Networks

A computer network is an interconnection of computers. The interconnection forms a facility that provides reliable and efficient means of communication among users and other devices. User communication in computer networks is assisted by computers, and the facility also provides communication among computers. Computer networks are also referred to as computer communication networks. Interconnection among computers may be via wired or wireless transmission medium [5,6,10,13,18].

There are two broad categories of computer networks:

- Wide area networks
- Local/metropolitan area networks

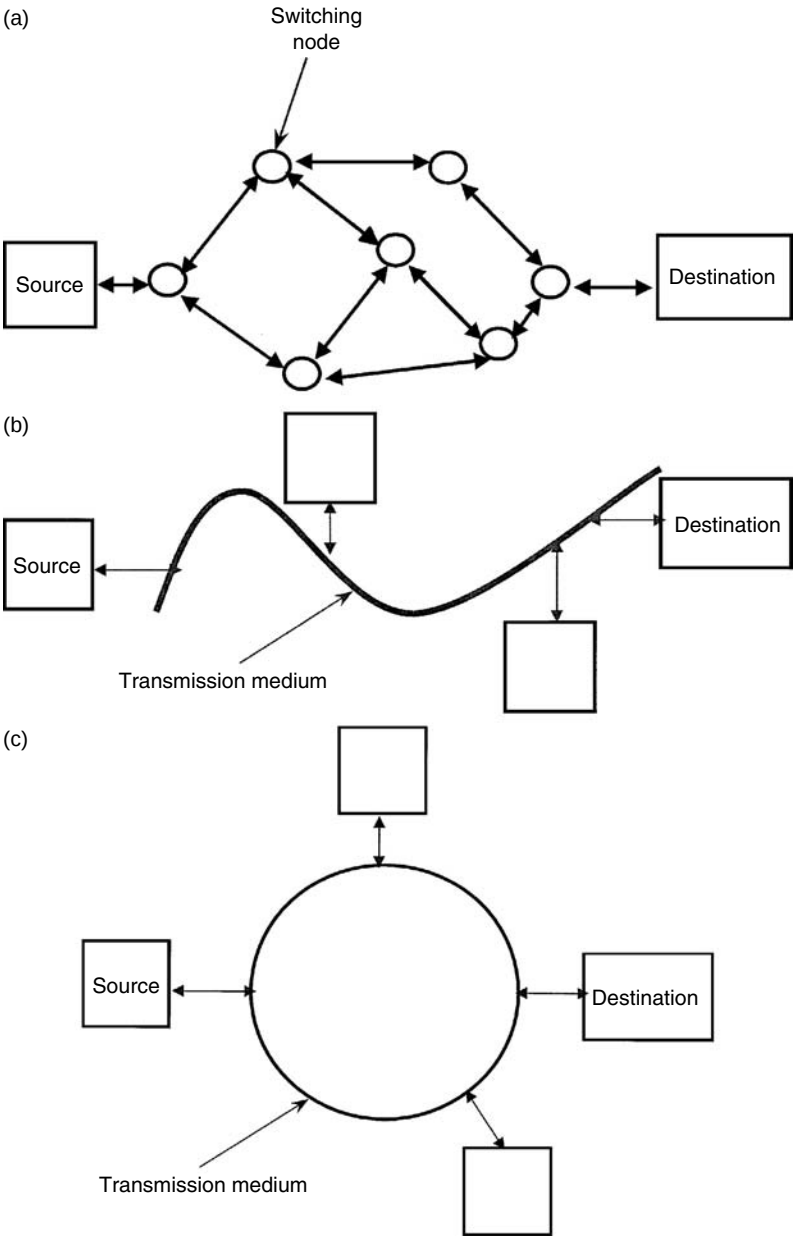


FIGURE 21.3 (a) A typical wide area computer communication network. (b) A typical local/metropolitan area communication bus network. (c) A typical local/metropolitan area communication ring network.

Wide area computer networks, as the name suggests, span a wider geographical area and essentially have a global scope. On the other hand, local/metropolitan area networks span a limited distance. Local area networks are generally confined to an industrial building or an academic institution. Metropolitan area networks also have limited geographical scope but it is relatively larger than that of the local area networks [19]. Typical wide and local/metropolitan area networks are shown in Figure 21.3.

Once a user is connected to a computer network, that user can communicate with any other user also connected to the network at some point. It is not required for a user to be connected directly to another

user in order to communicate. In fact, in wide area networks, two communicating users will rarely be directly connected with each other. This implies that the users will be sharing the transmission links for exchanging their information. This is one of the most important aspects of computer networks. Sharing of resources improves utilization of the resources and is, of course, cost-effective as well. In addition to sharing the transmission links, the users will also share the processing power of the computers at the switching nodes, buffering capacity to store the information at the switching nodes, and any other resources that are connected to the computer network. A user who is connected to a computer network at any switching node will have immediate access to all the resources (databases, research articles, surveys, and much more) that are connected to the network as well. Of course, access to specific information may be restricted and a user may require appropriate authorization to access the information.

The information from one user to another may need to pass through several switching nodes and transmission links before reaching its destination. This implies that a user may have many options available to select one out of many sequences of transmission links and switching nodes to exchange information. That adds to the reliability of the information exchange process. If one path is not available, not feasible, or not functional, some other path may be used. In addition, for better and effective sharing of resources among several users, it is not appropriate to let any user exchange a large quantity of information at a time; however, it is not uncommon that some users may have a large quantity of information to exchange. In that case, the information is broken into smaller units known as packets of information. Each packet is sent toward its destination as a separate entity and then all packets are assembled together at the destination side to re-create the original piece of information [2].

Due to the resource sharing environment, users may not be able to exchange their information at any time they wish because the resources (switching nodes, transmission links) may be busy serving other users. In that case, some users may have to wait for some time before they begin their communication. Designers of computer networks should design the network so that the total delay (including wait time) is as brief as possible and that the total amount of information successfully exchanged (throughput) is as large as possible.

Many aspects must be addressed for enabling networks to transport users' information from one point to another. The major aspects are:

- Addressing mechanism to identify users
- Addressing mechanism for information packets to identify their source and destination
- Establishing a connection between sender and receiver and maintaining it
- Choosing a path or a route (sequence of switching nodes and transmission links) to carry the information from a sender to a receiver
- Implementing a selected route or path
- Checking information packets for errors and recovering from errors
- Encryption and decryption of information
- Controlling the flow of information so that shared resources are not over-taxed
- Informing the sender that the information has been successfully delivered to the intended destination (acknowledgment)
- Billing for the use of resources
- Ensuring that different computers running different applications and operating systems can exchange information
- Preparing information appropriately for transmission over a given transmission medium

This is not an exhaustive list of items that need to be addressed in computer networks. In any case, all such issues are addressed by very systematic and detailed procedures. The procedures are called communication protocols. The protocols are implemented at the switching nodes by a combination of hardware and software. It is not advisable to implement all these features in one module of hardware or software because that will become very difficult to manage. It is a standard practice that these features be divided into different

smaller modules and then these modules can be interfaced together to collectively provide implementation of these features. International Standards Organization (ISO) has suggested dividing these features into seven distinct modules called layers. The proposed model is referred to as Open System Interconnection (OSI) reference model. The seven layers proposed in the OSI reference model are [2]:

- Application layer
- Presentation layer
- Session layer
- Transport layer
- Network layer
- Data link layer
- Physical layer

The physical layer deals with the transmission of information on the transmission medium. The data link layer handles the information on a single link. The network layer deals with the path or route of information from the switching node where the source is connected to the switching node where the receiver is connected. It also monitors end-to-end information flow. The remaining four layers reside with the user equipment. The transport layer deals with the information exchange from the source to the sender. The session layer handles the establishment of a session between the source and the receiver and maintains it. The presentation layer deals with the form in which information is presented to the lower layer. Encryption/decryption of information can also be performed at this layer. The application layer deals with the application that generates the information at the source side and what happens to it when it is delivered at the receiver side.

As the information begins from the application layer at the sender side, it is processed at every layer according to the specific protocols implemented at that layer. Each layer processes the information and appends a header and/or a trailer with the information before passing it on to the next layer. The headers and trailers appended by various layers contribute to the overhead and are necessary for transportation of the information. Finally, at the physical layer, the bits of information packets are converted to an appropriate signal and transmitted over the transmission medium. At the destination side, the physical layer receives the information packets from the transmission medium and prepares them for passing these to the next higher layer. As a packet is processed by the protocol layers at the destination side, its headers and trailers are stripped off before it is passed to the next layer. By the time information reaches the application layer, it should be in the same form as it was transmitted by the source.

Once a user is ready to send information to another user, he or she has two options. He or she can establish a communication with the destination prior to exchanging information or he can just give the information to the network node and let the network deliver the information to its destination. If communication is established prior to exchanging the information, the process is referred to as connection-oriented service and is implemented by using virtual circuit connections. On the other hand, if no communication is established prior to sending the information, the process is called connectionless service. This is implemented by using a datagram environment. In connection-oriented (virtual circuit) service, all packets between two users travel over the same path through a computer network and, hence, arrive at their destination in the same order as they were sent by the source. In connectionless service, however, each packet finds its own path through the network while traveling towards its destination. Each packet will therefore experience a different delay and the packets may arrive at their destination out of sequence. In that case, the destination will be required to put all the packets in proper sequence before assembling them [2,10,13].

As in all resource sharing systems, allocation of resources in computer networks requires careful attention. The main idea is that the resources should be shared among users of a computer network as fairly as possible. At the same, it is desired to maintain the network performance as close to its optimal level as possible. The fairness definition, however, varies from one individual to another and depends upon how one is associated with a computer network. Although fairness of resource sharing is being evaluated, two performance parameters—delay and throughput—for computer networks are considered. The delay

is the duration of time from the moment information is submitted by a user for transmission to the moment it is successfully delivered to its destination. The throughput is the amount of information successfully delivered to its intended destination per unit time. Due to the resource sharing environment in computer networks, these two performance parameters are contradictory. It is desired to have the delay as small as possible and the throughput as large as possible. For increasing throughput, a computer network must handle increased information traffic, but the increased level of information traffic also causes higher buffer occupancy at the switching nodes and, hence, more waiting time for information packets. This results in an increase in delay. On the other hand, if information traffic is reduced to reduce the delay, that will adversely affect the throughput. A reasonable compromise between throughput and delay is necessary for the satisfactory operation of a computer network [10,11].

21.3.1 Wide Area Computer Networks

A wide area network consists of switching nodes and transmission links as shown in Figure 21.3a. Layout of switching nodes and transmission links is based on the traffic patterns and expected volume of traffic flow from one site to another site. Switching nodes provide the users access to a computer network and implement communication protocols. When a user is ready to transmit his or her information, the switching node, to which the user is connected, will establish a connection if a connection-oriented service has been opted. Otherwise, the information will be transmitted in a connectionless environment. In either case, switching nodes play a key role in determining the path of the information flow according to some well-established routing criteria. The criteria include performance (delay and throughput) objectives among other factors based on user needs. For keeping the network traffic within a reasonable range, some traffic flow control mechanisms are necessary. In late 1960s and early 1970s, when data rates of transmission media used in computer networks were low (a few thousand bits per second), these mechanisms were fairly simple. A common method used for controlling traffic over a transmission link or a path was an understanding that the sender would continue sending information until the receiver sent a request to stop. The information flow would resume as soon as the receiver sent another request to resume transmission. Basically the receiver side had the final say in controlling the flow of information over a link or a path. As the data rates of transmission media started increasing, this method was not deemed efficient. To control the flow of information in relatively faster transmission media, a sliding window scheme was used. According to this scheme, the sender will continuously send information packets but no more than a certain limit. Once the limit is reached, the sender will stop sending the information packets and will wait for the acknowledgment that the packets have been transmitted. As soon as an acknowledgment is received, the sender may send another packet. This method ensures that there are no more than a certain specific number of packets in transit from sender to receiver at any given time. Again, the receiver has control over the amount of information that the sender can transmit. These techniques for controlling the information traffic are referred to as reactive- or feedback-based techniques because the decision to transmit or not to transmit is based on the current traffic conditions.

Reactive techniques are acceptable in low to moderate data rates of transmission media. As the data rates increase from kilobits per second to megabits and gigabits per second, the situation changes. Over the past several years, there has been a manifold increase in data rates. Optical fibers provide enormously high data rates. Size of the computer networks has also experienced tremendous increase. The amount of traffic flowing through these networks has been increasing exponentially. Given that, the traffic control techniques used in earlier networks are not quite effective anymore [11,12,22]. One more factor that has added to the complexity of the situation is that users are now exchanging different types of information through the same network. Consider the example of the Internet. The geographical scope of the Internet is essentially global. Extensive use of optical fiber as transmission media provides very high data rates for exchanging information. In addition, users are using the Internet for exchanging any type of information they come across, including voice, video, and data. All these factors have essentially necessitated the use of a modified approach for traffic management in computer networks. The main factor leading to this change is that the information packets are moving so fast through the computer networks that any feedback-based (or reactive)

control will be too slow to be of any use. Therefore, some preventive mechanisms have been developed to maintain the information traffic inside a computer network to a comfortable level. Such techniques are implemented at the sender side by ensuring that only as much information traffic is allowed to enter the network as can be comfortably handled by the networks [1,20,22]. Based on the users' needs and state of the technology, providing faster communications for different types of services (voice, video, data, and others) in the same computer network in an integrated and unified manner has become a necessity. These computer networks are referred to as broadband integrated services digital networks (BISDNs). BISDNs provide end-to-end digital connectivity and users can access any type of communication service from a single point of access. Asynchronous transfer mode (ATM) is expected to be used as a transfer mechanism in BISDNs. ATM is essentially a fast packet switching technique where information is transmitted in the form of small fixed-size packets called cells. Each cell is 53 bytes long and includes a header of 5 bytes. The information is primarily transported using a connection-oriented (virtual circuit) environment [3,4,8,12,17].

Another aspect of wide area networks is the processing speed of switching nodes. As the data rates of transmission media increase, it is essential to have faster processing capability at the switching nodes. Otherwise, switching nodes become bottlenecks and faster transmission media cannot be fully utilized. When transmission media consist of optical fibers, the incoming information at a switching node is converted from optical form to electronic form so that it may be processed and appropriately switched to an outgoing link. Before it is transmitted, the information is again converted from electronic form to optical form. This slows down the information transfer process and increases the delay. To remedy this situation, research is being conducted to develop large optical switches to be used as switching nodes. Optical switches will not require conversion of information from optical to electronic and vice versa at the switching nodes; however, these switches must also possess the capability of optical processing of information. When reasonably sized optical switches become available, use of optical fiber as transmission media together with optical switches will lead to all-optical computer and communication networks. Information packets will not need to be stored for processing at the switching nodes and that will certainly improve the delay performance. In addition, wavelength division multiplexing techniques are rendering use of optical transmission media to its fullest capacity [14].

21.3.2 Local and Metropolitan Area Networks

A local area network has a limited geographical scope (no more than a few kilometers) and is generally limited to a building or an organization. It uses a single transmission medium and all users are connected to the same medium at various points. The transmission medium may be open-ended (bus) as shown in Figure 21.3b or it may be in the form of a loop (ring) as shown in Figure 21.3c. Metropolitan area networks also have a single transmission medium that is shared by all the users connected to the network, but the medium spans a relatively larger geographical area, up to 150 km. They also use a transmission medium with relatively higher data rates. Local and metropolitan area networks also use a layered implementation of communication protocols as needed in wide area networks; however, these protocols are relatively simpler because of simple topology, no switching nodes, and limited distance between the senders and the receivers. All users share the same transmission medium to exchange their information. Obviously, if two or more users transmit their information at the same time, the information from different users will interfere with each other and will cause a collision. In such cases, the information of all users involved in a collision will be destroyed and will need to be retransmitted. Therefore, there must be some well-defined procedures so that all users may share the same transmission medium in a civilized manner and have successful exchange of information. These procedures are called medium access control (MAC) protocols.

There are two broad categories of MAC protocols:

- Controlled access protocols
- Contention-based access protocols

In controlled access MAC protocols, users take turns transmitting their information and only one user is allowed to transmit information at a time. When one user has finished his or her transmission, the next user begins transmission. The control could be centralized or distributed. No information collisions occur and, hence, no information is lost due to two or more users transmitting information at the same time. Examples of controlled access MAC protocols include token-passing bus and token-passing ring local area networks. In both of these examples, a token (a small control packet) circulates among the stations. A station that has the token is allowed to transmit information, and other stations wait until they receive the token [19].

In contention-based MAC protocols, users do not take turns transmitting their information. A user makes his or her own decision to transmit and also faces a risk of becoming involved in a collision with another station that also decides to transmit at about the same time. If no collision occurs, the information may be successfully delivered to its destination. On the other hand, if a collision occurs, the information from all users involved in a collision will need to be retransmitted. An example of contention- based MAC protocols is carrier sense multiple access with collision detection (CSMA/CD), which is used in Ethernet. In CSMA/CD, a user senses the shared transmission medium prior to transmitting its information. If the medium is sensed as busy (someone is already transmitting the information), the user will refrain from transmitting the information; however, if the medium is sensed as free, the user transmits the information. Intuitively, this MAC protocol should be able to avoid collisions, but collisions still do take place. The reason is that transmissions travel along the transmission medium at a finite speed. If one user senses the medium at one point and finds it free, it does not mean that another user located at another point of the medium has not already begun its transmission. This is referred to as the effect of the finite propagation delay of electromagnetic signal along the transmission medium. This is the single most important parameter that causes deterioration of performance in contention-based local area networks [11,19].

Design of local area networks has also been significantly impacted by the availability of transmission media with higher data rates. As the data rate of a transmission medium increases, the effects of propagation delay become even more visible. In higher speed local area networks such as Gigabit Ethernet, and 100-BASE-FX, the medium access protocols are designed to reduce the effects of propagation delay. If special attention is not given to the effects of propagation delay, the performance of high-speed local area networks becomes very poor [15,19].

Metropolitan area networks essentially deal with the same issues as local area networks. These networks are generally used as backbones for interconnecting different local area networks. These are high-speed networks and span a relatively larger geographical area. MAC protocols for sharing the same trans-mission media are based on controlled access. The two most common examples of metropolitan area networks are fiber distributed data interface (FDDI) and distributed queue dual bus (DQDB). In FDDI, the transmission medium is in the form of two rings, whereas DQDB uses two buses. FDDI rings carry information in one but opposite directions and this arrangement improves reliability of communication. In DQDB, two buses also carry information in one but opposite directions. The MAC protocol for FDDI is based on token passing and supports voice and data communication among its users. DQDB uses a reservation-based access mechanism and also supports voice and data communication among its users [19].

21.3.3 Wireless and Mobile Communication Networks

Communication without being physically tied-up to wires has always been of interest and mobile and wireless communication networks promise that. The last few years have witnessed unprecedented growth in wireless communication networks. Significant advancements have been made in the technologies that support wireless communication environment and there is much more to come in the future. The devices used for wireless communication require certain features that wired communication devices may not necessarily need. These features include low power consumption, light weight, and worldwide communication ability.

In wireless and mobile communication networks, the access to a communication network is wireless so that the end users remain free to move. The rest of the communication path could be wired, wireless,

or combination of the two. In general, a mobile user, while communicating, has a wireless connection with a fixed communication facility and rest of the communication path remains wired. The range of wireless communication is always limited and therefore the range of user mobility is also limited. To overcome this limitation, the cellular communication environment has been devised. In a cellular communication environment, a geographical region is divided into smaller regions called cells, thus the name cellular. Each cell has a fixed communication device that serves all mobile devices within that cell. However, as a mobile device, while in active communication, moves out of one cell and into another cell, service of that connection is transferred from one cell to another. This is called the handoff process [7,16].

The cellular arrangement has many attractive features. As the cell size is small, the mobile devices do not need very high transmitting power to communicate. This leads to smaller devices that consume less power. In addition, it is well known that the frequency spectrum that can be used for wireless communication is limited and can therefore support only a small number of wireless communication connections at a time. Dividing communication regions into cells allows the use of the same frequency in different cells as long as they are sufficiently far apart to avoid interference. This increases the number of mobile devices that can be supported. Advances in digital signal processing algorithms and faster electronics have led to very powerful, smaller, elegant, and versatile mobile communication devices. These devices have tremendous mobile communication abilities including wireless Internet access, wireless e-mail and news items, and wireless video (though limited) communication on handheld devices. Wireless telephones are already available and operate in different communication environments across the continents. The day is not far when a single communication number will be assigned to every newborn and will stay with that person irrespective of his/her location.

Another field that is emerging rapidly is the field of ad hoc wireless communication networks. These networks are of a temporary nature and are established for a certain need and for a certain duration. There is no elaborate setup needed to establish these networks. As a few mobile communication devices come in one another's proximity, they can establish a communication network among themselves. Typical situations where ad hoc wireless networks can be used are in the classroom environment, corporate meetings, conferences, disaster recovery situations, etc. Once the need for networking is satisfied, the ad hoc networking setup disappears.

21.4 Resource Allocation Techniques

As discussed earlier, computer networks are resource sharing systems. Users share the common resources as transmission media, processing power and buffering capacity at the switching nodes, and other resources that are part of the networks. A key to successful operation of computer networks is a fair and efficient allocation of resources among its users. Historically, there have been two approaches to allocation of resources to users in computer networks:

- Static allocation of resources
- Dynamic allocation of resources

Static allocation of resources means that a desired quantity of resources is allocated to each user who may use it whenever he or she needs to. If the user does not use his/her allocated resources, no one else can. On the other hand, dynamic allocation of resources means that a desired quantity of resources is allocated to users on the basis of their demands and for the duration of their need. Once the need is satisfied, the allocation is retrieved. In that case, someone else can use these resources if needed. Static allocation results in wastage of resources, but does not incur the overhead associated with dynamic allocation. Which technique should be used in a given situation is subject to the concept of supply and demand. If resources are abundant and demand is not too high, it may be better to have static allocation of resources; however, when the resources are scarce and demand is high, dynamic allocation is almost a necessity to avoid the wastage of resources.

Historically, communication and computer networks have dealt with both situations. Earlier communication environments used dynamic allocation of resources when users walked to a public call office to

make a telephone call or send a telegraphic message. After a few years, static allocation of res-ources was adopted, when users were allocated their own dedicated communication channels and these were not shared among others. In the late 1960s, the era of computer networks dawned with dynamic allocation of resources and all communication and computer networks have continued with this tradition to date. With the advent of optical fiber, it was felt that the transmission resources are abundant and can satisfy any demand at any time. Many researchers and manufacturers were in favor of going back to the static allocation of resources, but a decision to continue with dynamic resource allocation was made and that is here to stay for many years to come [10].

21.5 Challenges and Issues

Many challenges and issues are related to communications and computer networks that are still to be overcome. Only the most important ones will be described in this section.

High data rates provided by optical fibers and high-speed processing available at the switching nodes has resulted in lower delay for transferring information from one point to another. However, the propagation delay (the time for a signal to propagate from one end to another) has essentially remained unchanged. This delay depends only on the distance and not on the data rate or the type of transmission medium. This issue is referred to as latency vs. delay issue [11]. In this situation, traditional feedback-based reactive traffic management techniques become ineffective. New preventive techniques for effective traffic management and control are essential for achieving the full potential of these communication and computer networks [22].

Integration of different services in the same networks has also posed new challenges. Each type of service has its own requirements for achieving a desired level of quality of service (QoS). Within the networks any attempt to satisfy QoS for a particular service will jeopardize the QoS requirements for other services. Therefore, any attempt to achieve a desired level of quality of service must be uniformly applied to the traffic inside a communication and computer network and should not be intended for any specific service or user. That is another challenge that needs to be carefully addressed and its solutions achieved [13].

Maintaining security and integrity of information is another continuing challenge. The threat of sensitive information passively or actively falling into unauthorized hands is very real. In addition, proactive and unauthorized attempts to gain access to secure databases are also very real. These issues need to be resolved to gain the confidence of consumers so that they may use the innovations in communications and computer networking technologies to their fullest [13].

21.6 Summary and Conclusions

This chapter discussed the fundamentals of communications and computer networks and the latest developments related to these fields. Communications and computer networks have witnessed tremendous growth and sophisticated improvements over the last several decades.

Computer networks are essentially resource sharing systems in which users share the transmission media and the switching nodes. These are used for exchanging information among users that are not necessarily connected directly. There has been a manifold increase in transmission rates of transmission media and the processing power of the switching nodes (which are essentially computers) has also been multiplied. The emerging computer networks are supporting communication of different types of services in an integrated fashion. All types of information, irrespective of type and source, are being transported in the form of packets (e.g., ATM cells). Resources are being allocated to users on a dynamic basis for better utilization. Wireless communication networks are emerging to provide worldwide connectivity and exchange of information at any time.

These developments have also posed some challenges. Effective traffic management techniques, meeting QoS requirements, and information security are the major challenges that need to be surmounted in order to win the confidence of users.

References

1. Bae, J., and Suda, T., "Survey of traffic control schemes and protocols in ATM networks," *Proceedings of the IEEE,* Vol. 79, No. 2, February 1991, pp. 170–189.
2. Beyda, W., *Data Communications from Basics to Broadband*, 3rd., 2000.
3. Black, U., ATM: *Foundation for Broadband Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
4. Black, U., Emerging Communications Technologies, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1997.
5. Chou, C., "Computer networks in communication survey research," *IEEE Transactions on Professional Communication,* Vol. 40, No. 3, September 1997, pp. 197–208.
6. Comer, D., *Computer Networks and Internets*, Prentice-Hall, Englewood Cliffs, NJ, 1999.
7. Goodman, D., *Wireless Personal Communication Systems*, Addison-Wesley, Reading, MA, 1999.
8. Goralski, W., *Introduction to ATM Networking*, McGraw-Hill, New York, 1995.
9. Freeman, R., *Fundamentals of Telecommunications*, John Wiley & Sons, New York, 1999.
10. Ilyas, M., and Mouftah, H.T., "Performance evaluation of computer communication networks," *IEEE Communications Magazine,* Vol. 23, No. 4, April 1985, pp. 18–29.
11. Kleinrock, L., "The latency/bandwidth tradeoff in gigabit networks," *IEEE Communications Magazine,* Vol. 30, No. 4, April 1992, pp. 36–40.
12. Kleinrock, L., "ISDN-The path to broadband networks," *Proceedings of the IEEE,* Vol. 79, No. 2, February 1991, pp. 112–117.
13. Leon-Garcia, A., and Widjaja, I., *Communication Networks, Fundamental Concepts and Key Architectures*, McGraw Hill, New York, 2000.
14. Mukherjee, B., *Optical Communication Networks*, McGraw-Hill, New York, 1997.
15. Partridge, C., *Gigabit Networking*, Addison-Wesley, Reading, MA, 1994.
16. Rappaport, T., *Wireless communications*, Prentice-Hall, Englewood Cliffs, NJ, 1996.
17. Schwartz, M., *Broadband Integrated Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1996.
18. Shay, W., Understanding Communications and Networks, 2nd ed., PWS, 1999.
19. Stallings, W., *Local and Metropolitan Area Networks*, 6th ed., Prentice-Hall, Englewood Cliffs, NJ, 2000.
20. Stallings, W., *ISDN and Broadband ISDN with Frame Relay and ATM*, 4th ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.
21. Stallings, W., *High-Speed Networks, TCP/IP and ATM Design Principles*, Prentice-Hall, Englewood Cliffs, NJ, 1998.
22. Yuan, X., A study of ATM multiplexing and threshold-based connection admission control in connection-oriented packet networks, Doctoral Dissertation, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, August 2000.

22

Fault Analysis in Mechatronic Systems

Leila Notash
Thomas N. Moore
Queen's University

22.1 Introduction

As the degree of automation increases, particularly intelligent automation, high reliability, fail-safe and fault tolerance become an essential part of the mechatronic system design. A mechatronic system is reliable if no failure and malfunction could result in an unsafe system; is safe if it causes no injury or damage to the operator, environment and system itself; is fail-safe if the system could be stopped safely after the failure; and is fault tolerant if the system could complete its task safely after any failure.

Fault/failure corresponds to any condition or component/subsystem degradation (sharp or graceful degradation) that affects the performance of a system such that the system cannot function as it is required. As the application of the mechatronic systems expands to areas such as highly dynamic/unstructured or space/remote environments, medical and high-speed applications, the necessity for the system to be fail-safe (could stop with no harm to the environment, operator, and itself) and fault tolerant (tolerate the failure and complete the assigned task) increases.

A mechatronic system is called fault tolerant if after any failures there will be no interruption in the task/operation of the system. Fault tolerance and high reliability could be achieved by using high quality components, through design and robust control, and by incorporating redundancy in the design of mechatronic systems. A mechatronic system consists of mechanical, electrical, computer, and control (hardware and software) subsystems. Therefore, their redundancy could be in the form of hardware redundancy (redundancy in sensing, actuation, transmission, communication, and computing), software redundancy, analytical redundancy, information redundancy, and time redundancy.

22.2 Tools Used for Failure/Reliability Analysis

The failure analysis techniques could be classified as inductive techniques and deductive techniques (Wolfe, 1978). Inductive techniques, such as decision or event trees and failure modes and effects analysis (FMEA), consider the possible states of components/subsystems and determine their effects on the system, that is,

identify the undesired state. Deductive analyses, such as fault tree analysis (FTA), involve investigation of possible desired state of the overall system and identify the component states that contribute to the occurrence of the undesired state, that is, describe how the undesired state is achieved.

The event tree method is a pictorial representation of all the events (success or failures) that can occur in a system. Similar to other techniques, the event tree method can be used for systems in which all subsystems/components are continuously operating. This method is also widely used for systems in which some or all of the subsystems/components are in a standby mode with sequential operational logic and switching, such as safety oriented systems (Billinton and Allan, 1983).

FMEA is a bottom-up qualitative technique used to evaluate a design by identifying possible failure modes and their effects on the system, occurrence of the failure modes, and detection techniques. The history of FMEA goes back to the early 1950s when the technique was utilized in the design and development of flight control systems (Dhillon, 1983). Since then it has been widely used in the industry for specific designed systems with known knowledge of their components, subsystems, functions, required performance and characteristics, and so on. Criticality analysis (CA) is a quantitative method used to rank critical failure mode effects by taking into consideration the probability of their occurrence. FMECA is a design technique composed of FMEA and CA and provides a systematic approach to clarify hardware failures.

Fault tree analysis (FTA) is a top-down procedure which considers components in working or failed states, and it has been proven difficult to handle degraded component states. FTA can be used to obtain minimum cut sets, which define the modes of system failures and identify critical components. The reliability measures for the top event of FTA can be obtained provided that the failure data on primary events/failures is available.

22.3 Failure Analysis of Mechatronic Systems

The failure modes of a mechatronic system include failure modes of mechanical, electrical, computer, and control subsystems, which could be classified as hardware and software failures. The failure analysis of mechatronic systems consists of hardware and software fault detection, identification (diagnosis), isolation, and recovery (immediate or graceful recovery), which requires intelligent control.

The hardware fault detection could be facilitated by redundant information on the system and/or by monitoring the performance of the system for a given/prescribed task. Information redundancy requires sensory system fusion and could provide information on the status of the system and its components, on the assigned task of the system, and the successful completion of the task in case of operator error or any unexpected change in the environment or for dynamic environment.

The simplest monitoring method identifies two conditions (normal and abnormal) using sensor information/signal: if the sensor signal is less than a threshold value, the condition is normal, otherwise it is abnormal. In most practical applications, this signal is sensitive to changes in the system/process working conditions and noise disturbances, and more effective decision-making methods are required. Generally, monitoring methods can be divided into two categories: model-based methods and feature-based methods. In model-based methods, monitoring is conducted on the basis of system modeling and model evaluation. Linear, time-invariant systems are well understood and can be described by a number of models such as state space model, input–output transfer function model, autoregressive model, and autoregressive moving average (ARMA) model. When a model is found, monitoring can be performed by detecting the changes of the model parameters (e.g., damping and natural frequency) and/or the changes of expected system response (e.g., prediction error). Model-based monitoring methods are also referred to as failure detection methods.

Model-based systems suffer from two significant limitations. First, many systems/processes are non-linear, time-variant systems. Second, sensor signals are very often dependent on working conditions. Thus, it is difficult to identify whether a change in sensor signal is due either to the change of working conditions or to the deterioration of the process.

Feature-based monitoring methods use suitable features of the sensor signals to identify the operation conditions. The features of the sensor signal (often called the monitoring indices) could be time and/or

frequency domain features of the sensor signal such as mean, variance, skewness, kurtosis, crest factor, or power in a specified frequency band. Choosing appropriate monitoring indices is crucial. Ideally the monitoring indices should be: (i) sensitive to the system/process health conditions, (ii) insensitive to the working conditions, and (iii) cost effective. Once a monitoring index is obtained, the monitoring function is accomplished by comparing the value obtained during system operation to a previously determined threshold, or baseline, value. In practice, this comparison process can be quite involved. There are a number of feature-based monitoring methods including pattern recognition, fuzzy systems, decision trees, expert systems, and neural networks.

Fault detection and identification (FDI) process in dynamic systems could be achieved by analytical methods such as detection filters, generalized likelihood ratio (which uses Kalman filter to sense discrepancies in system response), and multiple mode method (which requires dynamic model of the system and could be an issue due to uncertainty in the dynamic model) (Chow and Willsky, 1984).

As mentioned above, the system failures could be detected and identified by investigating the difference between various functions of the observed sensor information and the expected values of these functions. In case of failure, there will be a difference between the observed and the expected behavior of the system, otherwise they will be in agreement within a defined threshold. The threshold test could be performed on the instantaneous readings of sensors, or on the moving average of the readings to reduce noise.

In a sensor voting system, the difference of the outputs of several sensors and each component (sensor or actuator) is included in at least one algebraic relation. When a component fails, the relations including that component will not hold and the relations that exclude that component will hold. For a voting system to be fail-safe and detect the presence of a failure, at least two components are required. For a voting system to be fail-operational and identify the failure, at least three components are required, for example, three sensors to measure the same quantity (directly or indirectly). As Chow and Willsky (1984) pointed out, for the detection and identification of a single failure among m components at least $(m - 1)$ relations are required (more relations are preferred for better performance in the presence of noise).

22.4 Intelligent Fault Detection Techniques

The fault tolerant control (robust control and decision-making process) should include allowable performance degradation in the failed state, criticality and likelihood of the failure, urgency of response to failure, tradeoffs between correctness and speed of response, normal range of system uncertainty, disturbance environment, component reliability vs. redundancy, maintenance goals (mean-time-to-failure, mean-time-to-repair, maintenance-hour/operation-hour, etc.), system architecture, limits of manual intervention, and life-cycle costs (Stengel, 1991).

Fault detection could be achieved by redundancy in sensing (measurement) and actuation, parallel redundancy (e.g., dual sensors or actuators), analytical redundancy, and artificial intelligence (expert systems, artificial neural network, or integration of both techniques) combined with redundancy.

Stengel (1991) classified the analytical redundancy into direct and temporal redundancy. Direct redundancy consists of algebraic relationship among instantaneous outputs of sensors and is useful for sensor failure detection, but not for actuator failure detection. Temporal redundancy includes the relationship among histories of sensor outputs and actuator inputs (also comparison of the outputs of dissimilar sensors at different times). Temporal redundancy could be used for both sensor and actuator FDI, for example, a sensor voting system with mixed displacement and velocity sensors could detect failures of both types of sensors. The computational complexity of temporal redundancy is higher compared to the direct redundancy case as it requires the dynamics of the system.

An expert system embodies in a computer the knowledge-based component of an expert skill in such a manner that the system can generate intelligent actions and advice and can, when required, justify to the user its line of reasoning. In general, an expert system is composed of three parts: an inference engine, a human-machine interface, and a knowledge base. The inference engine is the knowledge processor and is modeled after the expert's reasoning. The engine works with available information on a particular problem, coupled with the knowledge stored in the knowledge base to draw conclusions or recommendations.

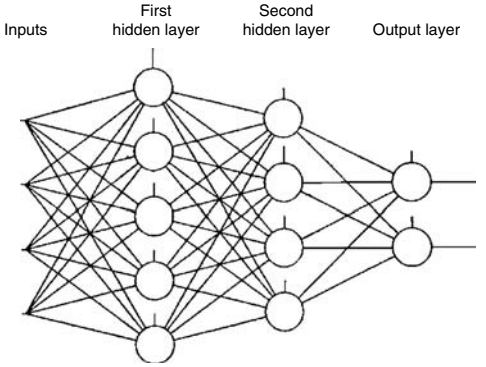


FIGURE 22.1 Architecture of a typical multilayer feedforward neural network.

The knowledge base contains highly specialized knowledge on the problem area as provided by the expert in the form of statistical analysis, empirical or semi-empirical rules, theoretic and computer simulation studies, and experimental testing. It includes problem facts, rules, concepts, and relationships.

Expert systems have obvious knowledge representation forms that make knowledge easy to manage, have the capability to explain their behavior, and can diagnose new faults using their knowledge bases. At the same time, self-learning is still a problem and computation time can be quite lengthy for difficult tasks.

A neural network is a highly nonlinear system with adaptation and generalization capabilities. There are many different architectures of neural networks; however, the multilayer feedforward neural network (refer to Figure 22.1) is one of the most popular ones. This is because of the simplicity, availability of efficient learning methods, generalization capabilities, and noise tolerance of these networks. This network is a collection of simple, interconnected nodes, also known as neurons, which operate in parallel and store knowledge on the strength of connections between the individual nodes. Such a parallel computing network, inspired by the computational architecture of the human brain, has been successfully applied to intelligent tasks such as learning, speech synthesis, and pattern recognition. The input vector feeds into each of the first layer neurons, the outputs of this layer feed into each of the second layer neurons, and so on. The last layer, which generates output to the external world, is called the output layer. The hidden layers are not connected to the external world. Often the neurons are fully connected between the layers, that is, every neuron in layer l is connected to every neuron in layer $l + 1$.

Training a neural network consists of the process of finding the set of interconnection weights (there is an interconnection weight associated with each neuron which modifies the input signal to that neuron in a specific manner), which results in a network output that satisfies a predefined criterion. Feedforward neural networks are trained using the backpropagation algorithm. This is a supervized training method. This means that the network will be presented with sample inputs and correct responses, called a training pattern. The network is then trained to reproduce the correct responses.

Neural networks have capabilities of association, memorization, error tolerance, self-adaptation, and multiple complex pattern processing. However, they cannot explain their own reasoning behavior and cannot diagnose new faults (those not already made available previously in training the network).

22.5 Problems in Intelligent Fault Detection

The fault detection scheme should be capable of detecting and identifying the failures correctly and promptly with minimum delays. This requires a reconfigurable robust controller. That is, the controller should distinguish between failures, uncertainties/inaccuracies in the model of the system, and disturbances such as sensor noise; and reduce the effect of measurement error and noise, uncertainties in the system model, and disturbances (even component failure) on the system output.

The sensor noise could be taken care of by statistical analysis on sensor readings. The uncertainties in the system model could be taken care of by estimating the effect of parameter uncertainties and compensating for it in the FDI system, or by minimizing the sensitivity of the FDI system to these uncertainties. The detection scheme should also be capable of monitoring the degradation of the system, as well as evolution and progress of failure over time (and predicting the failure), and responding to each accordingly.

22.6 Example Mechatronic System: Parallel Manipulators/Machine Tools

Parallel structured machine tools consist of multiple serial branches/legs acting in parallel on a common mobile platform with the spindle being connected to the mobile platform. Parallel manipulator-based devices have the advantages of not requiring actuation of base distal joints and of having their active joints acting in parallel on the mobile platform. These advantages can lead to parallel machine tools having desirable stiffness, accuracy, and dynamic characteristics, which, in turn, will provide high material removal rate (high product volume) with tight tolerances and in-process inspection capability (on-machine measurements of workpieces, fixtures, and tools during and after manufacturing process without breaking setups).

The failure analysis of parallel machine tools should include failures of parallel architecture, as well as failure of cutting tool, in addition to software failures.

22.6.1 Parallel Architecture Manipulators (Based on a Paper by Huang and Notash, 1999)

The following discussion will focus on the design orientated failure analysis of the mechanical system of parallel manipulators/machines.

Parallel manipulators consist of a base platform (stationary link), a mobile platform (end effector), and multiple branches/legs connecting the base and mobile platforms. Figure 22.2 depicts an example of a six-branch parallel manipulator.

The mechanical failure modes of manipulators could be classified as joint failure (component), link failure/breakage (component), branch failure (subsystem), end effector failure (subsystem), and device failure (system). Figure 22.3 represents the top level FTA of a three-branch parallel manipulator/machine.

22.6.1.1 Component Failures

Parallel (closed-loop) manipulators possess both active joints (joints that are sensed and actuated) and passive joints (unactuated joints which could be sensed or unsensed). Therefore, their failures could be due to the failures of active, passive sensed, or passive unsensed joints. The failure of any joint will cause

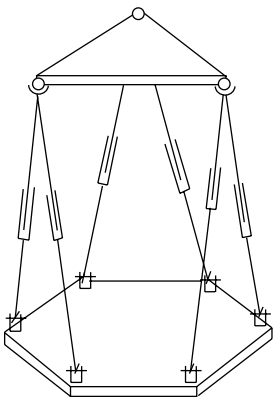


FIGURE 22.2 Example of a six-branch parallel manipulator/machine.

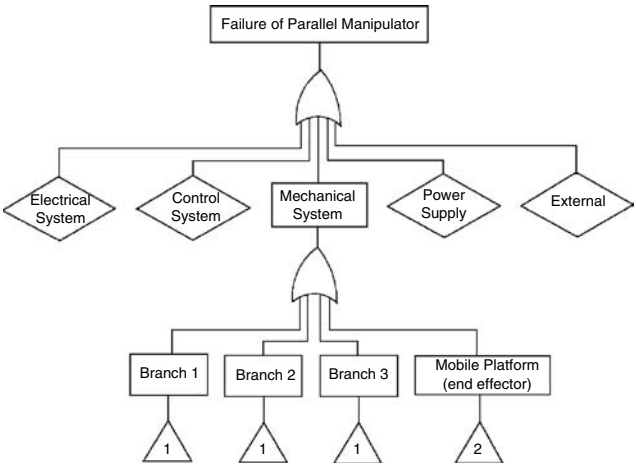


FIGURE 22.3 Top level fault tree for a three-branch parallel manipulator.

the failure of the parallel manipulator, unless the device is redundantly actuated/sensed or has a redundant unsensed joint(s) for the given task.

The common failure modes of active, passive sensed, and passive unsensed joints are the joint break and joint jam. The only failure modes of passive unsensed joints are the common joint failures. The failure modes of passive sensed joints include sensor failure, in addition to the common failure modes. In this case, the motion of the joint cannot be measured and the joint will be reduced to a passive unsensed joint. The major failure modes of active joints could be classified as actuator failure, transmission failure, and sensor failure. As a result of an actuator failure, the active joint degrades to a passive sensed joint, provided that the joint is back drivable; otherwise, the joint must be locked and the corresponding branch and the parallel manipulator will lose one DOF and an actuation. Because of a transmission failure, the actuator fails to drive the joint, and the active joint could only be used as a passive sensed joint. When the sensor of an active joint fails, although the actuator may operate properly, the motion of the joint cannot be controlled as there will be no reliable information available on the joint motion; hence, the active joint is degraded to a passive unsensed joint.

22.6.1.2 Subsystem Failures

The branches of a parallel device could be categorized as active or passive branches. An active branch possesses at least one active joint to provide a required force and to facilitate a suitable loci for the branch end location. A passive sensed branch has at least one sensed joint and its main function is to constrain the loci of the branch end position. Neither joint of a passive unsensed branch is sensed, and the branch is mainly used to constrain the motion of the mobile platform.

A branch of a parallel manipulator could fail because of component (link, joint) failures. As well, a branch will not follow its assigned path if it is in the workspace boundary, or at an internal singularity (where it loses one or more DOF). Therefore, the mechanical failure modes of a branch include branch break, loss of DOF, and loss of sensing/actuation.

22.6.1.3 Mechanical System Failures

A parallel manipulator could fail because of component and/or subsystem failures. Therefore, the mechanical failures of a parallel manipulator include loss of the DOF, loss of the actuation, loss of the motion constraint, and uncertainty configurations. A summary of the mechanical failure levels, modes, effects, and causes of parallel manipulators has been tabulated in Table 22.1.

TABLE 22.1 Failure Modes of Parallel Manipulators and Their Effects

Failure Levels			Failure Modes	Failure Causes	Effects
Components	Links		Break	Overload, fatigue, impact, material flaw	Reduction in number of branches
	Joints	Common	Joint break	Overload, fatigue, impact, material flaw	Reduction in number of branches
			Joint jam	Deterioration, external interference	Reduction in DOF of corresponding branch
		Active	Actuator failures	Depends on actuator type	Reduction in actuation, DOF if joint not back-drivable
			Transmission failures	Depends on transmission type	Reduction in actuation, DOF if joint not back-drivable
			Sensor failures	Depends on sensor type	Reduction in sensing, actuation, maybe DOF
		Passive sensed	Sensor failures	Depends on sensor type	Reduction in sensing
		Passive unsensed	Common failures (break, jam)	Overload, fatigue, impact, material flaw; deterioration, external interference	Reduction in number of branches; reduction in DOF
Branches	Common		Break	Joint/link break	Reduction in number of branches, maybe actuation and DOF, interference with other branches
			Loss of DOF	Joint jam, locked active joint, branch singularity	Reduction in DOF of manipulator
	Active		Loss of actuation	Active joint failure	Reduction in actuation, maybe DOF
			Loss of sensing	Sensor failure	Reduction in actuation, degradation to passive branch
	Passive sensed		Loss of sensing	Sensor failure	Reduction in sensing, degradation to passive unsensed branch
	Passive unsensed		Common failures (break, loss of DOF)	Joint/link break, joint jam, locked active joint, branch singularity	Reduction in constraint or DOF of manipulator
Manipulator			Loss of DOF	Joint jam, branch singularity, branch interference	Insufficient DOF
			Loss of actuation	Active joint/branch failure	Degradation in force and motion capabilities
			Loss of constraint	Reduction in number of active branches	Uncontrolled motion of manipulator
				Passive unsensed branch break	Uncontrolled motion of manipulator
				Uncertainty configuration	Instantaneous uncontrolled motion of manipulator

22.6.1.4 Failure Identification

A fault tolerant manipulator should be capable of identifying a failure, as well as tolerating the failure. The failed component (mechanical system) of a parallel manipulator, e.g., a failed joint sensor, could be identified via the manipulator controller using the information provided by the sensors of the device. A joint sensor fault detection scheme for a class of fault tolerant parallel manipulators, based on redundant sensing of joint displacements and the comparison of forward displacement solutions, was presented in (Notash, 2000).

While the failure of active joints could be identified based on the information provided by the sensor(s) on the corresponding joint, failure of passive joints could be identified by monitoring the overall performance of the manipulator in the software. For a given parallel manipulator, the criteria for failure should be incorporated in the simulation software. For example, the loss of DOF due to workspace boundary could be monitored (similar to the joint limits and branch interference) and the manipulator could be stopped before it reaches its envelope to prevent potential failure and damage to the device. As well, all of the potential special (uncertainty) configurations of the manipulator should be identified, and the closeness to these singularities should be monitored as the device moves around within its workspace.

22.6.1.5 Fault Tolerance through Redundancy

The fault tolerant capabilities of parallel manipulators could be improved by employing appropriate redundancies. Redundant sensing has been investigated for improving the fault tolerance capabilities of parallel manipulators, for simplifying the forward displacement analysis of these manipulators, and for facilitating fixtureless calibration of these devices. Redundancy in actuation has been considered for eliminating the uncertainty configurations of parallel manipulators. More work is required to develop methodologies for identifying the failed components of parallel manipulators with elements of redundancy, and compensating for their failures. For parallel manipulators, redundancy could be incorporated as redundant DOF (mobility), redundant sensing, and redundant actuation.

Redundant DOF could be achieved by incorporating additional joints into the parallel manipulator. A redundant DOF requires one more actuator on the parallel manipulator. This additional actuator is not considered as a redundant one because its failure will result in the failure of the parallel manipulator due to the loss of a required actuation. Redundancy in sensing could be obtained by sensing the existing passive unsensed joints of the manipulator, by adding a redundant passive sensed branch, or by using an external sensor such as a vision system. It should be noted that the information redundancy is achieved by redundant sensing, as well as by providing the task description of the manipulator, such as the Cartesian trajectory of the end effector (for robot path planning and machining operation). Redundancy in actuation could be accomplished by actuating the passive joints of the manipulator, or by adding an active branch (in addition to employing dual actuators).

22.6.2 Tool Condition Monitoring

An important element of the automated process control function is the real-time detection of cutting tool failure, including both wear and fracture mechanisms in machining operations. The ability to detect such failures online would allow remedial action to be undertaken in a timely fashion, thus ensuring consistently high product quality (quality of surface finish and dimensional precision) and preventing potential damage to the process machinery. The basic premise of any automated, real-time tool condition monitoring system is that there exists either a directly measurable, or a derived parameter, which can be related to advancing tool wear and/or breakage. Information about tool wear, if obtained online, can be used to establish tool change policy, adaptive control, economic optimization of machining processes, and full automation of machining processes.

In the ideal case, the system should be able to detect levels of wear well below those at which the tool would have to be replaced and should also be sensitive to relatively small changes in the level of wear. The latter characteristic would provide the system with the potential to "trend" the wear pattern and predict the amount of useful life left in the tool (allowable wear limit reached).

With respect to tool fracture, the system should be able to detect both small fractures, "chipping" phenomena, and catastrophic failure of a tool. Although prediction of such failures would be desirable, it is problematic whether this is a practical goal, at least in the near future. The number of variables, which determine the actual occurrence of tool fracture together with their complex interactions, and in many instances their underlying stochastic nature, make reliable prediction capabilities, at best, a long-term prospect in tool monitoring systems.

22.6.2.1 Cutting Tool Failure Monitoring Techniques

Tool condition monitoring systems are based upon either direct or indirect methods of quantifying the magnitude of tool failure.

The direct methods are those that utilize effects caused directly by tool failure. The direct methods, usually performed by means of optical, radiometric, pneumatic, or contact sensors can be effectively applied to the offline measurement of tool wear or breakage. However, such direct means of measuring tool failure have generally been found to be difficult to apply in practical shop floor applications. This is particularly true in those situations requiring online (real-time) monitoring capability.

Indirect methods of sensing tool failure depend upon the measurement of parameters, which are indirectly related to the condition of the cutting edge. For example, the cutting forces generated during a machining operation are dependent upon the condition of the tool's cutting edge. Generally, as the tool edge wears the generated cutting forces increase. Thus, measurement of the cutting forces present during a machining operation provides an indication of the tool condition, i.e., increasing cutting forces indicates increasing wear. In reality the relationship can be very complex. Other parameters that have been studied to determine their suitability as indicators of cutting tool failure include spindle motor current, acoustic emissions, cutting tool temperature, and noise and vibration signals. It is also possible to measure cutting forces directly and then relate these values to the condition of the cutting tool. In fact, this is one of the more common indirect tool wear monitoring methods. It has been reported that cutting force signals are more sensitive to tool wear than vibration or power measurements. The general reliability of force measurements is another reason for their popularity in tool condition monitoring applications. To use cutting force measurements for practical tool monitoring systems, there is a need to relate these forces to the state of tool condition online. However, since the measured cutting forces are affected by both cutting edge condition and changes in cutting conditions (feed rate, cutting speed, and depth of cut), the detection of tool failure using measurements of these forces becomes quite challenging in practice.

22.6.2.2 System Characteristics

Whether a tool condition monitoring system employs direct or indirect measures of tool failure (automated, computer-based system or not), it must include a number of common features if it is to be truly practical. Figure 22.4 shows the block diagram of a generalized tool condition monitoring system (Braun, 1986).

In the *measurement* section, the physical parameter (or possibly, parameters) of interest is converted to a form that is appropriate for further manipulation by the system (generally a digitized representation of an electric analog signal).

Within the *processing* section, various techniques are implemented in order to suppress noise, compress information, and emphasize important features of the acquired signal. Typical methods include analog

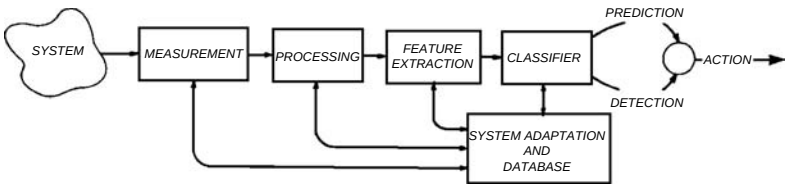


FIGURE 22.4 Block diagram of a generalized tool condition monitoring system.

or digital filtering, time domain averaging, Fourier transformation, parametric identification based on ARMA models, etc.

The purpose of the *feature extraction* section is to obtain a specific feature, or features, (often referred to as a feature vector), which can be used by the *classifier* to determine the specific type of failure and initiate appropriate corrective actions. Examples of features would include total power of the signal, crest factor value, power in a particular frequency range, frequency of the maximum peak, amplitude of the maximum peak, the autoregressive parameters of an ARMA model, etc. If multiple features are employed, they should be uncorrelated so that they provide independent indications of tool failure. When coupled with a hierarchical decision tree structure (or other appropriate structure) in the *classifier*, such multiple feature vectors can greatly improve the reliability of the tool monitoring system.

The *adaptation and database* section should not only efficiently manage all data storage and manipulation requirements but also provide the system, to as great a degree as possible, with the ability to learn from experience.

22.7 Concluding Remarks

It should be noted that the first and most practical step for increasing the reliability and improving the fault tolerance of mechatronic systems, for example, a parallel machine, is by enhancing the existing design, or by improving the robustness of the design, such as using coupled joints while designing the architecture of the manipulator. Redundancies through redesign are recommended for the applications where the fail-safe system could be very crucial, or the down time should be minimum and previously scheduled, such as the medical applications or space operation. It is also worth mentioning that not any redundancy could improve the fault tolerance of a system with no modification to the architecture of the device.

It is noteworthy to emphasize the importance of fail-safe simulation software and controller for a fault tolerant mechatronic system, which requires a robust software capable of monitoring the performance of system and responding to any system failures (including mechanical, electrical, and control systems).

References

Billinton, R., and Allan, R.N., *Reliability Evaluation of Engineering Systems: Concepts and Techniques,* Plenum Press, 1987.

Braun, S. ed., *Mechanical Signal Analysis—Theory and Applications,* Academic Press, 1986.

Chow, E.Y., and Willsky, A.S., "Analytical redundancy and the design of robust failure detection systems," *IEEE Trans. Automatic Control,* 29(7), 603–614, 1984.

Dhillon, B.S., *System Reliability, Maintainability and Management,* Petrocelli Books, 1983.

Huang, L., and Notash, L., "Failure analysis of parallel manipulators," *Proc. 10th IFToMM Congress on Theory of Machines and Mechanisms,* pp. 1027–1032, June, 1999.

Notash, L., "Joint sensor fault detection for fault tolerant parallel manipulators," *J. Robotic Systems,* 17(3), 149–157, 2000.

Stengel, R.F., "Intelligent failure-tolerant control," *IEEE Control Systems,* pp. 14–23, June 1991.

Wolfe, W.A., "Fault tree analysis," Atomic Energy of Canada, Report, 1978.

23

Logic System Design

M. K. Ramasubramanian
North Carolina State University

23.1 Introduction to Digital Logic

In analog electronics, voltages and current represent variables that vary continuously from the allowable minimum to the maximum. These variables are measured, amplified, added, and subtracted through analog circuits to achieve the desired results. For instance, measurement of temperature using thermocouples requires the amplification of voltages generated to a suitable range, calibration of the voltage with measured temperatures, and outputting the results on a voltmeter to indicate temperature. In this design, it may be necessary to subtract an offset voltage, multiply with a gain factor depending on the temperature range. The amplification of voltages and current are accomplished easily with operational amplifiers and transistors, respectively. The measured temperature can be used as the feedback signal in a control loop for a mechatronic temperature control system. In digital electronics, the variables assume a binary state, assuming a value of 0 or 1. In the above example, we might want to shut the solenoid valve down if the temperature was below desired value and open the valve if the temperature was above that value. In this case, we simply require a TRUE or FALSE input to the question "Is the temperature above or below the threshold?" The representation of these types of variables in circuits, which assume binary values, and their manipulation to achieve desired results is the topic of discussion in this chapter.

23.1.1 Logic Switching Levels

In digital circuits, voltage levels indicate binary states where the HIGH or TRUE state is represented by the maximum voltage value, typically 5 V, and the LOW or FALSE state is represented by the minimum voltage value, typically 0 V. In Boolean logic,"1" represents TRUE and "0" represents FALSE. In practice, any voltage above a minimum input threshold, V_{IH}, is interpreted as logic HIGH and any voltage below

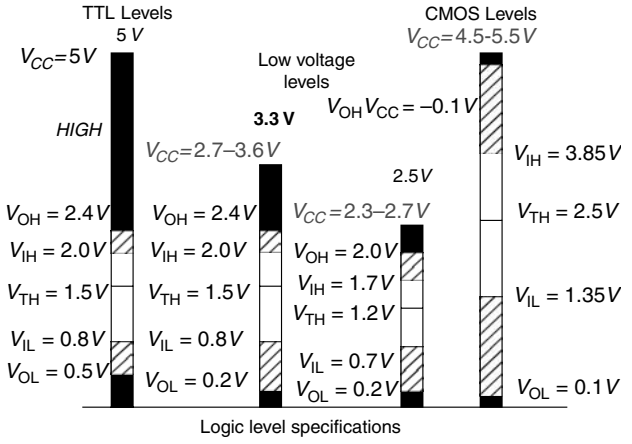


FIGURE 23.1 Switching levels for logic gates [1].

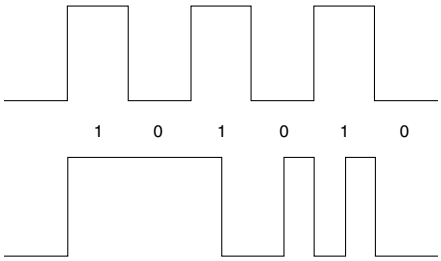


FIGURE 23.2 Periodic and nonperiodic logic level signals.

a maximum threshold, V_{IL}, is interpreted as logic LOW. The minimum output from a logic device for HIGH is represented by V_{OH} (different from V_{IH}), and maximum output level for a logic LOW is represented by V_{OL} (different from V_{IH}). These values depend on the type of logic device and a general chart of values for these parameters is shown in Figure 23.1 [1]. V_{CC} is the supply voltage. The difference between the V_{OH} and V_{IH}, or V_{OL} and V_{IL}, is called the noise margin. It is important to design the logic circuit with the constraint that voltages will never fall in the region between V_{IH} and V_{IL}, which is called the forbidden region where the logic device will fail to interpret signals. The differences between switching levels for different technologies such as 5-V logic, 3.3-V logic, CMOS (complementary metal oxide semiconductor), and TTL (transistor–transistor logic) should all be considered when interfacing these systems with each other.

A logic variable can rapidly change states as shown by an ideal pulse train in Figure 23.2. The variables can vary periodically or nonperiodically between 0 and 1. Logic gates read these signals as inputs, perform the appropriate Boolean operations among them, and generate the correct output at desired operating speeds. Robust design and use of logic functions and its implementation in circuits is an integral part of mechatronics design.

21.3.2 Logic Gate Application

Consider the example of an autonomous robot moving about on a table surface. The robot should move towards the destination denoted by a bright light source while avoiding obstacles and at the same time not falling off the edge. Assuming that we have three digital sensors, namely, obstacle detector,

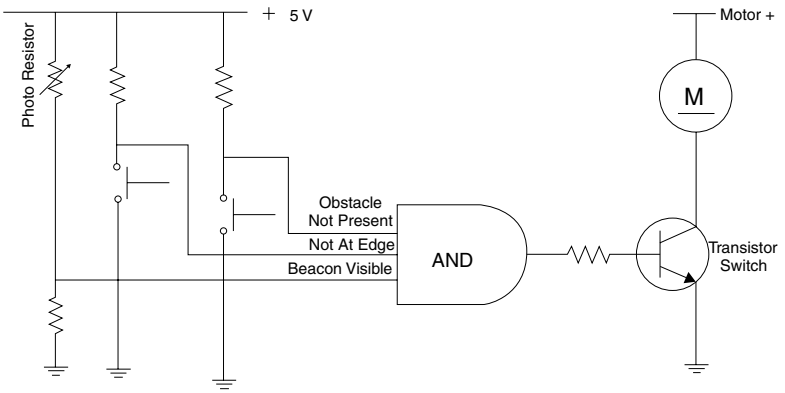


FIGURE 23.3 Forward motion logic implementation for a tabletop robot.

edge detector, and destination sensor, we can write a simple logic function for moving forward, as shown in Equation 23.1. Of course, this is not the complete logic required for the robot to function properly. However, we focus on one aspect of the problem to illustrate the use of logic functions.

$$\text{MOVE FORWARD} = (\text{OBSTACLE NOT DETECTED}) \textit{ AND } (\text{EDGE NOT DETECTED}) \textit{ AND } (\text{BEACON IS VISIBLE}) \tag{23.1}$$

The input from the three sensors is interfaced to a logic circuit consisting of logic gates, in this simple example, a three-input AND gate and the output drives the motors. Of course, other cases of behaviors for the robot where the edge is found or the beacon is not visible or an obstacle is detected have to be worked out to make this circuit robust and worthwhile. Figure 23.3 shows an implementation of the logical statement expressed in Equation 23.1.

23.2 Semiconductor Devices

23.2.1 Diode

In order to understand logic gates, it is important to develop a basic understanding of semiconductor devices, especially the diode and the transistor. A diode is a pn-junction, which means that the diode is made up of a p-type (electron deficient) material layer and an n-type (electron rich) material layer sandwiched together. When the positive terminal of a battery is connected to the p-side of the diode (anode) and the negative of the battery is connected to the n-side of the diode (cathode), then the diode is said to be forward biased as long as the voltage across the junction exceeds 0.7 V. When the terminals are reversed, the diode is said to be reverse biased and does not conduct until very high voltages are applied across the junction, known as the breakdown voltage. For all practical purposes, we can assume that a reverse-biased diode does not conduct. A schematic of a diode, its symbol, and a forward-biased circuit is shown in Figure 23.4. When forward biased, the diode can be treated as a simple closed switch with a 0.7 V drop across it and when the diode is reverse biased, the diode is an open switch.

23.2.2 Bipolar Transistor

A bipolar transistor has three semiconductor layers. In an npn-transistor, a very thin p-layer is sandwiched between two n-layers. Transistor types and their symbols are shown in Figure 23.5a,b.

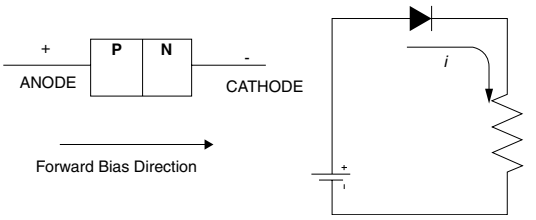


FIGURE 23.4 The diode and its behavior.

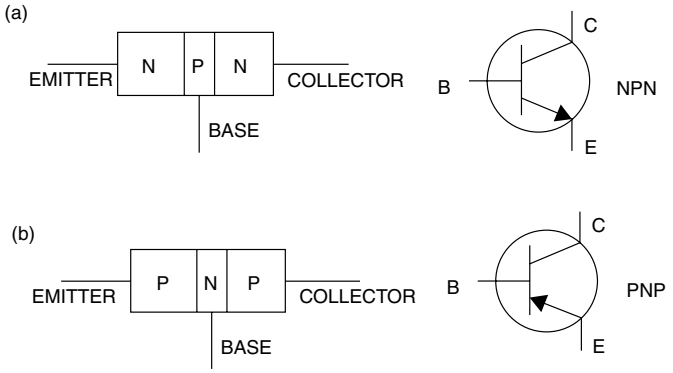


FIGURE 23.5 (a) npn-transistor symbol, and (b) pnp-transistor symbol.

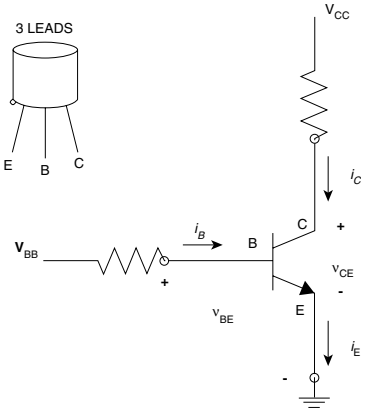


FIGURE 23.6 Schematic representation of the working of an npn-transistor.

There are three leads in a transistor, namely, the collector (C), emitter (E), and the base (B). For an npn-transistor in a circuit, as shown in Figure 23.6, the base–emitter junction is forward biased and will conduct if the voltage V_{BE} exceeds the forward bias voltage for the pn-junction, typically 0.7 V. V_{BE} is increased by increasing the voltage at B. However, the base–collector junction is reverse biased as the collector C is at a higher potential. As current flows in the base–emitter loop, the electrons from the emitter

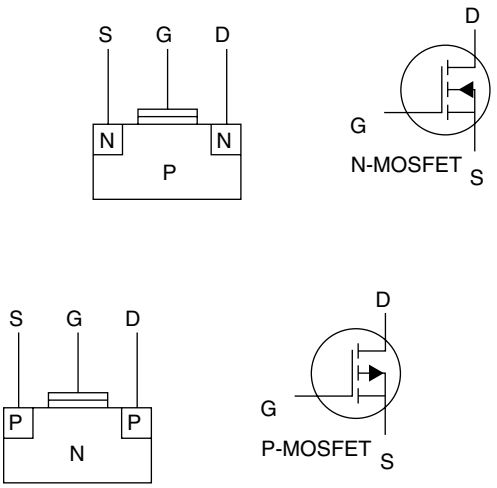


FIGURE 23.7 n- and p-channel MOSFETs and symbols.

flow into the base terminal by filling in the "holes" in the p-layer and subsequently releasing an electron from the p-layer out of the base terminal. However, because of a limited number of "holes" in the p-layer (which is very thin), the electrons from the emitter see a larger potential across the collector–emitter path and jump the junction. A large current, I_C, flows in the collector–emitter loop as a result. Thus, the transistor is a current amplifier. A small current flowing in the base–emitter loop, I_B, is amplified by typically a factor of about 100 in the collector–emitter path. As the current flow in the base–emitter is increased by increasing V_{BE}, the collector–emitter current increases by decreasing V_{CE}. Since the collector is connected to the power source, V_{CC}, and the emitter is connected to the ground, the device controls this current flow by controlling the drop in voltage across the collector–emitter junction, continuing to drop the voltage as the base–emitter current is increased. It is obvious that the voltage cannot drop below 0; in fact, it cannot drop below 0.2–0.35 V in a real device. Under these conditions, the transistor is said to be saturated and is acting as a closed switch. Circuits that are built with transistors in the saturating condition are called saturating circuits; for example, the TTL family of logic gates. Circuits that do not allow the transistor to saturate and find a stable operating point in the active region of the transistor are called nonsaturating circuits; for example, emitter-coupled logic (ECL) gates. The biggest advantage of a nonsaturating circuit is the speed with which states can be changed compared to a saturating circuit.

23.2.3 Field Effect Transistor

These devices are easier to make and uses less silicon. There are two major classes of field effect transistors (FETs), namely, the junction FET (JFET) and the metal oxide semiconductor FET (MOSFET). In both cases, a small input voltage controls the output current with practically no input current. The three terminals are called the source (S), drain (D), and gate (G). Figure 23.7 shows the symbols for the n- and p-channel enhancement type MOSFETs. MOSFET is the most popular of transistor technologies. A MOSFET gate has no electrical contact with the source and the drain. A silicon-dioxide layer insulates the gate. Electrical voltage applied at the gate attracts electrons to the region below the gate and provides an n-type channel in a p-type substrate for conduction between the drain and source. This is called the enhancement type of MOSFET. The other is the depletion-enhancement type where there is an n-channel present between the drain and source, but the channel resistance can be increased or decreased by applying either a negative or a positive voltage at the gate, respectively. Depletion-enhancement MOSFET symbols and function are described in Figure 23.8. MOSFET devices are slower than bipolar devices and are used in slower but high density circuits, due to ease of manufacture and use of less silicon.

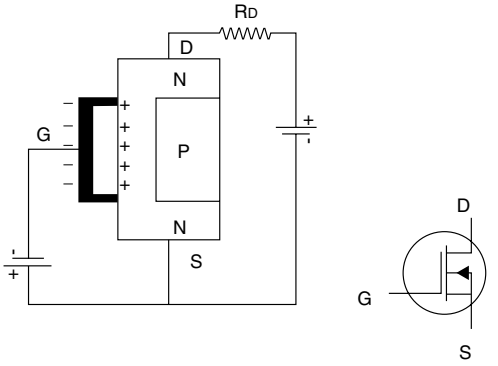


FIGURE 23.8 Depletion-enhancement type MOSFET.

23.3 Logic Gates

Logic gates are the basic building blocks of logic circuits and a computer. Mechatronic systems have a central computational element as well as specific logic functions implemented in hardware. A logic circuit consists of several logic gates working together. We will discuss the logic gates in general and as building blocks of mechatronic subsystems. Logic operations can be subdivided into two categories, namely, combinatorial and sequential. In the case of combinatorial logic circuits, the logic gates are used to produce an output based on instantaneous values of the inputs, whereas in the case of sequential logic circuits, the change in output depends on the present state as well as the state before the changes in input values, thus exhibiting memory behavior. Further, the sequential logic circuits can be synchronous or asynchronous. When the output changes synchronously with a clock input, it is said to be synchronous. When the inputs are read as soon as there is any change in it, it is called an asynchronous logic circuit.

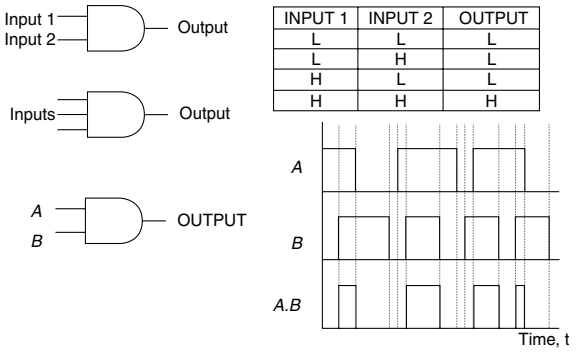
There are three fundamental logic operations, namely the AND, OR, and NOT functions. Other logic operations are derived operations from these fundamental ones. The AND gate symbol and its truth table are shown in Figure 23.9. The AND gate can have more than two inputs.

Figure 23.10 shows an OR gate. Here the output is HIGH when either of the inputs or both the inputs are HIGH. The OR gate can also have more than two inputs. Figure 23.11 shows an inverter, also known as a NOT gate. This gate takes one input and simply inverts the logic, that is, a HIGH input is returned as LOW output and vice versa.

Other common logic gates that are derived from these fundamental ones are NAND, NOR, and Exclusive OR gates. NAND gate is a combination of AND and NOT gates; NOR is a combination of OR and NOT gates, and Exclusive OR can be generated with a combination of OR, NAND, and AND gates. Figures 23.12 through 23.14 show the derived gate types, namely the NAND, NOR, and XOR gates and their truth tables, respectively. The logic functions and their implementation into hardware using gates is the basic building block of a digital computer.

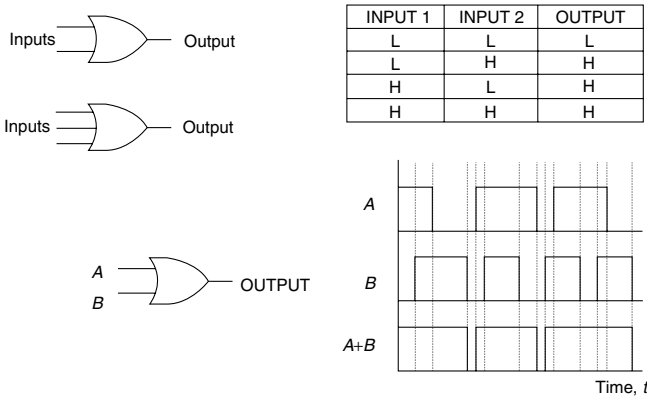
23.4 Logic Design

As in any design, it is important to keep it simple, robust, and cost effective. Mechatronics design or logic circuit design is no exception. When a logic function of a system is translated into relationships between inputs and outputs, it is not certain if the number of elements involved in realizing the design are the minimum or further simplification is possible. If the complexity is defined as the number of logic gates used, then the problem reduces to minimizing the logic function mathematically. However, if complexity is defined as the number of ICs used in the circuit (the amount of real estate occupied by the circuit), additional approaches have to be considered, namely using the same type of gate, as much



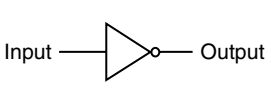
INPUT 1	INPUT 2	OUTPUT
L	L	L
L	H	L
H	L	L
H	H	H

FIGURE 23.9 AND gate, symbol, and behavior.



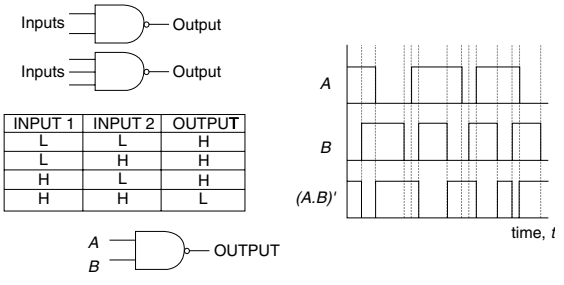
INPUT 1	INPUT 2	OUTPUT
L	L	L
L	H	H
H	L	H
H	H	H

FIGURE 23.10 OR gate, symbol, and behavior.



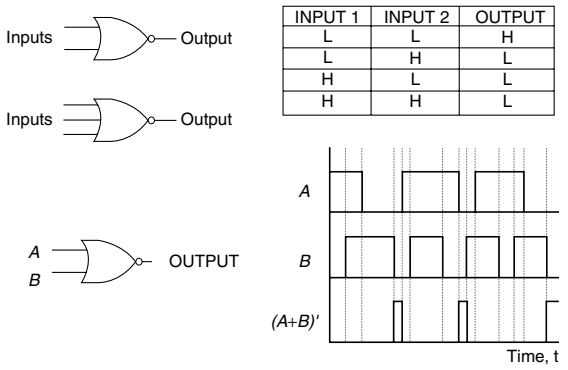
INPUT	OUTPUT
L	H
H	L

FIGURE 23.11 NOT gate or an inverter, symbol, and behavior.



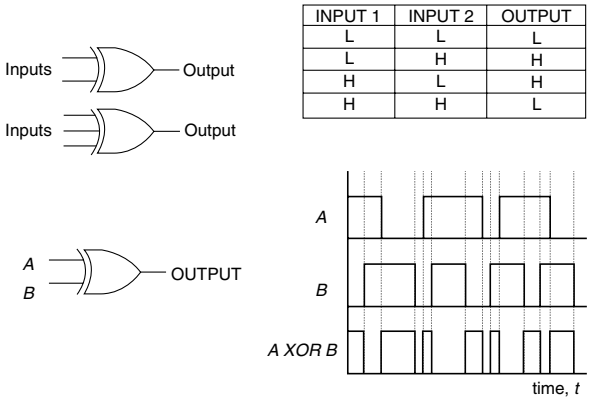
INPUT 1	INPUT 2	OUTPUT
L	L	H
L	H	H
H	L	H
H	H	L

FIGURE 23.12 NAND gate, symbol, and behavior.



INPUT 1	INPUT 2	OUTPUT
L	L	H
L	H	L
H	L	L
H	H	L

FIGURE 23.13 NOR gate, symbol, and behavior.



INPUT 1	INPUT 2	OUTPUT
L	L	L
L	H	H
H	L	H
H	H	L

FIGURE 23.14 XOR gate, symbol, and behavior.

as possible, although it may not be minimal in terms of number of gates. This will be preferred over using less different types of gates necessitating use of more ICs, in which some of the gates are unused.

23.4.1 Minimization

A method for minimizing Boolean functions is the Karnaugh map (K-map). From a physical description of the problem, logic statements are written as shown in Equation 23.1 for the tabletop robot problem. A truth table is generated showing the relationship between inputs and outputs. Let us take a truth table for a three-variable design, shown in Figure 23.15.

The logical function can be written as

$$X = A'B'C' + A'BC' + ABC' + A'BC \tag{23.2}$$

An implementation of the function without any further consideration will require four 3-input AND gates, one 4-input OR gate, and three inverters. If we assume that both complemented and uncomplemented forms of the signal for each variable are available, we still end up with a complex two-level circuit

INPUTS			OUTPUT
A	B	C	X
0	0	0	1
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	0

FIGURE 23.15 Truth table for a logic circuit design and minimization.

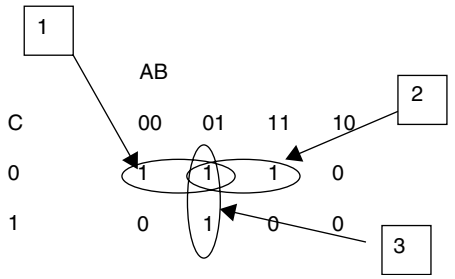


FIGURE 23.16 Karnaugh map for the logic design problem with three inputs and one output.

for what needs to be accomplished. Applying Karnaugh mapping, we can attempt to minimize the Boolean function and hence simplify the type and number of logic gates needed for circuit implementation.

The Karnaugh map is derived from the truth table shown in Figure 23.15. The two variables AB are grouped for column designations and the third variable provides the row designation. The values are arranged in such a way that adjacent columns or rows differ by only 1 bit.

Figure 23.16 essentially represents the logic described in the truth table in Figure 23.15. Because adjacent blocks in a K-map differ by 1 bit, the bit that changes is insignificant in a grouping of adjacent ones. In order to obtain the minimized function, adjacent ones on a K-map are identified by covering each one on the map at least once in a row or a column grouping, observing that in each case one variable is insignificant with respect to the value of *X*, the output. That variable is eliminated and the process is continued until all the groupings are evaluated. Finally, the reduced set of product terms is combined with an OR function to give the minimized function.

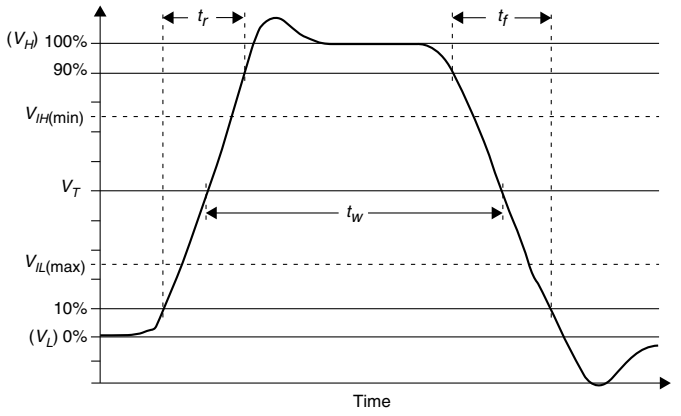


FIGURE 23.17 A real pulse and definition of characteristic parameters [2].

Figure 23.16 shows three sets of adjacent ones in rows and columns identified by circles around them and the following observations can be made:

Group 1. Only variable *B* changes states. Hence, it can be eliminated and the minimized form for the grouping is $A'C'$.

Group 2. Only variable *A* changes states. Hence, it can be eliminated and the minimized form of the grouping is BC'.

Group 3. Only variable *C* changes states. Hence, it can be eliminated and the minimized form of the grouping is $A'B$.

Hence, the minimized form for the logic function is

$$X = A'C' + BC' + A'B$$

This can be implemented with one 2-input AND gate IC and one 3-input OR gate. A K-map is helpful in minimizing up to six variables.

23.4.2 Dynamic Characteristics

Having studied the logic function and obtaining a minimum, we can build the logic circuit. However, in order to ensure that the circuit will work as intended over the entire operating range, dynamic characteristics of logic circuits must be considered. It was stated earlier that the input signal can change rapidly in a system and the logic circuit should perform as intended at frequencies at which the system is expected to operate.

The correct functioning of the logic circuit when the inputs are changing rapidly is an important consideration in design. In our discussion thus far, we have assumed that the logic signal is an ideal square wave and that the logic gates function without any delay. Let us examine the effects of relaxing these two assumptions to obtain some insight into the dynamic behavior of logic circuits.

A real pulse is shown in Figure 23.17 [2]. The rise time is denoted by t_r and fall time by t_f. The pulse further shows a settling time, overshoot, and undershoot when changing states. The signal amplitude is specified as the difference between the two stable signal levels for high (V_H) and low (V_L), that is, from 100% to 0%, and t_w is the pulse width of the signal measured at 50% of the amplitude. t_{THL}, t_{TLH} are the transition times for the output signal to go from high to low and low to high, respectively. t_{PHL} and t_{PLH} are propagation delay times for high to low and low to high transitions, respectively. For medium speed operation, t_{PHL}, and t_{PLH} are typically about 30 ns.

When an input to a logic gate changes states, the output lags behind by a characteristic time delay called the propagation delay, measured by the time difference between the input at 50% of the amplitude and the output at 50% of the amplitude. A simplified model of a real pulse for an inverter is shown in

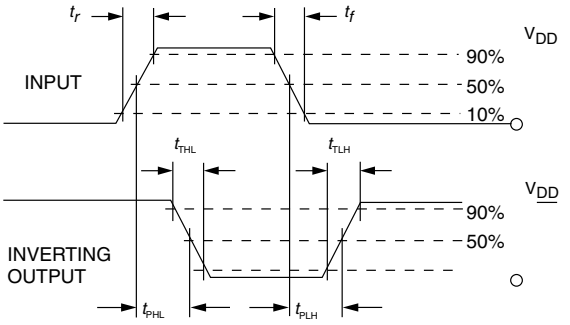


FIGURE 23.18 Propagation delay definition.

Figure 23.18 [3]. Values for the propagation, typically expressed in nanoseconds, are available in the datasheet for a device.

For a logic circuit, a propagation delay analysis is carried out by mapping out the total delay from input to output as the inputs change states, and identifying any static problems (frequency independent) and dynamic problems. Additional gates may be added to the circuit of the problem.

23.4.3 Other Design Considerations

The number of logic gates that a given gate can drive is called *fan-out* and the number of gates that can be connected to the input of a given gate is called *fan-in*. These data are given in the data sheet and should be adhered to. Further, minimizing the number of ICs needed in a logic circuit is an important consideration that might require modifying the design to use the same kind of gate although more numbers may be used than the minimum circuit identified with K-map analysis. Use of the same type of gates for compatibility between ICs is another design consideration in logic circuits.

23.5 Logic Gate Technologies

The first of the logic families that became commercially available was the resistor–transistor logic (RTL), where the transistor is used as a high-speed switch in circuits. Diode–transistor logic (DTL) and transistor–transistor logic (TTL) followed in the evolution. While the RTL and the DTL are obsolete, the TTL gates are still widely used. There are several variations of TTL logic, namely, the high-speed (H), low-power (L), Shottkey (S), and low-power Shottkey (LS). CMOS logic gates are an entirely different implementation of logic gates based on the complimentary metal-oxide semiconductor devices (CMOS) technology. These devices have low-power requirements and improved noise characteristics. These devices are extremely static sensitive and are easily damaged. A mixture of CMOS and bipolar processes resulted in the BiCMOS technology, using internal CMOS components and high-power bipolar outputs. Several different families evolved from the original BiCMOS processes [1].

23.5.1 Resistor–Transistor Logic

Figure 23.19a shows an resistor–transistor (RTL) inverter and Figure 23.19b shows a NAND gate. The transistor is assumed to operate in the saturation mode. Because of the nature of a transistor, there is a minimum voltage drop across the collector–emitter junction at saturation and there is a minimum current required in the base–emitter loop to saturate the transistor. In Figure 23.19a, the output voltage is between 0.3 and 5.0 V. The NAND gate shown has an output of 0.6 V for logic LOW as the collector–emitter drop for the two transistors in series has to be added up. If we add an additional transistor to add an input, we have additional power dissipation and the output voltage is at 0.9 V for a logic LOW causing problems

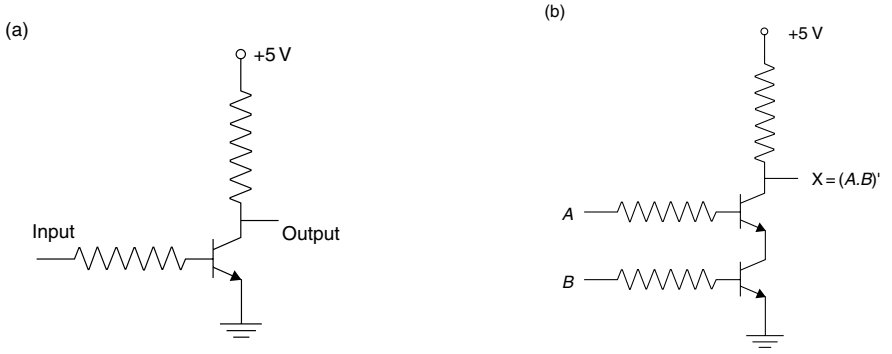


FIGURE 23.19 (a) Resistor–transistor NOT gate, (b) resistor–transistor NAND gate.

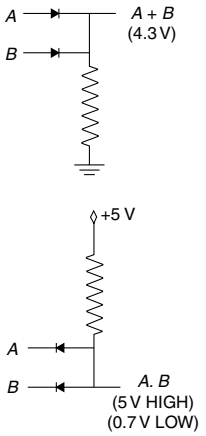


FIGURE 23.20 Diode–resistor logic.

with the logic devices that it is driving. The logic function operation becomes unreliable as output can get into the forbidden region. Further, the presence of resistors in the base–emitter loop tends to slow the device. Because of these limiting characteristics, RTL gates are obsolete.

23.5.2 Diode–Transistor Logic

Diodes themselves can be used to build logic gates for simple applications as shown in Figure 23.20. The diode drops the voltage by 0.7 V across the pn-junction when conducting, resulting in 0.7 V for a LOW and 4.3 V for a HIGH at the output. It is readily seen that the cascading of several of these circuits will push the circuit into the forbidden region, resulting in erroneous logic. Moreover, the diode resistor logic cannot implement an inverter (NOT) function, and it is not practical to produce high density ICs with diodes and resistors. Because of these shortcomings, the diode-resistor logic gate is obsolete.

A diode–transiistor logic (DTL) gate is shown in Figure 23.21. Here, the diodes are used for the OR function and the transistor is used for the NOT function to give a NOR gate. Still the presence of the resistor at the base of the transistor causes power dissipation and reduces the speed of operation. Figure 23.22 shows an improved diode–transistor design that eliminates the bias resistor, thereby improving the speed of operation. DTL devices are obsolete owing to the same limitations discussed earlier.

23.5.3 Transistor–Transistor Logic

In Figure 23.22, it can be observed that the diodes at the input are forward biased while the diode at the base of the transistor is reverse biased when any input is LOW. On the other hand, when all inputs are HIGH, the base diode is forward biased and the transistor conducts, giving the NAND function.

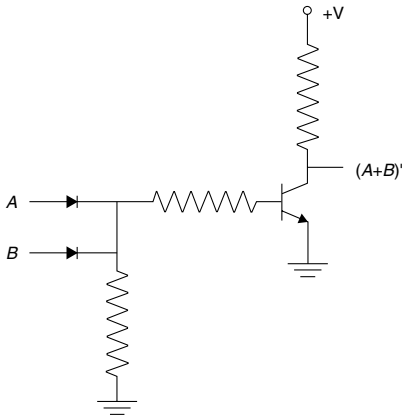


FIGURE 23.21 Diode–transistor logic NOR gate.

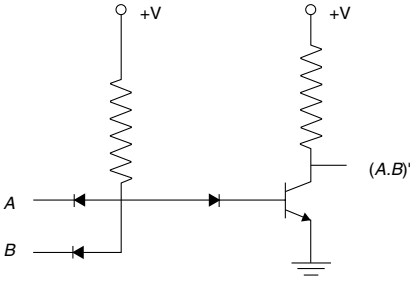


FIGURE 23.22 Improved diode–transistor NAND logic gate.

In a transistor npn, for example, the base–emitter is forward biased during conduction and collector–base is reverse biased. If we have a transistor with multiple emitter leads, then we can use the emitter–base junction for the input diodes. The base–collector junction is used for the base-diode in the DTL gate. The result is a transistor–transistor logic (TTL) gate implementation of a NAND function in Figure 23.23 [4].

Here, when any of the inputs is LOW, the base–emitter loop conducts and the emitter of the first stage transistor is at 0.2 V, giving HIGH for output at the inverter. When all the inputs are HIGH, the transistor multiple emitters (first stage) is cutoff. Therefore, all other transistors conduct with a logic LOW at output. The manufacturer's data sheet for each device provides circuit diagrams, and all technical data including maximum and minimum input values, propagation delay, rise and fall times, fan-out, fan-in limitations, power consumption, and application suggestions. These are excellent sources of information for the designer.

23.5.4 Emitter-Coupled Logic

Emitter-coupled logic (ECL) devices are bipolar devices in which the transistor is never saturated or completely shut off. The result is very high speed compared to TTL or CMOS implementations. The ECL gates are used in several applications where high speed is essential, for example, computer cache memory. Figure 23.24 shows a NOR/OR gate [5]. V_{CC} is connected to ground (0 V) while V_{EE} is connected to supply voltage, –5.2 V for better noise immunity. The transition time from one state to another is less than 1 ns, resulting in several gigahertz operating frequency when ECL gates are used.

Because the transistors are not fully saturated, the ECL gates output at HIGH and LOW are about –0.75 and –1.6 V, respectively. The bias voltage is set at the base of transistor Q_4, the value is the average

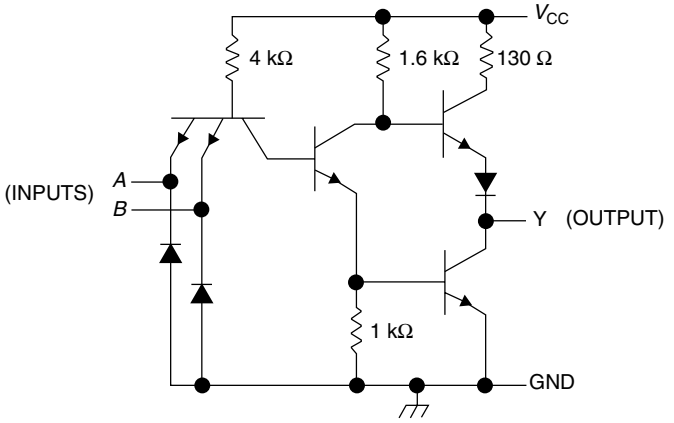


FIGURE 23.23 Transistor–transistor logic implementation of a NAND gate [4].

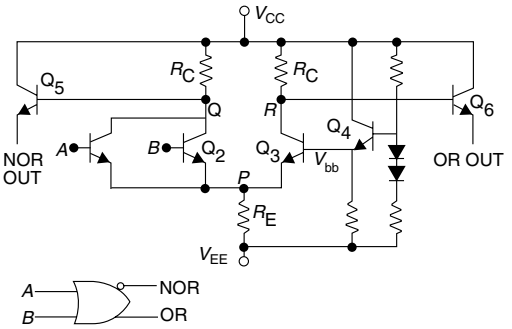


FIGURE 23.24 Emitter-coupled logic NOR/OR gate [5].

of HIGH and LOW values for the gate; in this case, the value would be –1.175 V. Next, the resistors are selected to control the current flow and prevent transistor saturation. When the signal at *A* or *B* is HIGH (–0.75 V), the transistors conduct. The voltage at point *P* becomes –1.5 V (V_{BE} of Q_2 + V_A). This reduces the difference between V_{bb} and the voltage at point *P* below the threshold for transistor Q_3 to conduct and hence it is off, raising the voltage at point *R* to 0 V. This turns transistor Q_6 on. With a V_{BE} threshold of 0.75 V, the measured OR output is –0.75 V, a logic HIGH. The value of resistance R_C and R_E are chosen so that the voltage at point Q is –0.85 V when transistor Q_1 or Q_2 is conducting. It is true when *B* is HIGH and *A* is LOW or when *A* and *B* are both HIGH. When both *A* and *B* are LOW, the transistors Q_1 and Q_2 are off and Q_3 conducts lowering the voltage at point *P* to –1.925 V. Voltage at point *R* is –0.85 V resulting in an OR output of –1.6 V at the OR output and –0.75 V at the NOR output.

Because of the constant operation of the transistors in the active region, there is continuous current draw and hence heat dissipation. The ECL devices draw four to five times the power of a comparable TTL device. Hence, this is used cautiously as front-end devices where speed is essential, while using HCMOS or TTL gates elsewhere. In order to mix ECL gates with TTL or CMOS devices, special level shifters are used, for example, National Semiconductor's 100325 Low Power ECL-to-TTL Translator. This device converts an ECL input (–0.75 (H) and –1.6 (L)) to a TTL output (2.4 V min for HIGH and 0.5 V max. for LOW), while maintaining a rise or fall time of less than 1 ns.

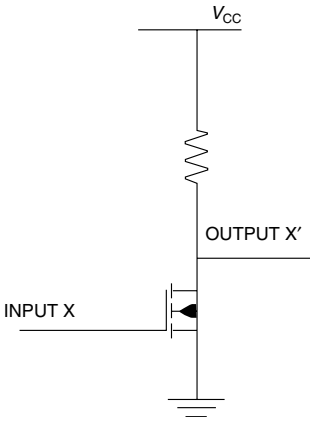


FIGURE 23.25 An NMOS inverter.

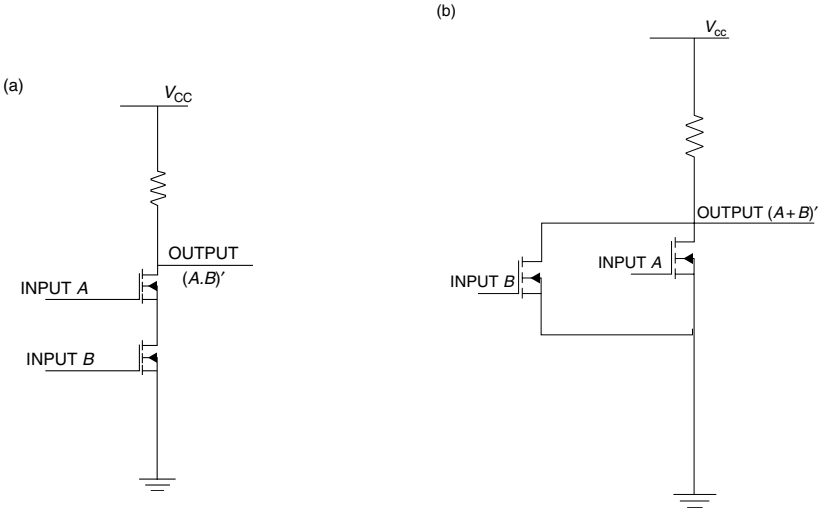


FIGURE 23.26 (a) An NMOS NAND gate, (b) an NMOS NOR gate.

23.5.5 CMOS Logic

As discussed earlier, MOSFET can be used as a transistor switch without significant power dissipation. NMOS logic gates are designed with n-MOSFETs and PMOS logic gates are designed with p-MOSFET transistors. As an example, an NMOS inverter is shown in Figure 23.25. Figure 23.26a shows a logic NAND function when two n-MOSFETs are connected in series and Figure 23.26b shows a parallel arrangement of two n-MOSFETs to give a NOR gate. The NMOS circuits shown have a pull-up resistor and a pull-down n-MOSFET. To eliminate the resistor, the pull-up side of the circuit is replaced with p-MOSFET. The modified NOT or inverter circuit is shown in a commercial implementation in Figure 23.27 [3]. Additional diodes are shown for static protection of the device. This is known as a CMOS circuit since the pull-down and pull-up parts of the circuit have complimentary MOSFET devices. When two n-MOSFETs are connected in the pull-down side of the circuit in series, the pull-up resistor is replaced by two p-MOSFETs in parallel, and vice versa. A CMOS implementation of the NOR gate is shown in Figure 23.28.

An important characteristic of CMOS gates is their low-power consumption as there is practically no current flow in both HIGH and LOW states. However, the device is slower than a bipolar transistor device.

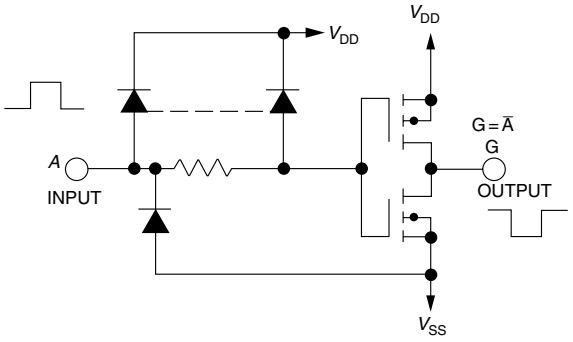


FIGURE 23.27 A CMOS inverter [3].

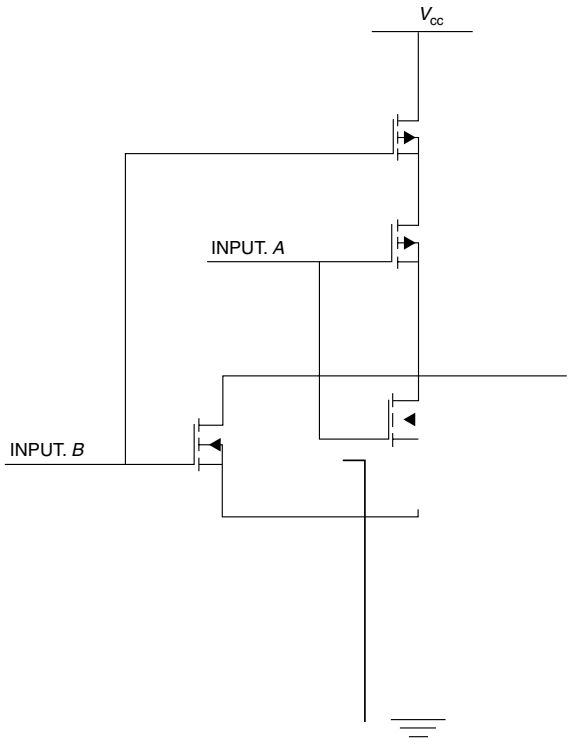


FIGURE 23.28 A CMOS NOR gate.

With decreasing transistor sizes due to advancements in fabrication technologies, the speed of CMOS devices continue to increase.

23.6 Logic Gate Integrated Circuits

A commercial logic gate ICs has several gates of the same type on it. For example, Figure 23.29 shows a commercial quad-AND gate IC. The chip itself is powered with V_{CC} and GND pins, *A* and *B* pins are inputs, and the *Y* pins are the corresponding outputs. You can use one or all of the gates on a chip as needed.

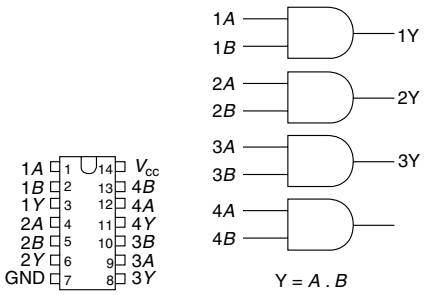


FIGURE 23.29 A DIP package of quad-AND gate IC.

The designation of the chips requires some attention. Let us take the Texas Instruments designations for an AND gate, namely, SN5408, SN54LS08, SN54S08, SN7408, SN74LS08, and SN74S08. While the designation shown on a device has much more information than what is shown here, the basic information that we should be aware of is the function designation (00 = NAND gate, 02 = NOR, 04 = Inverter, 08 = AND, etc.), and technology type (HC for high-speed CMOS, LS for low-power Schottky, etc.). For other notations used in chip designations, refer to the Texas Instruments Logic Selection Guide [1].

23.7 Programmable Logic Devices (PLD)

Programmable logic devices (PLDs) are ICs with several uncommitted logic gates in them, the connections among which are programmable based on the logic circuit design that needs to be implemented. This is especially helpful when very large circuits consisting of several thousands of logic gates have to be built and tested. For large circuit design and testing, it is not practical to use standard logic gate ICs since each IC has at most four or six logic gates on it, requiring large circuit boards and interconnects. The PLD consists of several hundred logic gates on it and the device design is programmable with a special programming hardware. When more than one PLD is used to implement a design, programmable interconnects are used between PLDs. One type of fully PLD, called the programmable logic array (PLA), consists of an AND level in the middle and an OR level at output, similar to a TTL single logic gate structure, with both layers being programmable. All input signals are connected to an inverter level, which provides both the normal and complemented values of input variables to the AND level. Appropriate connections are made at the AND level and at the OR level to produce the desired logic outputs. In this device all the levels are programmable.

A simpler version of PLD, called a programmable array logic (PAL) device, consists of a programmable AND layer and a fixed OR layer. This is easier and less expensive to manufacture, although it is not as flexible as a PLA. A variety of combinations is available to suit various needs. A schematic of a PLA is shown in Figure 23.30 [6] where the connections to be made in the hardware are marked with an X. When programmed these connections will be made or "fused" and verified by the programming hardware.

23.8 Mechatronics Application Example

A driver circuit for a DC motor is a good example for the use of transistors and logic gates. The objectives of the design are the following:

1. The motor should drive forward and reverse at different speeds.
2. The motor should either coast to a stop or brake abruptly.
3. The motor should drive at different speeds, controllable by a microprocessor.

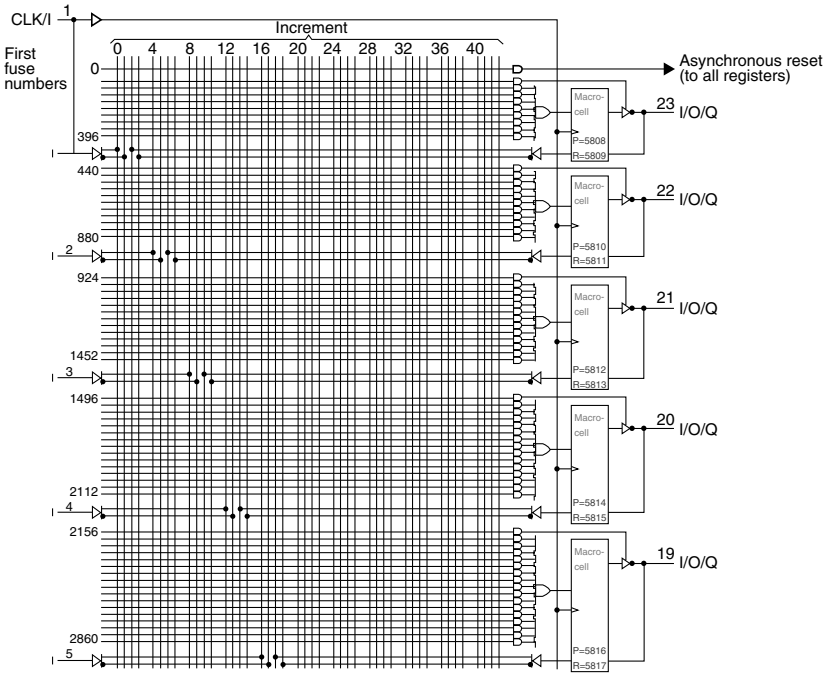
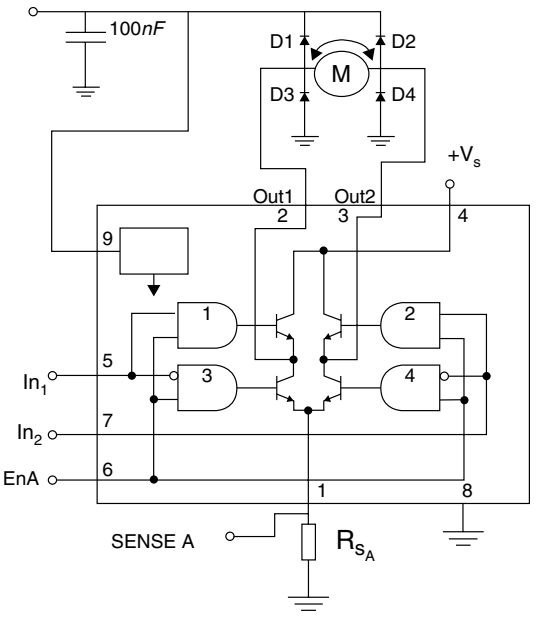


FIGURE 23.30 Programmable logic array [6].

The complete logic and power circuit implementation of the solution to this design problem is shown in Figure 23.31, which is known as the H-bridge. The motor is connected between the output pins (out1 and out2) [7]. The EN (enable) and IN1 (input 1) and IN2 (input 2) are the inputs. The behavior of the system is given by the adjacent table in Figure 23.31. When the enable signal is LOW, regardless of the input states, all the AND gates are LOW, and the power transistors are all off and the motor is off. If the motor is moving when the enable line switches to LOW, the motor coasts to a stop. When the enable input is HIGH, it can be seen that when IN1 is high and IN2 is LOW, transistors 1 and 4 are on, and 2 and 3 are off. This drives the motor one way as the current can flow through the motor to the ground through the two diagonal transistors. Since transistors 2 and 3 are off, short circuit from power to ground is prevented. This is designed by inputting the complements of IN1 and IN2 to the AND gates driving transistors 3 and 4, respectively. When IN2 is HIGH and IN1 is LOW, the motor runs in the opposite direction (while the enable is HIGH). Since transistors 2 and 4 are closed and 1 and 3 are open, current flows in the opposite direction through the motor.

When enable is HIGH, and the inputs IN1 and IN2 are either turned HIGH or LOW at the same time while the motor is moving, then the motor terminals are forced to V_{CC} or ground. However, the motor power is off since IN1 and IN2 are LOW. Now, the motor is a generator trying to maintain a potential difference across its terminal as the rotor moves in a magnetic field. The emf generated is forced to the source or sink potential. This brings the motor to a rapid stop, identified as the fast stop or the braking function. Further, the IN1 and IN2 lines can be used for direction and braking functions, while the enable can be pulsed at different duty cycle levels (pulse width modulation) to achieve different speeds. Since the motor is free running when enable is LOW regardless of input, as EN is switched rapidly, the inertia of the rotor helps smooth out the motion. The selection of pulse repetition time (PRT) and arrangement of pulses within the PRT in a uniform fashion to produce desired PWM signals should be done to fine-tune the performance of this system.



INPUTS		FUNCTION
ENA=H	In_1=H In_2=L	FORWARD
	In_1=L In_2=H	REVERSE
	In_1=In_2	FAST MOTOR STOP
ENA=L	In_1=X In_2=X	FREE RUNNING MOTOR STOP

FIGURE 23.31 H-bridge motor driver circuit [7].

References

1. "Logic Selection Guide, First Half 2001," Texas Instruments, Document sdyu001o.pdf. Source: www.ti.com.
2. "Designing with Logic," Texas Instruments, Document sdya009C.pdf. Source: www.ti.com.
3. "CD4069UB Types- Quad-Inverter," Texas Instruments, Datasheet, schs054.pdf, 1998. Source: www.ti.com.
4. "SN5400 Quadruple 2-Input Positive NAND-Gates," Texas Instruments, Datasheet, sdls025.pdf, March 1988, Source: www.ti.com.
5. Koga, R., Crain, W.R., Hansel, S.J., Crawford, K.B., Pinkerton, S.D., Peozin, S.H., Moses, S.C., and Maher, M., "Ion Induced Charge Collection and SEU Sensitivity of Emitter Coupled Logic (ECL) Devices," *IEEE Trans. on Nuclear Science*, 42(6), 1823–1828, 1995.
6. High-performance *Impact-X*™ Programmable Array Logic Circuits, TIBPAL22V10-7C, TI, 1995. Product datasheet.
7. "Dual Full-Bridge Driver L298," SGS Thomson Microelectronics Datasheet. Source: www.st.com.

24

Architecture

Daniel A. Connors
University of Colorado at Boulder

Wen-Mei W. Hwu
University of Illinois

The microprocessor industry is divided into the computer and embedded sectors. Both computer and embedded microprocessors share aspects of computer system design, instruction set architecture, organization, and hardware. The term **architecture** is used to describe these fundamental aspects, and more directly refers to the hardware components in a computer system and the flow of data and control information among them. In this section, various types of microprocessors will be described, fundamental architecture mechanisms relevant in the operation of all microprocessors will be presented, and microprocessor industry trends discussed.

24.1 Types of Microprocessors

Computer microprocessors are designed for use as the central processing units (CPU) of computer systems such as personal computers, workstations, servers, and supercomputers. Although microprocessors started as humble programmable controllers in the early 1970s, virtually all computer systems built after 1990 use microprocessors as their central processing units. The dominating architecture in the computer microprocessor domain today is the Intel 32-bit Architecture, also known as IA-32 or X86. Other high profile architectures in the computer microprocessor domain include Intel Itanium Processor Family (IPF), HP PA-RISC, SUN Microsystems SPARC, and IBM/Motorola PowerPC.

Embedded microprocessors are increasingly used in consumer and telecommunications products to satisfy the demands for quality and functionality. Major product areas that require embedded

microprocessors include digital TV, DVD players, digital camera, network switches, high speed modems, digital cellular phones, video games, laser printers, and automobiles. Future improvements in energy consumption, fabrication cost, and performance will further enable new applications such as intelligent hearing aid. Many experts expect that embedded microprocessors will form the fastest growing sector of the semiconductor business in the next decade [1].

Embedded microprocessors have been categorized into DSP processors and Embedded CPUs owing to historic reasons. DSP processors have been designed and marketed as special purpose devices that are mostly programmed by hand to perform digital signal processing computations. A recent trend in the DSP market is to use compilers to alleviate the need for tedious hand coding in DSP development. Leading DSP microprocessor vendors include Texas Instruments, Lucent Technologies, Motorola, and ST.

Embedded CPUs are often used to execute operating system, networking, and user interface code in a consumer and telecommunications products. They have been traditionally derived from out-of-date computer microprocessors. Embedded CPUs often reuse the compiler and related software support developed for their computer cousins. Recycling the microprocessor design and compiler software minimizes engineering cost. Major vendors of embedded CPUs include IBM, Motorola, ARM, and MIPS.

An important recent trend in the embedded microprocessor market is toward integrating an embedded CPU, a DSP processor, and application-specific logic to form a single-chip solution. This approach is enabled by the ever-increasing transistor density achieved by the semiconductor fabrication technology. The major benefit is reduced system cost and energy consumption. In these designs, embedded CPU and DSP processor vendors no longer market microprocessor chips. Rather, they make their designs available for licensing by solution developers such as a cell phone vendor. The embedded CPU and the DSP processors thus incorporated into these single-chip solutions are called embedded CPU cores and DSP processor cores. For example, MIPS customized its embedded CPU core for use in Nitendo64. IBM, ARM, ST, NEC, and Hitachi offer similar products and services. Because of an increasing need to perform DSP computation in consumer and telecommunication products, an increasing number of embedded CPUs have extensions to enable more effective DSP computation.

There are several ways in which the needs of embedded computing differ from those of more traditional general-purpose computing systems. Constraints on the code size, weight, and power consumption place stringent requirements on embedded microprocessors and the software they execute. Also, constraints rooted in real-time requirements are often a significant consideration in many embedded systems. Furthermore, cost is a severe constraint in designing and manufacturing embedded processors.

In spite of the different constraints and product markets, both computer and embedded microprocessors share many main elements in their design. These main elements will be described. Additionally, over the past decade, substantial research has gone into the design of microprocessors embodying parallelism at the instruction level, as well as aggressive compiler optimization and analysis techniques for harnessing this opportunity. Exploitation of parallelism is a very effective approach to reducing the power consumption under given performance requirements. Much of this effort has since been validated through the proliferation of high-performance general purpose microprocessors, such as the Intel Itanium II processor, on the basis of these technologies. Nevertheless, growing demand for high performance in embedded computing systems is creating new opportunities to leverage these techniques in application-specific domains. The research of instruction-level parallelism (ILP) has developed distinct architecture methodologies referred to as very long instruction word (VLIW) and explicitly parallel instruction computing (EPIC) technology. Overall, these techniques represent fundamental, substantial changes in computer architecture.

24.2 Major Components of a Microprocessor

The main hardware of a microprocessor system can be divided into sections according to their functionalities. A popular approach is to divide a system into four subsystems: the central processor, the memory subsystem, the input/output (I/O) subsystem, and the system interconnect. Figure 24.1 shows

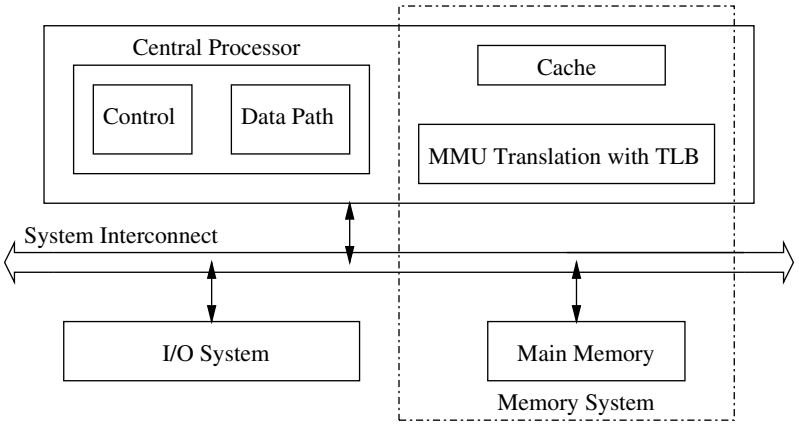


FIGURE 24.1 Subsystems of a computer system.

the connection between these subsystems. The main components and characteristics of these subsystems will be described.

24.2.1 Central Processor

A modern microprocessor system's central processor is typically further divided into control unit and data path.

24.2.1.1 Control Unit

The control unit of a microprocessor generates the control signals to orchestrate the activities in the data path. There are two major types of communication lines between the control unit and the data path: the control lines and the condition lines. The control lines deliver the control signals from the control unit to the data path. Different signal values on these lines trigger different actions in the data path. The condition lines carry the status of the execution from data path to the control unit. These lines are needed to test conditions involving the registers in the data path in order to make future control decisions. Note that the decision is made in the control unit but the registers are in the data path. Therefore, the conditions regarding the register contents are formed in the data path and then communicated to the control unit for decision making. A control unit can be implemented with hardwiring, microprogramming, or a combination of both.

A hardwired control unit is designed as a finite state machine that is realized with registers, combinational logic, and wires. Once constructed, the design can be changed only through physically rewiring the unit. Therefore, the resulting circuits are called hardwired control units. Design optimizations are typically performed on the combinational logic to minimize component count and maximize operation speed, which makes the resulting circuits exhibit little structure. The lack of structure makes it very difficult to design and debug complicated control units with this technique. Therefore, hardwiring is normally used when the control unit is relatively simple.

Most of the design difficulties in the hardwired control units are due to the effort of optimizing the combinational logic. One popular alternative to hardwired control unit design is to use either read only memory (ROM) or random access memory (RAM) to implement the combinational logic. A control unit whose combinational logic is realized by the use of ROM or RAM is called a *microprogrammed control unit*. The memory used is called *control memory* (CM). The practice of realizing the combinational circuit in a control unit with ROM/RAM is called *microprogramming*. The concept of microprogramming was first introduced by Maurice V. Wilkes.

The idea of using a memory to implement a combinational circuit can be illustrated with a simple example. Assume that we are to implement a logic function with three input variables, as described in the

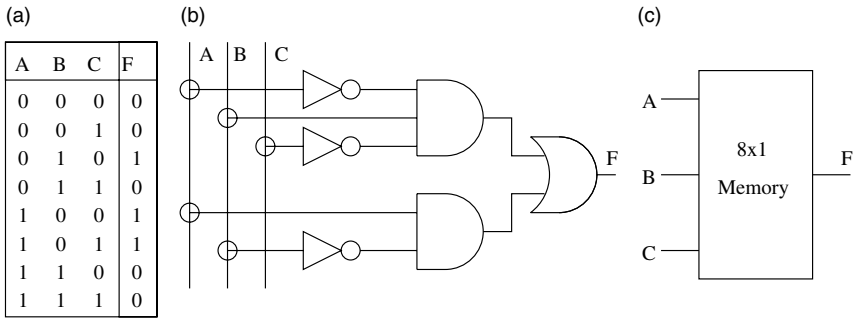


FIGURE 24.2 Using memory to simplify combinational logic design (a) truth table, (b) Karaugh map approach, and (c) memory approach.

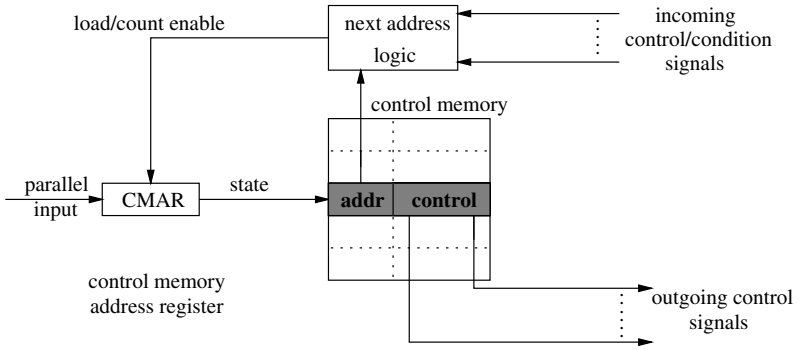


FIGURE 24.3 Basic model of microprogrammed control units.

truth table illustrated in Figure 24.2a. A common way to realize this function is to use Karnaugh maps to derive highly optimized logic and wiring. The result is shown in Figure 24.2b. The same function can also be realized using a memory with eight 1-bit locations to retain the eight possible combinations of the three input variables. Location *i* contains an *F* value corresponding to the *ith* input combination. For example, location 3 contains the *F* value (0) for the input combination 011. The three input variables are then connected to the address input of the memory to complete the design (Figure 24.2c). In essence, the memory contains the entire truth table. Considering the decoding logic and storage cells involved in a 8x1 memory, it is obvious that the memory approach uses a lot more hardware components than the Karnaugh map approach. However, the design is much simpler in the memory approach.

Figure 24.3 illustrates the general model of a microprogrammed control unit. Each control memory location consists of an address field and some control fields. The address field plus the next address logic implements the combinational logic for generating the next state value. The control fields implement the combinational logic for generating the control signals. The state register/counter is referred to as *control memory address register* (CMAR) for an obvious reason: the contents of the register are used as the address input to the control memory. An important insight is that the CMAR stores the state of the control unit.

In modern microprocessor designs, hardwiring and microprogramming are often used in conjunction with each other. In such a design, hardwiring is used to handle common, simpler cases of instruction execution whereas microprogramming is used to handle complex cases such as string move instructions in the IA-32 microprocessors.

24.2.1.2 Data Path

The data path of a microprocessor contains the main arithmetic and logic execution units required to execute instructions. Designing the data path involves analyzing the function(s) to be performed, then

specifying a set of hardware registers to hold the computation state, and designing computation steps to transform the contents of these registers into the final result. In general, the functions to be performed are divided into steps, each of which can be done with a reasonable amount of logic in one clock cycle. Each step brings the contents of the registers closer to the final result. The data path must be equipped with sufficient amount of hardware to allow these computation steps in one clock cycle. The data path of a typical microprocessor contains integer and floating-point register files, ten ormore functional units for computation and memory access, and pipeline registers. The concept of pipelining and pipeline registers will be introduced later in this chapter.

24.2.2 Memory Subsystem

The memory system serves as a repository of information in a microprocessor system. The processing unit retrieves information stored in memory, operates on the information, and returns new information back to memory. The memory system is constructed from basic semiconductor DRAM units called modules or banks.

There are several properties of memory, including speed, capacity, and cost that play an important role in the overall system performance. The speed of a memory system is the key performance parameter in the design of the microprocessor system. The latency (L) of the memory is defined as the time delay from when the processor first requests data from memory until the processor receives the data. Bandwidth is defined as the rate at which information can be transferred to and from the memory system. Memory bandwidth and latency are related to the number of outstanding requests (R) that the memory system can service:

$$\text{BW} = \frac{R}{L} \tag{24.1}$$

Bandwidth plays an important role in keeping the processor busy with work. However, technology tradeoffs to optimize latency and improve bandwidth often conflict with the need to increase the capacity and reduce the cost of the memory system.

24.2.2.1 Cache Memory

Cache memory, or simply cache, is a fast memory constructed using semiconductor SRAM. In modern computer systems, there is usually a hierarchy of cache memories. The top level (Level-1 or L1) cache is closest to the processor and the lowest level is closest to the main memory. Each lower level cache is about 3–5 times slower than its predecessor level. The purpose of a cache hierarchy is to satisfy most of the processor memory accesses in one or a small number of clock cycles. The L1 cache is often split into an instruction cache and a data cache to allow the processor to perform simultaneous accesses for instructions and data. Cache memories were first used in the IBM mainframe computers in the 1960s. Since 1985, cache memories have become a standard feature for virtually all microprocessors.

Cache memories exploit the principle of locality of reference. This principle dictates that some memory locations are referenced more frequently than others on the basis of two program properties. *Spatial locality* is the property that an access to a memory location increases the probability that the nearby memory location will also be accessed. Spatial locality is predominantly based on sequential access to program code and structured data. In order to exploit spatial locality, memory data are placed into cache inmultiple-word units called *cache lines. Temporal locality* is the property that access to amemory location greatly increases the probability that the same location will be accessed in the near future. Together, the two properties assure that most memory references will be satisfied by the cache memory.

Set associativity refers to the flexibility in placing a memory data into a cache memory. If a cache design allows each memory data to be placed into any of N cache locations, it is referred to as an N-way set-associative cache. A 1-way set-associative cache is also called a direct-mapped cache, as is shown in Figure 24.4a. A 2-way set-associative cache, as shown in Figure 24.4b, allows a memory data to reside in

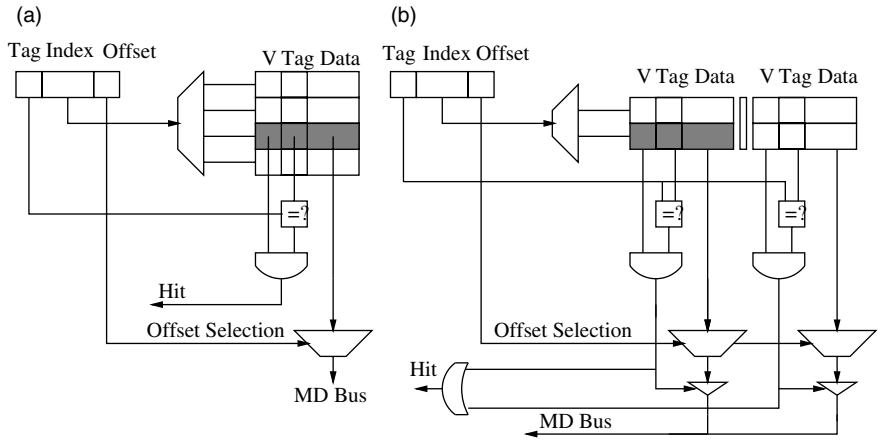


FIGURE 24.4 Cache memory (a) direct mapped design and (b) two-way set associative design.

one of two locations in the cache. Finally, an extreme design called fully associative cache allows amemory data to be placed anywhere in the cache.

Cache misses occur when the data requested does not reside in any of the possible cache locations. Misses in caches can be classified into three categories: conflict, compulsory, and capacity. Conflict misses are misses that would not occur for fully associative caches with least recently used (LRU) replacement. Compulsory misses are misses required in cache memories for initially referencing a memory location. Capacity misses occur when the cache size is not sufficient to keep data in the cache between references. Complete cache miss definitions are provided in Reference 2.

The latency in cache memories is not fixed and depends on the delay and frequency of cache misses. A performance metric that accounts for the penalty of cache misses is *effective latency*. Effective latency depends on the two possible latencies: hit latency (L_{HIT}), the latency experienced for accessing data residing in the cache; and miss latency (L_{MISS}), the latency experienced when accessing data not residing in the cache. Effective latency also depends on the *hit rate* (H), the percentage ofmemory accesses that are hits in the cache, and *miss rate* (M or 1 - H), the percentage of memory accesses that miss in the cache. Effective latency in a cache system is calculated as

$$L_{\text{effective}} = L_{\text{HIT}} \times H + L_{\text{MISS}} \times (1 - H) \tag{24.2}$$

In addition to the base cache design and size issues, there are several other dimensions of cache design that affect the overall cache performance and miss rate in a system. The main memory update method dictates when the main memory will be updated by store operations. In a *write-through* cache, each write is immediately reflected to the main memory. In a *write-back* cache, the writes are reflected to main memory only when the respective cached data is purged from cache to make room for other memory data. Cache allocation designates whether cache locations are allocated on writes and/or reads. Finally, cache replacement algorithms for associative structures can be designed in various ways to extract additional cache performance. These include LRU, least frequently used (LFU), random, and first in, first out (FIFO). These cache management strategies attempt to exploit the properties of locality. Traditionally, when caches service misses they would *block* all new requests. However, *nonblocking* cache can be designed to service multiple miss requests simultaneously, thus alleviating delay in accessing memory data.

In addition to the multiple levels of cache hierarchy, additional memory buffers can be used to improve cache performance. Two such buffers are a streaming/prefetch buffer and a victim cache [3]. Figure 24.5 illustrates the relation of the streaming buffer and victim cache to the primary cache of a memory system. A streaming buffer is used as a prefetching mechanism for cache misses. When a cache miss occurs, the streaming buffer begins prefetching successive lines starting at the miss target. A victim cache is typically

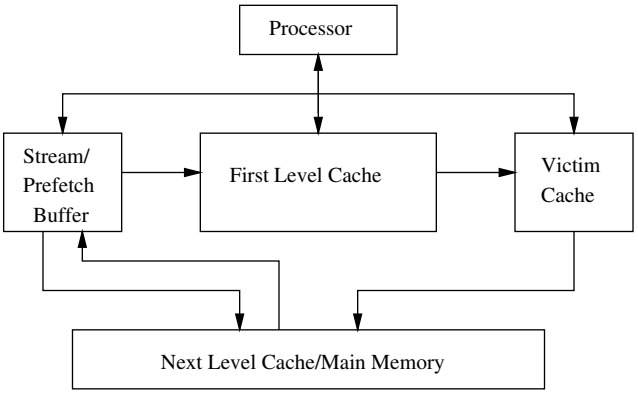


FIGURE 24.5 Advanced cache memory system.

a small fully associative cache loaded only with cache lines that are removed from the primary cache. In the case of a miss in the primary cache, the victim cache may hold the missed data. The use of a victim cache can improve performance by reducing the number of conflict misses. Figure 24.5 illustrates how cache accesses are processed through the streaming buffer into the primary cache on cache requests and from the primary cache through the victim cache to the secondary level of memory on cache misses.

Overall, cache memory is constructed to hold the most important portions of memory. Techniques using either hardware or software can be used to select which portions of main memory to store in cache. However, cache performance is strongly influenced by program behavior and numerous hardware design alternatives.

24.2.2.2 Virtual Memory

Cache memory illustrated the principle that the memory address of data can be separate froma particular storage location. Similar address abstractions exist in the two-level memory hierarchy of main memory and disk storage. An address generated by a program is called a *virtual address*, which needs to be translated into a *physical address* or location in main memory. Virtual memory management is a mechanism that provides the programmers with a simple, uniform method to access both main and secondary memories. With virtual memory management, the programmers are given a virtual space to hold all the instructions and data. The virtual space is organized as a linear array of locations. Each location has an address for convenient access. Instructions and data have to be stored somewhere in the real system; these virtual space locations must correspond to some physical locations in the main and secondary memory. Virtual memory management assigns (or maps) the virtual space locations into the main and secondary memory locations. The mapping of virtual space locations to the main and secondary memory is managed by the virtual memory management mechanism. The programmers are not concerned with the mapping.

The most popular memory management scheme today is demand paging virtual memory management, where each virtual space is divided into pages indexed by the page number (PN). Each page consists of several consecutive locations in the virtual space indexed by the page index (PI). The number of locations in each page is an important system design parameter called page size. Page size is usually defined as a power of two so that the virtual space can be divided into an integer number of pages. Pages are the basic units of virtual memory management. If any location in a page is assigned to the main memory, the other locations in that page are also assigned to main memory. This reduces the size of the mapping information.

The part of the secondary memory to accommodate pages of the virtual space is called the swap space. Both the main memory and the swap space are divided into page frames. Each page frame can host a page of the virtual space. The mapping record in the virtual memory management keeps track of the association between pages and page frames.

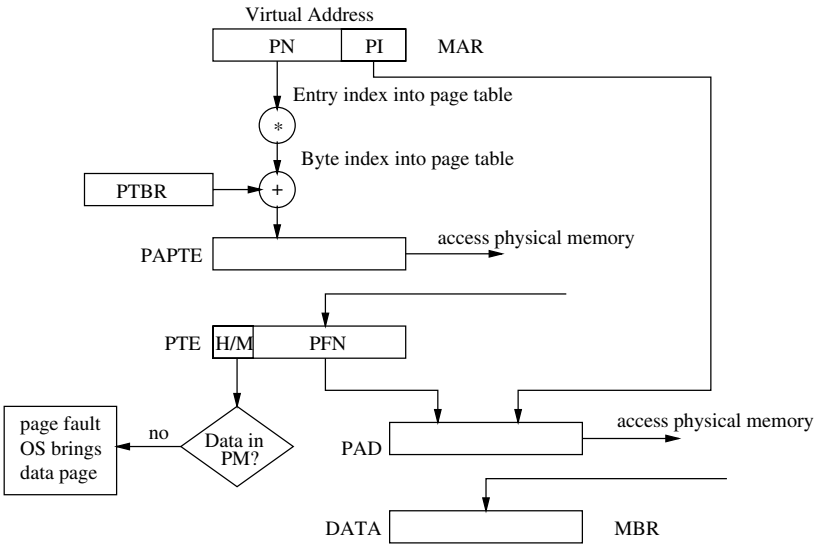


FIGURE 24.6 A simple virtual memory access algorithm.

When a virtual space location is requested, the virtual memory management looks up the mapping record. If the mapping record shows that the page containing requested virtual space location is in main memory, the management performs the access without any further complication. Otherwise a secondary memory access has to be performed. Accessing the secondary memory is a complicated task and is usually performed as an operating system service.When a page is mapped into the secondary memory, the virtual memory management has to request a service in the operating system to transfer the requested page into the main memory, update its mapping record, and then perform the access. The operating system service thus performed is called the page fault handler.

The core process of virtual memory management is a memory access algorithm. A one-level memory access algorithm is illustrated in Figure 24.6. At the start of the memory access, the algorithm receives a virtual address in a memory address register (MAR), looks up the mapping record, requests an operating system service to transfer the required page if necessary, and performs the main memory access. The mapping is recorded in a data structure called the page table located in main memory at a designated location marked by the page table base register (PTBR).

Each page is mapped by a page table entry (PTE) that occupies a fixed number of bytes in the page table. Thus, one can easily multiply the page number by the size of each PTE to form a byte index into the page table. The byte index of the PTE is then added to the PTBR to form the physical address (PAPTE) of the required PTE. Each PTE includes two fields: a hit/miss bit and a page frame number. If the hit/miss (H/M) bit is set (hit), the corresponding page is in main memory. In this case, the page frame hosting the requested page is pointed to by the page frame number (PFN). The final physical address (PAD) of the requested data is then formed by concatenating the PFN and PI. The data is returned and placed in the memory buffer register (MBR) and the processor is informed of the completed memory access. Otherwise (miss), a secondary memory access has to be performed. In this case, the page frame number should be ignored. The fault handler has to be invoked to access the secondary memory. The hardware component that performs the address translation part of the memory access algorithm is called the memory management unit (MMU).

The complexity of the algorithm depends on the mapping structure. A very simple mapping structure is used in this section to focus on the basic principles of the memory access algorithms. However, more complex two-level schemes are often used when the page table becomes a large portion of the main memory. There are also requirements for such designs in a multiprogramming system, where there are

multiple processes active at the same time. Each processor has its own virtual space and therefore its own page table. As a result, these systems need to keep multiple page tables at the same time. It usually takes too much main memory to accommodate all the active page tables. Again, the natural solution to this problem is to provide additional levels of mapping where a second-level page table is used to map the main page table. In such designs, only the second-level page table is stored in a reserved region of main memory, while the first page table is mapped just like the code and data between the secondary storage and the main memory.

24.2.2.3 Translation Lookaside Buffer

Hardware support for virtual memory management generally includes a translation lookaside buffer (TLB) to accelerate the translation of virtual addresses into physical addresses. A TLB is a cache structure that contains the frequently used page table entries for address translation.With a TLB, address translation can be performed in a single clock cycle when TLB contains the required page table entries (TLB hit). The full address translation algorithm is performed only when the required page table entries are missing from the TLB (TLB miss).

Complexities arise when a system includes both virtual memory management and cache memory. The major issue is whether address translation is done before accessing the cache memory. In *virtual* cache systems, the virtual address directly accesses cache. In a *physical* cache system, the virtual address is translated into a physical address before cache access. Figure 24.7 illustrates the both the *virtual* and *physical* cache approaches.

A physical cache system typically overlaps the cache memory access and the access to the TLB. The overlap is possible when the virtual memory page size is larger than the cache capacity divided by the degree of cache associativity. Essentially, since the virtual page index is the same as the physical address index, no translation for the lower indexes of the virtual address is necessary. These lower index bits can be used to access the cache storage while the PN bits go through the TLB. The PFN bits that come out of the TLB are compared with the tag bits of the cache storage output, as shown in Figure 24.6, to determine the hit/miss status of cache memory access. Thus, the cache storage can be accessed in parallel with the TLB.

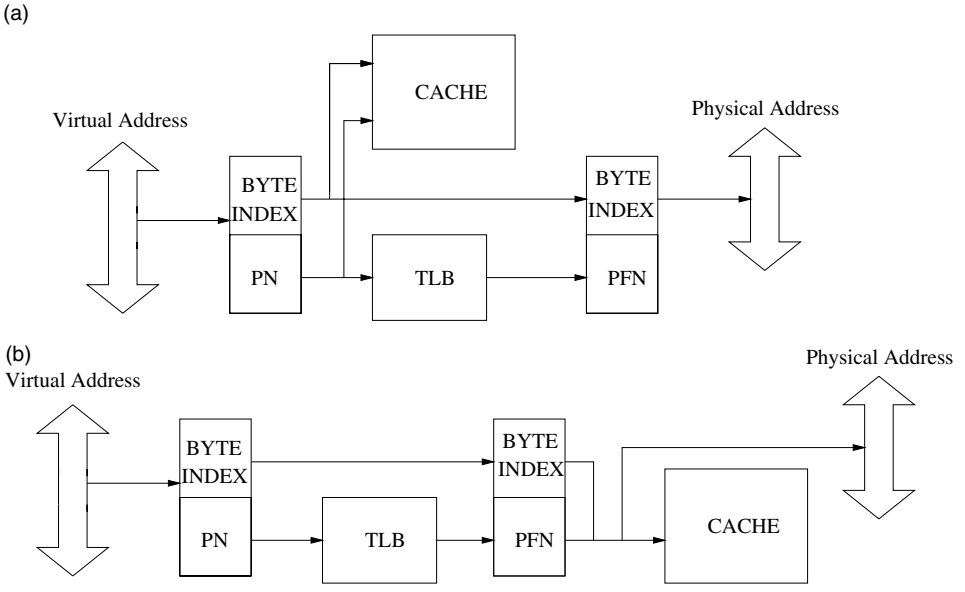


FIGURE 24.7 Translation lookaside buffer (TLB) architectures (a) virtual cache and (b) physical cache.

Virtual caches have their pros and cons. Typically, with no TLB logic between the processor and the cache, access to cache can be achieved at lower cost in virtual cache systems. This is particularly true in multiaccess-per-cycle cache systems where a multiported TLB is needed for a physical cache design. However, the virtual cache alternative introduces virtual memory consistency problems. The same virtual address from two different processes mean different physical memory locations. Solutions to this form of aliasing are to attach process identifier to the virtual address or to flush cache contents on context switches. Another potential alias problem is that different virtual addresses of the same process may be mapped into the same physical address. In general, there is no easy solution to this second form of aliasing; typical solutions involve reverse translation of physical addresses to virtual addresses.

Physical cache designs are not always limited by the delay of the TLB and cache access. In general, there are two solutions to allow large physical cache design. The first solution, employed by companies with past commitments to page size, is to increase the set associativity of cache. This allows the lower index portion of the address to be used immediately by the cache in parallel with virtual address translation. However, large set associativity is very difficult to implement in a cost-effective manner. The second solution, employed by companies without past commitment, is to use a larger page size. The cache can be accessed in parallel with the TLB access similar to the other solution. In this solution there are fewer address bits that are translated through the TLB, potentially reducing the overall delay. With larger page sizes, virtual caches do not have advantage over physical caches in terms of access time. With the advance of semiconductor fabrication processes, in particular the increasing levels of metals, it has become increasingly inexpensive to have highly set-associative caches and multiported TLB. As a result, physical caches have become much more favored solutions today.

24.2.3 Input/Output Subsystem

The input/output (I/O) subsystem transfers data between the internal components (CPU and main memory) and the external devices (disks, networks, printers, keyboards, pointing devices, and scanners).

24.2.3.1 Peripheral Controllers

The CPU usually controls the I/O subsystem by reading from and writing into the I/O (control) registers. There are two popular approaches for allowing the CPU to access these I/O registers: I/O instructions and memory-mapped I/O. In an I/O instruction approach, special instructions are added to the instruction set to access I/O status flags, control registers, and data buffer registers. In a memory-mapped I/O approach, the control registers, the status flags, and the data buffer registers are mapped as physical memory locations.

Due to the increasing availability of chip area and pins, microprocessors are increasingly including peripheral controllers on chip. This trend is especially clear for embedded microprocessors.

24.2.3.2 Direct Memory Access Controller

A direct memory access (DMA) controller is a peripheral controller that can directly drive the address lines of the system bus. The data is directly moved from the I/O data buffer registers to the main memory, rather than from the data buffer registers to a CPU register, then from CPU register to main memory. DMA controllers greatly increase the achieved transfer bandwidth of I/O operations and have become a standard feature of microprocessor systems today.

24.2.4 System Interconnect

System interconnect is the facility that allows the components within a computer system to communicate with each other. There are numerous logical organizations of these system interconnect facilities.

Dedicated links or point-to-point connections enable dedicated communication between components. There are different system interconnect configurations based on the connectivity of the system components. A complete connection configuration, requiring $N \cdot (N - 1)/2$ links, is created when there is one link between every possible pair of components. A *hypercube* configuration assigns a unique n-tuple {1,0} as the coordinate of each component and constructs a link between components whose coordinates

differ only in one dimension, requiring $N \cdot \log N$ links. A *mesh* connection arranges the system components into a N-dimensional array and has connections between immediate neighbors, requiring $2 \cdot N$ links.

Switching networks are a group of switches that determine the existence of communication links among components. A cross-bar network is considered the most general form of switching network and uses a $N \times M$ two-dimensional array of switches to provide an arbitrary connection between N components switching network is the multistage network that employs multiple stages of shuffle networks to provide on one side and M components on another side using $N \cdot M$ switches and $N + M$ links. Another a permutation connection pattern between N components on each side by using $N \cdot \log N$ switches and $N \cdot \log N$ links.

Shared buses are single links that connect all components to all other components and are the most popular connection structures. The sharing of buses among the components of a system requires several aspects of bus control. First, there is a distinction between bus masters, the units controlling bus transfers (CPU, DMA), and bus slaves, the other units (memory, programmed I/O interface).

Bus interfacing and bus addressing are the means to connect and disconnect units on the bus. Bus arbitration is the process of granting the bus resource to one of the masters. Arbitration typically uses a selection scheme on the basis of some type of priority assignment. Fixed-priority arbitration gives every master a fixed priority, and dynamic-priority arbitration such as round-robin ensures every master become the most favorable at one point in time.

Bus timing refers to the method of communication among the system units and can be classified as either synchronous or asynchronous. Synchronous bus timing uses a shared clock that defines the time other bus signals change and stabilize. Clock sharing by all units allows the bus to be monitored at agreed time intervals and action taken accordingly. However, the synchronous system bus must operate at the speed of the slowest component. Asynchronous bus timing allows units to use different clocks, but the lack of a shared clock makes it necessary to use extra signals to determine the validity of bus signals. The use of these signals to determine the validity of bus signals is called hand-shaking protocols, which typically reduce the achievable transfer bandwidth via the bus.

24.3 Performance Enhancing Hardware Techniques

24.3.1 Pipelining

In the 1970s, only supercomputers and mainframe computers were pipelined. Today, virtually all microprocessors are pipelined. In fact, pipelining has been a major reason why microprocessors today outperform supercomputer CPU's built less than 10 years ago. Pipelining is a technique to coordinate parallel processing of operations [3]. This technique has been used in assembly lines of major industries for more than a century. The idea is to have a line of workers specializing in different pieces of work required to finish a product. A conveying belt carries products through the line of workers. Each worker performs a small piece of work on each product. Each product is finished after it is processed by all the workers in the assembly line.

The obvious advantage of pipelining is to allow one worker to immediately start working on a new product after finishing the work on a current product. The same methodology is applied to instruction processing in microprocessors. Figure 24.8a shows an example five-stage pipeline dividing instruction execution into fetch (F), decode (D), execute (E), memory (M), and write-back (W) operations, each requiring various stage-specific logic. Between each stage is a stage register (SR), or pipeline register, used to hold the instruction information necessary to control the instruction. A very basic principle of pipelining is that the work performed by each stage must take about the same amount of time. Otherwise, the efficiency will be significantly reduced because one stage becomes a bottleneck of the entire pipeline. That is, the time duration of the slowest pipeline stage determines the overall clock frequency of the processor. Because of this constraint and the often-time slower memory speeds, some of the principle five stages are often divided into smaller stages. For instance, the memory stage may be divided into three

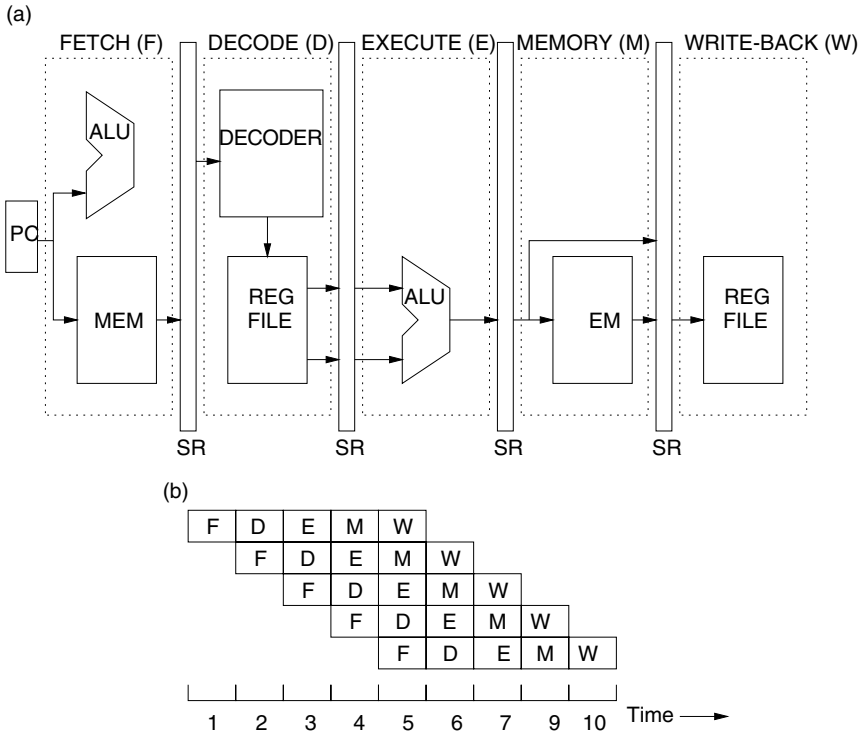


FIGURE 24.8 Pipeline architecture (a) machine and (b) overlapping instructions.

stages, allowing memory accesses to be pipelined and the overall processor clock speed to be a fraction of the memory access latency.

The time required to finish N instructions in a pipeline with K stages can be calculated. Assume a cycle time of T for the overall instruction completion, and an equal T/K processing delay at each stage. With a pipeline scheme, the first instruction completes the pipeline after T, and there will be a new instruction out of the pipeline per stage delay T/K. Therefore, the delays of executing N instructions without (1) and with (2) pipelining respectively are

$$T^{\times}(N) \tag{24.3}$$

$$T + (T/k)^{\times}(N-1) \tag{24.4}$$

There is an initial delay in the pipeline execution model before each stage has operations to execute. The initial delay is usually called *pipeline startup delay*, and is equal to total execution time of one instruction (T). The speedup of a pipelined machine relative to a nonpipelined machine is calculated as

$$\frac{T \times N}{T + (T/k) \times (N-1)} \tag{24.5}$$

When N is much larger than the number of pipe stages k, the ideal speedup approaches k. This is an intuitive result since there are k parts of the machine working in parallel, allowing the execution to go about k times faster in ideal conditions.

The overlap of sequential instructions in a processor pipeline is shown in Figure 24.8b. The processing of each instruction is shown as a five-stage process that goes from left to right; each stage is marked by the initial of its name: F(etch), D(ecode), E(xecute), M(emory), W(riteback). A new instruction is initiated in

every clock cycle. The instruction pipeline becomes full after the pipeline delay of $k = 5$ cycles. Although the pipeline configuration executes operations in each stage of the processor, two important mechanisms are constructed to ensure correct functional operation between dependent instructions in the presence of data hazards. Data hazards occur when instructions in the pipeline generate results that are necessary for later instructions that are already started in the pipeline. In the pipeline configuration of Figure 24.8a, register operands are initially retrieved during the decode stage. However, the execute and memory stage can define register operands and contain the correct current value but are not able to update the register file until the later write-back execution stage. Forwarding (or register bypassing) is the action of retrieving the correct operand value for an executing instruction between the initial register file access and any pending instruction's register file updates. Interlocking is the action of stalling an operation in the pipeline when conditions cause necessary register operand results to be delayed. It is necessary to stall early stages of the machine so that the correct results are used, and the machine does not proceed with incorrect values for source operands. The primary causes of delay in pipeline execution are initiated due to instruction fetch delay and memory latency.

24.3.2 Branch Handling

Branch instructions pose serious problems for pipelined processors by causing hardware to fetch and execute incorrect instructions until the branch instructions are completed. Executing incorrect instructions can result in severe performance degradation through the introduction of wasted cycles into the execution process.

There are several methods for dealing with pipeline stalls caused by branch instructions. The simplest performance scheme handles branches by marking branch instructions as either *taken* or *not taken*. The compiler marks each branch by using a special branch opcode for each case. The designation allows the pipeline to fetch subsequent instructions according to the compiler's choice of opcode. However, the fetched instruction may need to be discarded and the instruction fetch restarted when the branch outcome is inconsistent with the compiler's designation.

Delayed branching is a scheme that treats the set of sequential instructions following a branch as delay slots. The delay-slot instructions are executed whether or not the branch instruction is taken. Limitations on delayed branches are caused by the compiler and program characteristics being unable to identify numerous instructions that execute independent of the branch direction. Improvements have been introduced to provide *nullifying* branches, which include a predicted direction for the branch.When the prediction is incorrect, the delay-slot instructions are nullified.

A more modern approach to reducing branch penalties uses hardware to dynamically predict the outcome of a branch. Branch prediction strategies reduce overall branch penalties by allowing the hardware to continue processing instructions along the predicted control path, thus eliminating wasted cycles. Efficient execution can be maintained while branch targets are correctly predicted. However, performance penalty is incurred when a branch is mispredicted. Branch target buffer is a cache structure that is accessed in parallel with the instruction fetch. It records the past history of branch instructions so that a prediction can be made while the branch is fetched again. The prediction method adapts the branch prediction to the run-time program behavior, generating a high prediction accuracy. The target address of each branch is also saved in the buffer so that the target instruction can be fetched immediately if a branch is predicted taken.

Several methodologies of branch target prediction have been constructed [4]. Figure 24.9 illustrates the relation of several general branch prediction schemes to the processor pipeline. The table in Figure 24.9a retains history information for each branch. The history includes the previous branch directions for making predictions on future branch directions. The simplest history is last taken, which uses 1 bit to recall whether the branch condition was taken or not taken during its most recent execution. A more effective branch predictor, as shown in Figure 24.9b, uses a 2-bit saturating state history counter to determine the future branch outcome. Two bits rather than one bit allow each branch to be tagged as strongly or weakly taken or not taken. Every correct prediction reinforces the prediction, while an incorrect

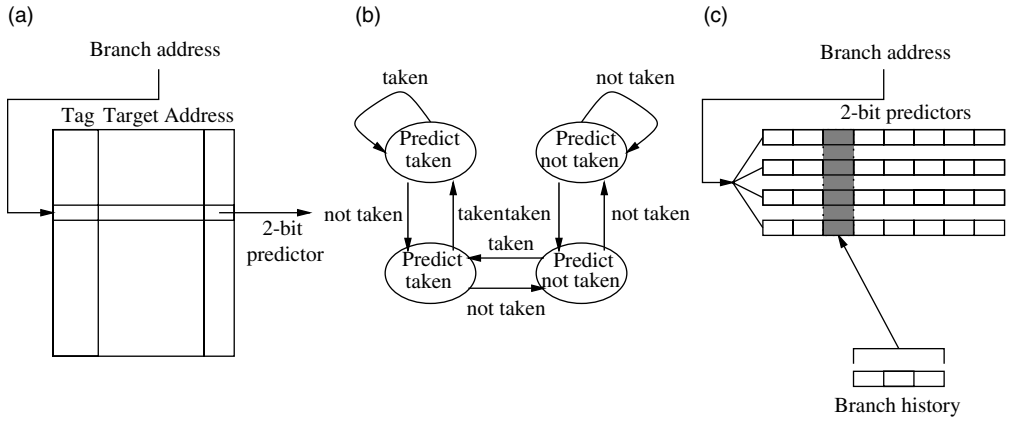


FIGURE 24.9 Branch prediction.

prediction weakens it. It takes two consecutive mispredictions to reverse the direction (whether taken or not taken) of the prediction.

Recently, more complex two-level adaptive branch prediction schemes have been built, which use two levels of branch history to make predictions [5]. An example of such predictor is shown in Figure 24.9c. The first level is the branch outcome history of the last branches encountered. The second level is the branch behavior for the last occurrences of a specific pattern of branch histories, commonly maintained as 2-bit saturating counters. There are alternative ways of constructing both levels of adaptive branch prediction schemes, the mechanisms can contain information that is either based on individual branches, groups (set-based), or all (global). Individual information contains the branch history for each branch instruction. Set-based information groups branch according to their instruction address, thereby forming sets of branch history. Global information uses a global history containing all branch outcomes. The second level containing branch behaviors can also be constructed using any of the three types. In general, the first-level branch history pattern is used as an index into the second-level branch history.

24.3.3 Dynamic Instruction Execution

A major limitation of pipelining techniques is the use of in-order instruction execution. When an instruction in the pipeline stalls, no further instructions are allowed to proceed to ensure proper execution of in-flight instruction. This problem is especially serious for multiple issue machines, where each stall cycle potentially costs work of multiple instructions. However, in many cases, an instruction could execute properly if no data dependence exists between the stalled instruction and the instruction waiting to execute. Dynamic instruction execution, also known as out-of-order execution, is an approach that uses hardware to rearrange the instruction execution to reduce the effect of stalls. The concept of dynamic execution uses hardware to detect dependences in the in-order instruction stream sequence and rearrange the instruction sequence in the presence of detected dependences and stalls.

Today, most modern superscalar microprocessors use dynamic execution to increase the number of instructions executed per cycle. Such microprocessors use basically the same concept: all instructions pass through an issue stage in order, are executed out of order, and are retired in order. There are several functional elements of this common sequence that have developed into computer architecture concepts. The first functional concept is *scoreboarding*. Scoreboarding is a technique for allowing instructions to execute out of order when there are available resources and no data dependences. Scoreboarding originates from the CDC 6600 machine's issue logic, named the scoreboard. The overall goal of scoreboarding is to execute every instruction as early as possible.

A more aggressive approach to dynamic execution is *Tomasulo's algorithm*. This scheme was employed in the IBM 360/91 processor. Although there are many variations on this scheme, the key concept of avoiding write-after-read (WAR) andwrite-after-write (WAW) dependences during dynamic execution is attributed to Tomasulo. In Tomasulo's scheme, the functionality of the scoreboarding is provided by *reservation stations*. Reservation stations buffer the operands of instructions waiting to issue as soon as they become available. The concept is to issue new instructions immediately when all source operands become available instead of accessing such operands through the register file. As such, waiting instructions designate the reservation station entry that will provide their input operands. This action removes WAW dependences caused by successive writes to the same register by forcing instructions to be related by dependences instead of by register specifiers. In general, renaming of register specifiers for pending operands to the reservation station entries is called *register renaming*. Overall, Tomasulo's scheme combines scoreboarding and register renaming. Tomasulo [6] provides complete details of his scheme in the context of a floating point unit design. Patt, et al. first described the extensions required to implement complete modern CPU's with the Tomasulo's Algorithm [7].

24.4 Instruction Set Architecture

There are several elements that characterize an instruction set architecture, word size, instruction encoding, and architecture style.

24.4.1 Word Size

Programs often differ in the size of data they prefer to manipulate. Word processing programs operate on 8-bit or 16-bit data that correspond to characters in text documents. Many applications require 32-bit integer data to avoid frequent overflow in arithmetic calculation. Scientific computations often require 64-bit floating point data to achieve desired accuracy. Operating systems and databases may require 64-bit integer data to represent a very large name space with integers. As a result, the processors are usually designed to access data of a variety of sizes from memory systems. This is a well-known source of complexity in microprocessor design.

The endian convention specifies the numbering of bytes within a memory word. In the little endian convention, the least significant byte in a word is numbered byte 0. The number increases as the positions increase in significance. The DEC VAX and X86 architectures follow the little endian convention. In the big endian convention, the most significant byte in a word is numbered 0. The number decreases as the positions decrease in significance. The IBM 360/370, HP PA-RISC, SUN SPARC, and Motorola 680X0 architectures follow the big endian convention. The endian convention determines how the word is stored in the address space or in a binary file. The difference usually manifests itself when users try to transfer binary files between machines using different endian conventions.

24.4.2 Instruction Encoding

Instruction encoding plays an important role in the code density and performance of microprocessors. Traditionally, the cost of memory capacity was the determining factor in designing either a fixed-length or variable-length instruction set. Fixed-length instruction encoding assigns the same encoding size to all instructions. Fixed-length encoding is generally a product of the increasing advancements in memory capacity.

Variable-length instruction set is the termused to describe the style of instruction encoding that uses different instruction lengths according to addressing modes of operands. Common addressing modes include either register or methods of indexing memory. Figure 24.10 illustrates two potential designs found in modern use of decoding variable-length instructions. The first alternative in Figure 24.10a involves an additional instruction decode stage in the original pipeline design. In this model, the first stage is used to determine instruction lengths and steer the instructions to the second stage where the actual instruction

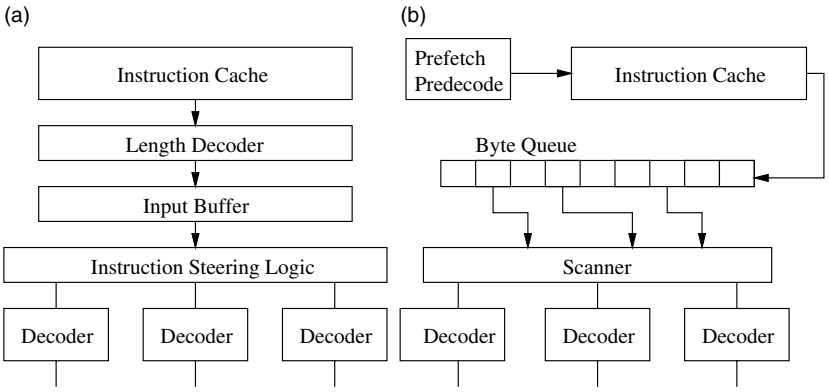


FIGURE 24.10 Variable-sized instruction decoding (a) staging and (b) predecoding.

decoding is performed. The second alternative in Figure 24.10b involves predecoding and marking instruction lengths in the instruction cache. The primary advantage of this scheme is the simplification of the number of decode stages in the pipeline design. However, the method requires a larger instruction cache structure for holding the resolved instruction information. Both design methodologies have been effectively used in decoding X86 variable instructions [8].

24.4.3 Architecture Style

Several instruction set architecture styles have existed over the past three decades of computing. First, complex instruction set computers (CISC) characterized designswith variable instruction formats, numerous memory addressing modes, and large numbers of instruction types. The original CISC philosophy was to create instruction sets that resembled high-level programming languages in an effort to simplify compiler technology. In addition, the design constraint of small memory capacity also led to the development of CISC. Two examples of the CISC model are the Digital Equipment Corporation VAX and Intel X86 architecture families.

Reduced instruction set computers (RISC) gained favor with the philosophy of uniform instruction lengths, load-store instruction sets, limited addressing modes, and reduced number of operation types. RISC concepts allow the microarchitecture design of machines to be more easily pipelined, reducing the processor clock cycle frequency and the overall speed of a machine. The RISC concept resulted from improvements in compiler technology and memory size. The HP PA-RISC, Sun SPARC, IBM Power PC, MIPS, and DEC Alpha machines are examples of RISC architectures.

Architecture models that specify multiple operations to issue in a clock cycle are very long instruction word (VLIW). VLIWs issue a fixed number of operations conveyed as a single long instruction and place the responsibility of creating the parallel instruction packet on the compiler. Early VLIW processor suffered from code expansion due to unfilled operation slots in the long instructions. Examples of VLIW technology are the Multiflow Trace, Cydrome Cydra machines, and TI-C6X. Explicitly parallel instruction computing (EPIC) is similar in concept to VLIW in that both use the compiler to explicitly group instructions for parallel execution. In fact, many of the ideas for EPIC architectures come from previous RISC and VLIW machines. In general, the EPIC concept solves the excessive code expansion and scalability problems associated with VLIW models by providing encoding mechanisms to reduce the need to represent unfilled operation slots in long instructions. Also, the trend of compiler-controlled architecture mechanisms such as predicated execution and speculative execution to be described later in this section are generally considered part of the EPIC-style architecture domain. The Intel IA-64, Philips Trimedia, and Lucent/Motorola StarCore are examples of EPIC machines.

24.5 Instruction Level Parallelism

Explicitly parallel instruction computing processors are equipped with instruction set architecture mechanisms designed to facilitate compiler's effort to arrange for the parallel execution of many operations in each clock cycle. These mechanisms, referred to as instruction level parallelism (ILP) features, are new instruction set architecture concepts for improving microprocessor performance.

24.5.1 Predicated Execution

Branch instructions are recognized as a major impediment to exploiting ILP. Branches force the compiler and hardware to make frequent predictions of branch directions in an attempt to find sufficient parallelism. Branch prediction strategies reduce this problem by allowing the compiler and hardware to continue processing instructions along the predicted control path, thus eliminating these wasted cycles. However, misprediction of these branches can result in severe performance degradation through the introduction of wasted cycles into the instruction stream.

Predicated execution provides an effective means to eliminate branches from an instruction stream. Predicated execution refers to the conditional execution of an instruction based on the value of a boolean source operand, referred to as the predicate of the instruction. This architectural support allows the compiler to use an *if-conversion* algorithm to convert conditional branches into predicate defining instructions, and instructions along alternative paths of each branch into predicated instructions [9]. Predicated instructions are fetched regardless of their predicate operand value. Instructions whose predicate operands are true are executed normally. Conversely, instructions whose predicate operands are false are nullified, and thus are prevented from modifying the processor state. Predicated execution allows the compiler to trade instruction fetch efficiency for the capability to expose ILP to the hardware along multiple execution paths.

Predicated execution offers the opportunity to improve branch handling in microprocessors. Eliminating frequently mispredicted branches may lead to a substantial reduction in branch prediction misses. As a result, the performance penalties associated with mispredicting these branches are removed. Eliminating branches also reduces the need to handle multiple branches per cycle for wide issue processors. Finally, predicated execution provides an efficient interface for the compiler to exposemultiple execution paths to the hardware. Without compiler support, the cost of maintaining multiple execution paths in hardware can be prohibitive.

The essence of predicated execution is the ability to suppress the modification of the processor state on the basis of some execution condition. There must be a way to express this condition and a way to express when the condition should affect execution. Full predication cleanly supports this through a combination of instruction set and microarchitecture extensions. These extensions can be classified as support for suppression of execution and expression of condition. The result of the condition that determines if an instruction should modify state is stored in a set of 1-bit registers. These registers are collectively referred to as the predicate register file. The values in the predicate register file are associated with each instruction in the extended instruction set through the use of an additional source operand. This operand specifies which predicate register will determine whether the operation should modify processor state. If the value in the specified predicate register is 1, or true, the instruction is executed normally; if the value is 0, or false, the instruction is suppressed.

Predicate register values may be set using predicate define instructions, as described in the HPL Playdoh architecture [10]. There is a predicate define instruction for each comparison opcode in the original instruction set. The major difference with conventional comparison instructions is that these predicate defines have up to two destination registers and that their destination registers are predicate registers. The instruction format of a predicate define is shown below.

$$\text{pred\_<}cmp\text{>}\ \text{Pout1}_{<type>},\ \text{Pout2}_{<type>},\ \text{src1},\ \text{src2}\ (\text{P}_{in})$$

This instruction assigns values to *Pout1* and *Pout2* according to a comparison of *src1* and *src2* specified by <*cmp*>. The comparison <*cmp*> can be: equal (eq), not equal (ne), greater than (gt), and so forth. A predicate <*type*> is specified for each destination predicate. Predicate defining instructions are also predicated, as specified by P_{in}.

The predicate <*type*> determines the value written to the destination predicate register given the result of the comparison and of the input predicate, P_{in}. For each combination of comparison result and P_{in}, one of three actions may be performed on the destination predicate. It can write 1, write 0, or leave it unchanged. There are six predicate types that are particularly useful, the unconditional (*U*), *OR*, and *AND* type predicates and their complements. Table 24.1 contains the truth table for these predicate definition types.

Unconditional destination predicate registers are always defined, regardless of the value of P_{in} and the result of the comparison. If the value of P_{in} is 1, the result of the comparison is placed in the predicate register (or its compliment for $\overline{U}$). Otherwise, a 0 is written to the predicate register. Unconditional predicates are utilized for blocks that are executed along only one path in the if-converted code region.

The *OR* type predicates are useful when execution of a block can be enabled along multiple paths, such as logical AND (&&) and OR (||) constructs in C. *OR* type destination predicate registers are set if P_{in} is 1 and the result of the comparison is 1 (0 for $\overline{OR}$), otherwise the destination predicate register is unchanged. Note that *OR* type predicates must be explicitly initialized to 0 before they are defined and used. However, after they are initialized, multiple *OR* type predicate defines may be issued simultaneously and in any order on the same predicate register. This is true since the *OR* type predicate either writes a 1 or leaves the register unchanged, which allows implementation as a wired logical *OR* condition. *AND* type predicates are analogous to the *OR* type predicate. *AND* type destination predicate registers are cleared if P_{in} is 1 and the result of the comparison is 0 (1 for $\overline{AND}$), otherwise the destination predicate register is unchanged.

Figure 24.11 contains a simple example illustrating the concept of predicated execution. Figure 24.11a shows a common programming if-then-else construction. The related control flow representation of that programming code is illustrated in Figure 24.11b. Using if-conversion, the code in Figure 24.11b is then transformed into the code shown in Figure 24.11c. The original conditional branch is translated into *pred_eq* instructions. Predicate register *p1* is set to indicate if the condition *(A = B)* is true, and *p2* is set if the condition is false. The "then" part of the if statement is predicated on *p1* and the "else" part is

TABLE 24.1 Predicate Definition Truth Table

		P_{out}					
P_{in}	Comparison	*U*	$\overline{U}$	*OR*	$\overline{OR}$	*AND*	$\overline{AND}$
0	0	0	0	—	—	—	—
0	1	0	0	—	—	—	—
1	0	0	1	—	1	0	—
1	1	1	0	1	—	—	0

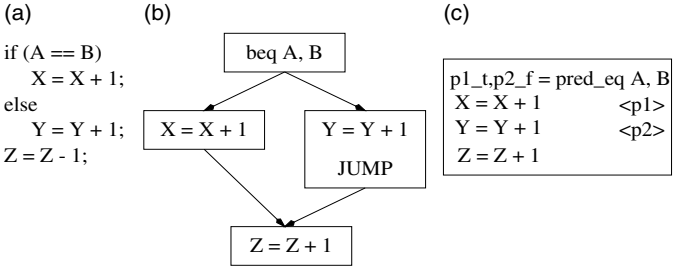


FIGURE 24.11 Instruction sequence (a) program code, (b) traditional execution, and (c) predicated execution.

predicated on *p2*. The *pred_eq* simply decides whether the addition or subtraction instruction is performed and ensures that one of the two parts is not executed.

24.5.2 Speculative Execution

The amount of ILP available within basic blocks, defined as consecutive instruction sequences without branching in or out, is extremely limited in most programs. As such, processors must optimize and schedule instructions across basic block boundaries to achieve higher performance. In addition, future processors must contend with both long latency load operations and long latency cache misses. When load data is needed by subsequent dependent instructions, the processor execution must wait until the cache access is complete.

In these situations, our-of-order processors dynamically perform branch prediction and reorder the instruction stream to execute nondependent instructions. As a result, they have ability of exploiting parallelism between instructions before and after a correctly predicted branch instruction. However, this approach requires complex circuitry at the cost of chip die space as well as additional power consumption. Similar performance gains can be achieved using static compile-time speculation methods without complex, power hungry out-of-order logic. Speculative execution, a technique for executing an instruction before knowing that its execution is required, is an important technique for exploiting ILP in programs. Speculative execution is best known for its use in hiding memory latency. A compiler can utilize speculative code motion to achieve higher performance in several ways. First, in regions of code where insufficient ILP exists to fully utilize the processor resources, useful instructions from other regions may be executed. Second, instructions at the beginning of long dependence chains may be executed early to reduce the computation's critical path. Finally, long latency instructions may be initiated early to overlap their execution with other useful operations. Figure 24.12 illustrates a simple example of code before and after a speculative compile-time transformation is performed to execute a load instruction above a conditional branch.

Figure 24.12a shows how the branch instruction and its implied control flow define a control dependence that restricts the load operation from being scheduled earlier in the code. Cache miss latencies would halt the processor unless out-of-order execution mechanisms were used. However, with speculation support, Figure 24.12b can be used to hide the latency of the load operation.

The solution requires the load to be speculative or nonexcepting. A speculative load will not signal exceptions such as address alignment errors or address space access violations. Essentially, the load remains silent for these exceptions. The additional check instruction in Figure 24.12b enables these signals to be detected when the execution does reach the original location of the load. When the other path of branch's execution is taken, such silent signals are meaningless and can be ignored. Using this mechanism, the load can be placed above all existing control dependences, providing the compiler with the ability to hide load latency. Details of compiler speculation can be found in Reference 11.

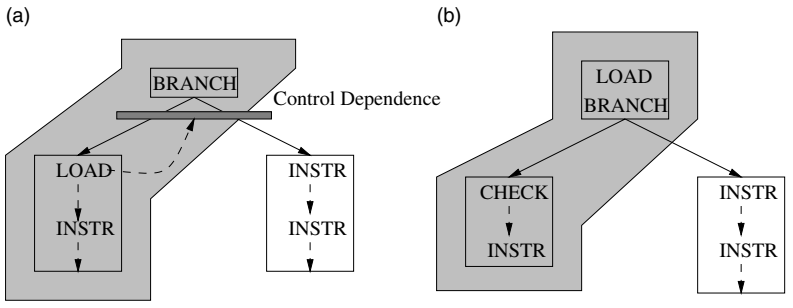


FIGURE 24.12 Instruction sequence (a) traditional execution and (b) speculative execution.

24.6 Industry Trends

The microprocessor industry is one of the fastest moving industry today. Healthy demands from the market place have stimulated strong competition, which in turn resulted in great technical innovations.

24.6.1 Computer Microprocessor Trends

Recent trends of computer microprocessors include deep pipelining, high clock frequency, wide instruction issue, speculative and out-of-order execution, predicated execution, multimedia data types, large on-chip caches, floating point capabilities, and multiprocessor support. In the area of pipelining, the Intel Pentium 4 processor is pipelined approximated twice as deeply as its predecessor Pentium 3. The deep pipeline has allowed the clock of Pentium 4 processor to run at a much higher clock frequency than Pentium 3. This trend has, however, been reversed in 2005 owing to power budget limitations. The pipeline depth of the Intel IA32 microprocessors that succeeded Pentium 4 have been reduced toward that of Pentium 3.

In the area of wide instruction issue, the Pentium 4 processor can decode and issue up to three X86 instructions per clock cycle, compared to the two-instruction issue bandwidth of Pentium. More recently, the Intel Itanium I and Itanium II processors can issue up to six instructions per clock cycle. Wide instruction issue, however, requires multiported register and multiple cache access ports that can significantly increase power consumption. As a result, future microprocessors will likely maintain or reduce the issue width compared to their recent predecessors.

Pentium 4 has dedicated a very significant amount of chip area to Branch History Table, Branch Target Buffer, Reservation Stations, Load-store Queue, and Reorder Buffer to support speculative and out-of-order execution. These structures together allow the Pentium 4 processor to maintain a large instruction window within which it performs aggressive speculative and out-of-order execution. All these structures, however, consume power in an intensive manner. As a result, the trend of larger instruction windows has also slowed owing to power budget limitations.

One important trend of the computer microprocessors in general is the slow down of the increase in complexity, size, and clock frequency of processor cores. Rather, the industry is moving into incorporating multiple processor cores on the same chip. If all the cores can be productively used, such model can achieve much higher performance than a single core given the same power and chip area budget. This, however, places much more burden on the programmer, compiler, and operating system than traditional single-core models.

In the area of predicated execution, Pentium 4 supports a conditional move instruction that was not available in Pentium. This trend is furthered by the next generation IA-64 architecture where all instructions can be conditionally executed under the control of predicate registers. This ability will allow future microprocessors to execute control-intensive programs much more efficiently than their predecessors.

In the area of data types, the multimedia instructions from Intel and AMD have become a standard feature of all X86 microprocessors. These instructions take advantage of the fact that multimedia data items are typically represented with a smaller number of bits (8–16 bits) than the width of an integer data path today (32–64 bits). On the basis of an observation, the same operation is often repeated on all data items in multimedia applications; the architects of multimedia instructions specify that each such instruction performs the same operation on several multimedia data items packed into one register word. Intel first proposed MMX instructions that process several integer data items simultaneously to achieve significant speedup in targeted applications. In 1998, AMD proposed the 3DNow! instructions to address the performance needs of 3D graphics applications. The 3DNow! instructions are designed on the basis of the concept that 3D graphics data items are often represented in single precision floating-point format, and they do not require the sophisticated rounding and exception handling capabilities specified in the IEEE Standard format. Thus, one can pack two graphics floating-point data into one double-precision floating-point register for more efficient floating-point processing of graphics applications. Note that

MMX and 3DNow! are similar in concepts applied to integer and floating-point domains. More recently, Intel proposed the SSE instructions to compete with AMD 3DNow!.

In the area of large on-chip caches, the popular strategies used in computer microprocessors are either to enlarge the first-level caches or to incorporate second-level and sometimes third-level caches on chip. For example, the AMD K7 microprocessor has a 64 KB first-level instruction cache and a 64 KB first-level data cache. These first-level caches are significantly larger than those found in the previous generations. For another example, the Intel Celeron microprocessor has a 128 KB second-level combined instruction and data cache. These large caches are enabled by the increased chip density that allows much more transistors on the chip. The Compaq Alpha 21364 microprocessor has a 64 KB first-level instruction cache, a 64 KB first-level data cache, and a 1.5 MB second-level combined cache. The recent Intel Itanium processors have up to 9 MB third-level combined cache on chip.

In the area of floating-point capabilities, the computer microprocessors in general have much stronger floating-point performance than their predecessors. For example, the Intel Pentium 4 processor achieves several times of floating-point performance improvements of the Pentium processor. For another example, most RISC and EPIC microprocessors now have floating-point performance that rival supercomputer CPUs built just a few years ago.

Owing to the increasing demand of multiprocessor enterprise computing servers, many computer microprocessors now seamlessly support cache coherence protocols. For example, the AMDK7 microprocessor provides direct support for seamless multiprocessor operation when multiple K7 microprocessors are connected to a system bus. This capability was not available in its predecessor AMD K6. The more recent AMD Opteron processors further support HyperTransport protocol to allow each processor in a multiprocessor system to have much higher communication bandwidth than what the traditional memory controllers can support.

24.6.2 Embedded Microprocessor Trends

There are three clear trends in embedded microprocessors. The first trend is to integrate a DSP core with an embedded CPU/controller core. Embedded applications increasingly require DSP functionalities such as data encoding in disk drives and signal equalization for wireless communications. These functionalities enhance the quality of services of their end consumer products. At the 1999 Embedded Microprocessor Forum, ARM, Hitachi, and Siemens all announced products with both DSP and embedded microprocessors [12].

Three approaches exist in the integration of DSP and embedded CPUs. One approach is to simply have two separate units placed on a single chip. The advantage of this approach is that it simplifies the development of the microprocessor. The two units are usually taken from existing designs. The software development tools can be directly taken from each unit's respective software support environments. The disadvantage is that the application developer needs to deal with two independent hardware units and two software development environments. This usually complicates software development and verification.

An alternative approach to integrating DSP and embedded CPUs is to add the DSP as a coprocessor of the CPU. This CPU fetches all instructions and forwards the DSP instructions to the coprocessor. The hardware design is more complicated than the first approach owing to the need to more closely interface the two units, especially in the area of memory accesses. The software development environment also needs to be modified to support the coprocessor interaction model. The advantage is that the software developers now deal with a much more coherent environment.

The third approach to integrating DSP and embedded CPUs is to add DSP instructions to a CPU instruction set architecture. This usually requires brand new designs to implement the fully integrated instruction set architecture. The benefit is that software developers need to deal with just one development environment.

The second trend in embedded microprocessors is to support the development of single-chip solutions for large volume markets. Many embedded microprocessor vendors offer designs that can be licensed

and incorporated into a larger chip design that includes the desired input/output peripheral devices and application specific integrated circuit (ASIC), and field programmable gate array (FPGA) design. This paradigm is referred to as system-on-a-chip design. A microprocessor that is designed to function in such a system is often referred to as a licensable core.

The third major trend in embedded microprocessors is aggressive adoption of high-performance techniques. Traditionally, embedded microprocessors are slow to adopt high-performance architecture and implementation techniques. They also tend to reuse software development tools such as compilers from the computer microprocessor domain. However, owing to the rapid increase of required performance in embedded markets, the embedded microprocessor vendors are now making fast moves in adopting high-performance techniques. This trend is especially clear in the DSP microprocessors. Texas Instruments, Motorola/Lucent, and Analog Devices all ship aggressive EPIC/VLIW DSP microprocessors and associated compilers.

24.6.3 Microprocessor Market Trends

Readers who are interested in market trends for microprocessors are referred to Microprocessor Report, a periodic publication by MicroDesign Resources (www.MDRonline.com). In every issue, there is a summary of microarchitecture features, physical characteristics, availability, and pricing of microprocessors.

References

1. J. Turley, "RISC volume gains but 68K stll reigns,"*Microprocessor Report*, vol. 12, pp. 14–18, January 1998.
2. W. W. Hwu and T. M. Conte, "The susceptibility of programs to context switching," *IEEE Transactions on Computers*, vol. C-43, pp. 993–1003, September 1994.
3. J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*. San Francisco, CA: Morgan Kaufman, 1990.
4. J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the 8th International Symposium on Computer Architecture*, pp. 135–148, May 1981.
5. Y. N. Patt,W.-M. Hwu, and M. Shebanow, "HPS, a new microarchitecture: rationale and introduction," in *Proceedings of the 18th AnnualWorkshop on onMicroprogramming*, pp. 103–106, December 1985.
6. L. Gwennap, "Klamath extends P6 family," *Microprocessor Report*, vol. 1, pp. 1–9, February 1997.
7. T. Y. Yeh and Y. N. Patt, "A comprehensive instruction fetch mechanism for a processor supporting speculative execution," in *Proceedings of the 25th International Symposium on Microarchitecture*, pp. 129–139, December 1992.
8. R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM Journal of Research and Development*, vol. 11, pp. 25–33, January 1967.
9. J. R. Allen, et al. "Conversion of control dependence to data dependence," in *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, pp. 177–189, January 1983.
10. V. Kathail, M. S. Schlansker, and B. R. Rau, "HPL PlayDoh architecture specification: Version 1.0," Tech. Rep. HPL-93-80, Hewlett-Packard Laboratories, Palo Alto, CA, February 1994.
11. S. A. Mahlke, et al. "Sentinel scheduling: A model for compiler-controlled speculative execution," *ACM Transactions on Computer Systems*, vol. 11, November 1993.
12. *Embedded Microprocessor Forum*, (San Jose, CA), October 1998.

25

Control with Embedded Computers and Programmable Logic Controllers

Hugh Jack
Andrew Sterian
Grand Valley State University

25.1 Introduction

Modern control systems include some form of computer, most often an embedded computer or programmable logic controller (PLC). An embedded computer is a microprocessor- or microcontroller-based system used for a specific task rather than general-purpose computing. It is normally hidden from the user, except for a control interface. A PLC is a form of embedded controller that has been designed for the control of industrial machinery (see Figure 25.1).

A block diagram of a typical control system is shown in Figure 25.2. The controller monitors a process with sensors and affects it with actuators. A user interface allows a user or operator to direct and monitor the control system. Interfaces to other computers are used for purposes such as programming, remote monitoring, or coordination with another controller.

When a computer is applied to a control application, there are a few required specifications. The system must always remain responsive and in control of the process. This requires that the control software be real-time so that it will respond to events within a given period of time, or at regular intervals. The systems are also required to fail safely. This is done with thermal monitoring for overheating, power level detection for imminent power loss, or with watchdog timers for unresponsive programs.

25.2 Embedded Computers

An embedded computer is a microprocessor- or microcontroller-based system designed for dedicated functionality in a specialized (i.e., nongeneral-purpose) electronic device. Common examples of embedded computers can be found in cell phones, microwave ovens, handheld computing devices, automotive systems, answering machines, and many other systems.

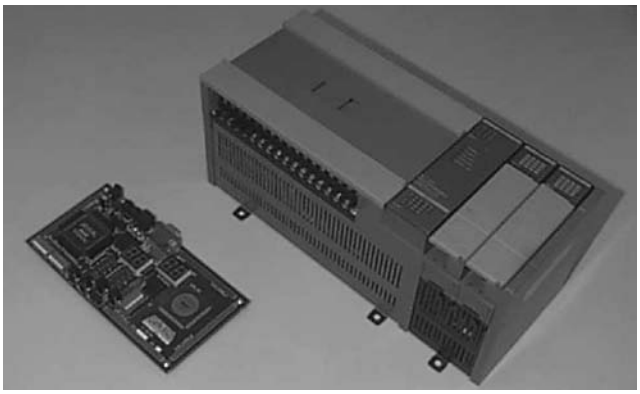


FIGURE 25.1 An embedded computer with an Altera FPGA (front-left) and an Allen Bradley SLC500 programmable logic controller (top-right).

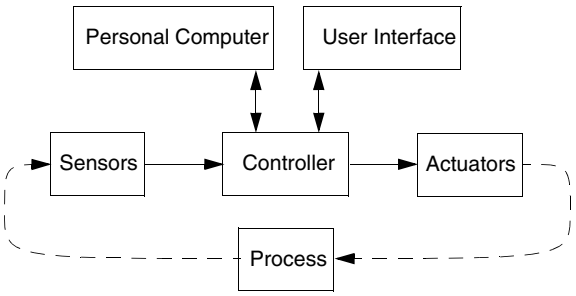


FIGURE 25.2 An example block diagram of a computer controlled application.

The design constraints and parameters for an embedded computer are usually different from those of a general-purpose computer. Although the latter is designed for maximum computing power and support for the latest interconnection and peripheral standards, an embedded computer is designed to be just powerful enough and to support only the interfaces and protocols that are specifically required. The constraints of an embedded computer design often include size, power consumption and heat dissipation, and cost.

25.2.1 Hardware Platforms

25.2.1.1 Microcontroller-Based Systems

Microcontrollers are closely related to the microprocessors that power today's general-purpose computers. They differ from microprocessors, in general, by being highly integrated, with built-in peripherals that minimize total system part count, having low power consumption, providing a small amount of on-chip RAM and ROM, and having several general-purpose input/output (I/O) lines available for instrument sensors and control. For this reason, a microcontroller-based embedded system may be designed with very few external components. In contrast, a microprocessor-based system requires external RAM, external peripherals, and I/O interfaces, and often dissipates so much heat that active cooling is required for proper operation.

The peripherals built into many microcontrollers include serial-line interfaces (such as RS232), timers, pulse generators, event counters, etc. These peripherals support many sensor and actuator control functions.

For example, pulse generators and timers can be used to construct stepper motor drive sequences. Microcontrollers are becoming increasingly specialized with respect to the data communication interfaces they support. While many support the ubiquitous RS232, SPI, and I^2C protocols, recent microcontrollers have built-in support for interfaces such as USB.

In order to minimize power consumption, most microcontrollers have a special sleep or standby mode in which no instructions are executed and very little power is consumed. Microcontrollers can be programmed to awaken in response to an external event so that the program code is executed, and power consumed, only when necessary.

Microcontrollers are a very large semiconductor market due to the wide range and high volume of devices that use them. There are many manufacturers and models of microcontrollers, ranging from tiny 8-pin devices with minimal functionality and costing mere pennies, to large devices with hundreds of pins, many features, and much higher cost. This broad spectrum reflects the highly specific nature of an embedded computer and its design.

25.2.1.2 FPLD-Based Systems

Field-programmable logic devices (FPLDs) such as complex programmable logic devices (CPLDs) and-field-programmable gate arrays (FPGAs) are a more recent alternative to microcontrollers for embedded computer design. An FPLD represents a programmable hardware device; the actual hardware functionality of the device is what is being designed. A microcontroller, in contrast, has fixed hardware functionality and is programmed with software. It is possible, however, to design an FPLD that behaves as a microcontroller, and is further programmed in software. The programmable hardware functionality, however, affords the designer a much greater degree of flexibility over a fixed hardware solution. The price for this flexibility, however, is complexity.

FPLDs may be designed from the ground up or may be composed of one or more predesigned core and peripheral blocks. It is possible, for example, to purchase microcontroller core functions, peripheral functions, etc. and assemble them to form a customized microcontroller on an FPLD with non-recurring engineering (NRE) costs that are much lower than a full custom chip design.

FPLDs can often be programmed "on-the-fly," allowing for reconfigurable computing. This is a computing paradigm that reprograms a system at the hardware level while it is in operation, according to system demands. This means, for example, that the same hardware device can implement multiple bus protocols, interfaces, or algorithms as needed, rather than requiring a larger and more expensive device that supports all of the necessary functions but only uses one at a time.

25.2.1.3 Digital Signal Processing Systems

Digital signal processing (DSP) devices are in many ways similar to microcontrollers with respect to peripheral integration, power consumption, etc. but also have specialized hardware support for common DSP operations, such as filtering. DSP devices are ideal for use in systems that process speech and music, or for robust control and communications applications. The specialized hardware support of these devices means that they are capable of sustaining much higher effective computation rates (on signal processing tasks), but at the same clock speed and power dissipation as the more general-purpose microcontrollers.

25.2.1.4 Real-Time Systems

Most embedded systems must operate in *real time*, that is, they must respond in a timely fashion to external events such as user commands and sensor readings. When an absolute upper limit on response time is required (and guaranteed), the system is a *hard real-time system*; otherwise, it is a soft real-time system. Systems that have safety constraints, such as automotive and industrial control systems, are often hard real-time systems so that absolute maximum time delays can be computed and verified for safety-critical events.

Real-time computation is effected using *interrupts*. These are mechanisms supported by all common microcontrollers that cause a change in the flow of execution of the program when the interrupt occurs. The program that is executed in response to the interrupt is expected to respond in some way to the interrupt.

For example, an interrupt may occur when a digital logic level changes at a device pin, indicating a sensor condition, or it may occur when the user presses a button on a keypad, indicating that an action is desired and is to be performed immediately.

The implementation of a real-time software system may either be custom designed or may make use of a commercial real-time operating system (RTOS). Since the design of an interrupt-driven real-time system has many potential pitfalls, the usage of a mature RTOS can greatly speed development time.

25.2.1.5 Embedded Modules

The functionality of commercially available embedded computing modules has been steadily increasing. It is common to find powerful microcontrollers, an Ethernet interface, and basic Internet protocol support all combined in a very small form factor for less than $50 in single quantities. This level of integration can greatly speed development time for network-enabled control or remote sensing applications.

25.2.2 Hardware Interfacing

25.2.2.1 Mechanical Switches

Switches are easily interfaced to digital logic with a resistor as shown in Figure 25.3. The mechanical nature of the switch may lead to *bounce* or oscillation of the digital signal for a brief period during the switch opening/closing action. This bounce may be eliminated in the software or with a small amount of additional hardware.

25.2.2.2 Analog Inputs

Analog inputs that indicate one of the two conditions can be interfaced to a digital logic input with a simple comparator (Figure 25.4). A threshold voltage is set with a resistor divider. The comparator generates a digital signal, which indicates whether the analog input voltage is above or below the threshold voltage. This approach can be used for sensors such as optical interrupters (for part counting, motor movement detection, etc.), temperature limit sensors, and many others.

When the analog voltage itself is of interest (as in, for example, temperature measurements), an analog-to-digital converter (ADC) can be used to provide either a serial or a parallel representation of the voltage with a precision ranging anywhere from 8 bits to 16 bits and above. A serial ADC may require as few as two digital I/O pins on a microcontroller for transferring data, while a parallel ADC requires at least as

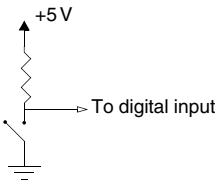


FIGURE 25.3 A mechanical switch is easily interfaced to a digital input on a microcontroller using a single resistor.

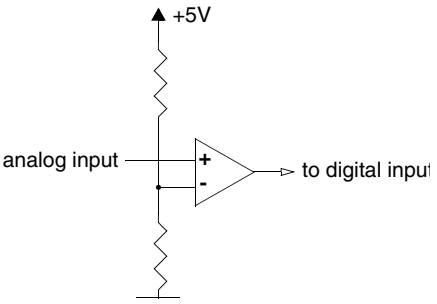


FIGURE 25.4 A digital input driven by a comparator detects whether an analog voltage signal is above or below a threshold voltage (set with a resistor divider network).

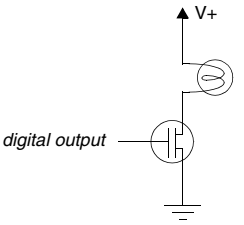


FIGURE 25.5 A digital output can control a high-current device (such as a lamp pictured to the right) using a transistor as a switch.

many pins as there are bits of resolution, but can transfer an entire analog reading in one transaction for faster throughput.

Some microcontrollers have built-in ADC peripherals with multiple input channels enabling highly integrated low-cost analog sensor systems.

Analog voltages that are very small may be amplified with an instrumentation amplifier prior to analog-to-digital conversion. Instrumentation amplifiers have very high gain and high impedance, and hence, are suitable for sensors with very weak driving voltages and currents.

25.2.2.3 Simple Actuators

Embedded computers are not usually capable of driving most practical actuators directly, since the latter often require voltages and currents not compatible with digital circuitry. As with sensors, however, some simple interface circuitry is all that is required. Simple on/off actuators such as lamps, LEDs, relay coils, etc. can be driven from a digital output, using a transistor as a switch, as shown in Figure 25.5. The digital output controls the on/off state of the transistor, which, in turn, either allows or does not allow current to flow through the actuator.

Motors can also be controlled using transistors as interfaces between digital outputs and the high-current motor coils. The lamp in Figure 25.5 can be replaced with a DC motor to allow simple on/off control of the motor. A set of four transistors arranged in an H-bridge configuration allows such a motor to rotate in either direction. Two H-bridge configurations can be used to control a stepper motor. In all cases, the speed and direction of rotation are under direct control of the embedded computer through its digital outputs.

25.2.2.4 Analog Outputs

For actuators that require a variable analog voltage or current, a digital-to-analog converter (DAC) can be used as an interface between the embedded computer and the actuator. As with ADCs, DACs are available in a variety of bit widths, conversion speeds, number of channels, etc. Often, the current driving capacity of these devices is not sufficient and an additional buffer amplifier is required to meet the current demands of the actuator.

25.2.3 Programming Languages

Embedded computers are most commonly programmed in low-level languages for maximum control over the hardware resources. The most time-critical sections of the code are generally programmed in assembly language, which is the lowest-level language understood by a microcontroller. The C language is generally used for higher-level structured programming. Even higher-level languages, such as C++ or Java, are not well suited for embedded programming as they require larger amounts of memory and are not designed for low-level access to hardware resources.

Figure 25.6 shows a fragment of an assembly language program written for the Microchip PIC 16F84A microcontroller. It enables power to a DC motor (through an external interface circuit) when two digital inputs are both at a logic 1 level. The code runs continuously, always checking for the status of the two digital inputs (which may be manual switches, current sensors, etc.).

The same code fragment written in C is shown in Figure 25.7. The code is more compact and easier to read since C is a higher-level language than assembly.

```
loop:
    btfss   PORTA,0     Check digital input bit 0 of Port A
    goto    turnoff         and disable motor if not 1
    btfss   PORTA,1     Check digital input bit 1 of Port A
    goto    turnoff         and disable motor if not 1

    bsf     PORTB,5     Enable motor by setting bit 5 of Port B
    goto    loop            and check inputs again

turnoff:
    bcf     PORTB,5     Disable motor by clearing bit 5 of Port B
    goto    loop            and check inputs again
```

FIGURE 25.6 Fragment of assembly code for Microchip PIC 16F84A microcontroller. This code fragment examines two digital inputs (bits 0 and 1 of input Port A) and sets bit 5 of output Port B if both inputs are at a logic 1 level. The output can be used to enable or disable a DC motor with appropriate interface circuitry.

```
while (1) {
        if (PA0 && PA1) {// Check status of bits 0 and 1 in Port A
                PB5 = 1;      // Set bit 5 of Port B
        } else {
                PB5 = 0;      // Clear bit 5 of Port B
        }
}
```

FIGURE 25.7 Fragment of C code to effect the same functionalilty as the code in Figure 25.6.

25.3 Programmable Logic Controllers

The modern programmable logic controller (PLC) is the successor of relay-based controls. The technological shift began in the 1960s, when the limitations of electromechanical relay-based controllers drove General Motors to search for electronic alternatives. The answer was provided in 1970 by Modicon, who provided a microprocessor-based control system. The programming language was modeled after relay ladder logic diagrams to ease the transition of designers, builders, and maintainers to these new controllers. Throughout the 1970s the technology was refined and proven, and since the early 1980s they have become ubiquitous on the factory floor.

Most PLC components are in card form that can be interchanged quickly in the event of a failure. A typical PLC application has about one hundred inputs and outputs, but the scale of the applications varies widely. A small PLC costing $200 might have six inputs and four outputs. A large application might involve multiple PLCs working together over an entire plant and collectively have tens of thousands of inputs and outputs. In general, the aggregated cost of PLC hardware per input and output is approximately $10–$50. This does not include the cost of sensors (typically $50–$100), actuators (typically $50–$200), installation (typically $10–$100), design, or programming.

Manufacturing control systems always require logical control and sometimes continuous control. Logical control involves the examination of binary inputs (on or off) from sensors and setting binary outputs to drive actuators. A simple example is a photosensor that detects a box on a conveyor and actuates an air cylinder to divert the box. Continuous control systems are used less frequently because of their higher costs and increased complexity. A typical continuous controller might use an analog output card ($1000) to output a voltage to a variable frequency motor driver ($1000) to control the velocity of a conveyor.

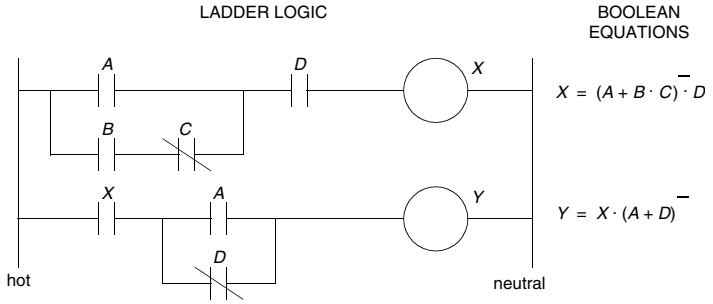


FIGURE 25.8 A simple ladder logic program with equivalent Boolean equations.

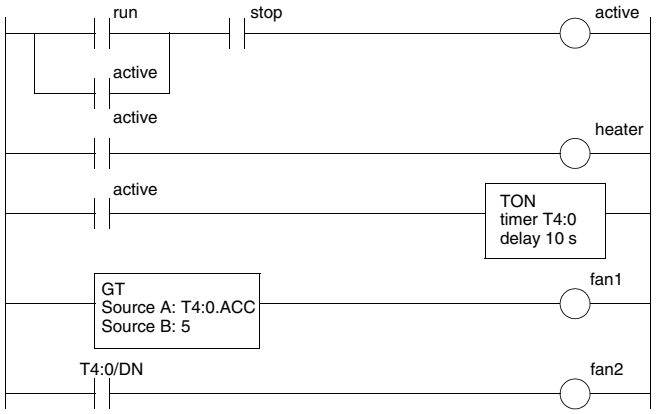


FIGURE 25.9 A complex ladder logic example.

25.3.1 Programming Languages

Every PLC can be programmed with ladder logic. Ladder logic uses input contacts (shown with two vertical lines) and output coils (shown with a circle). A contact with a slash through it represents a normally closed contact. In ladder logic, the left-hand rail is energized. When the contacts are closed in the right combinations, power can flow through the coil to the right-hand neutral rail.

Consider the ladder logic example in Figure 25.8. It is assumed that the *hot* rail at the left side has power, and the right side rail is *neutral*. When the contacts are opened and closed in the right combinations they allow power to flow through the output coils, thus actuating them. The program logic is interpreted by working from the left side of the ladder. In the first rung if A and D are on, the output X will be turned on. This can also be accomplished by turning B on, turning C off, and turning D on. In the second, the output Y will be on if X is on and A is on, or D is off. Notice that the branches behave as OR functions and the contacts in line act as an AND function. It is possible to write ladder logic rungs as Boolean equations, as shown on the right-hand side of the figure.

The example in Figure 25.8 contains only conditional logic, but Figure 25.9 shows a more complex example of a ladder logic program that uses timers and memory values. When the *run* input is active, output *heater* will turn on, 5 s later *fan1* will turn on, followed by *fan2* at 10 s. The first rung of the program will allow the system to be started with a normally open *run* push button input, or stopped with a normally closed push button *stop*. All stop inputs are normally closed switches, so the contact in this rung needs to be normally open to reverse the logic. The output *active* is also used to branch around the *run* to *seal-in* the run state. The next line of ladder logic turns on an output *heater* when the system is active. The third

line will run a timer when *active* is on. When the input to the *TON* timer goes on, the timer *T4:0* will begin counting, and the timer element *T4:0.ACC* will begin to increment until the delay value of 10 s is reached, at this point the timer done bit *T4:0/DN* bit will turn on and stay on until the input to the timer is turned off. The fourth rung will compare the accumulated time of the timer and if it is greater than 5 the output *fan1* will be turned on. The final rung of the program will turn on *fan2* after the timer has delayed 10 s.

A PLC scans (executes) a ladder logic program many times per second. Typical execution times range from 5 to 100 ms. Faster execution times are required for processes operating at a higher speed.

The notations and function formats used in Figure 25.9 are based on those developed by a PLC manufacturer. In actuality, every vendor has developed a different version of ladder logic.

25.3.1.1 IEC 61131-3 Programming Languages

The IEC 61131 standards (formerly IEC 1131) have been created to unify PLCs [3,5]. The major portions of the standard are listed below.

IEC 61131-1 Overview
IEC 61131-2 Requirements and Test Procedures
IEC 61131-3 Data Types and Programming
IEC 61131-4 User Guidelines
IEC 61131-5 Communications
IEC 61131-7 Fuzzy Control

The most popular part of the standard is the programming specification, IEC 61131-3. It describes five basic programming models including ladder diagrams (LD), instruction list (IL), structured text (ST), sequential function charts (SFC), and function block diagrams (FBD). These languages have been designed to work together. It is possible to implement a system using a combination of the languages, or to implement the same function in different languages. A discussion of ST, SFC, and FDB programs follows.

Structured Text

A structured text program is shown in Figure 25.10. This program has the same function as the previous ladder logic example. The first line defines the program name. This is followed by variable definitions. The variables *run* and *stop* are inputs to the controller from sensors and switches. The variables *heater*,

```
PROGRAM example
VAR_INPUT
      run : BOOL ;
      stop : BOOL ;
END_VAR
VAR_OUTPUT
      heater : BOOL ;
      fan1 : BOOL ;
      fan2 : BOOL ;
END_VAR
VAR
      active :BOOL ;
      delay : TON ;
END_VAR
      active :=(run OR active) & stop ;
      heater := active ;
      delay(EN :=active,PRE := 10) ;
      IF ( delay.ACC >5 ) THEN
                              fan1 :=1 ;
      ELSE
                              fan1 := 0 ;
      END_IF ;
      fan2 :=delay.DN ;
END_PROGRAM
```

FIGURE 25.10 A structured text program equivalent to Figure 25.9.

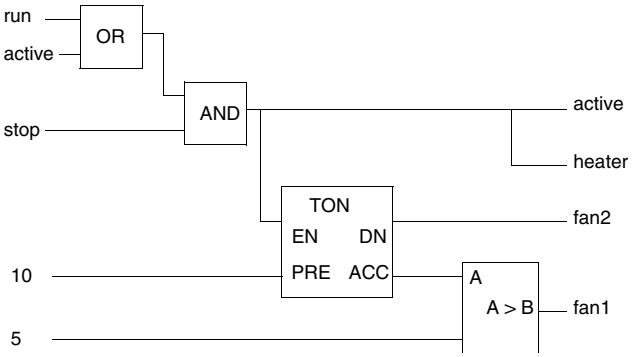


FIGURE 25.11 A FBD program equivalent to Figure 25.9.

fan1, and *fan2* are outputs to the actuators in the system. The variables *active* and *delay* are internal to the program only.

The program section immediately follows the variable declarations. In the program the first two lines set the values of *active* and *heater*. The instruction *delay* (*. . .*) calls the instantiated timer. The argument *EN* := *active* sets the timer to run, and *PRE* := 10 sets the timer delay to 10 s. The following lines use an "if" statement to set the value of *fan1*, using the accumulated timer value *delay.ACC*. The value of *fan2* is then set when the timer accumulator has reached the delay time and set the done bit *delay.DN*.

Structured text is popular and shows potential for eventually replacing ladder logic as the most popular programming language.

Function Block Diagrams

A data flow model is the basis of function block diagrams. In these programs, the data flows from the inputs on the left to the outputs on the right. The example in Figure 25.11 is equivalent to the previous ladder logic example. The *OR* and *AND* functions are used to set the values of *active* and *heater*. The *TON* timer uses the enable *EN* and delay *PRE* inputs to drive the accumulator *ACC* and *DN* outputs. The *DN* output drives *fan2* while the *ACC* value is compared to the value of 5 to set the output *fan1*.

Data flow diagrams can be very useful for doing a high-level design of a control system.

Sequential Function Charts

An SFC is used to describe a system in terms of *steps* and *transitions*. A step describes a mode of operation or state in which some *action* is performed, normally setting outputs. Transitions determine the change of states, normally by examining inputs. (Note: Some readers may notice that SFCs are based on Petri nets.)

Figure 25.12 shows an example of an SFC to control storage tanks. When the controller is started and the *power* input goes true, it will empty the tanks. After that the *run* input will start cycles where both the tanks are filled and then emptied repeatedly.

In this example, the flow of control begins at the initial step *start*, and then moves to step *S1*. The action associated with the step is *R*, which will reset, or turn off the outputs *in_valve1* and *in_valve2*. The system will remain in step *S1* until the transition is fired by input *power*. After this there are two possible paths. If *empty1* and *empty2* are both true, the left-hand branch will be followed, otherwise the right-hand transition will fire and that branch will be followed. The left-hand branch sets the *run_light* on (with *S*), and turns off the *outlet_valve*. The right-hand branch will turn on *outlet_valves* until the inputs *empty1* and *empty2* are both on. At that point *run_light* will be turned on, and the *out_valves* turned off. Regardless of which branch was followed, the flow of execution will pause at the following transition until the input *run* becomes true.

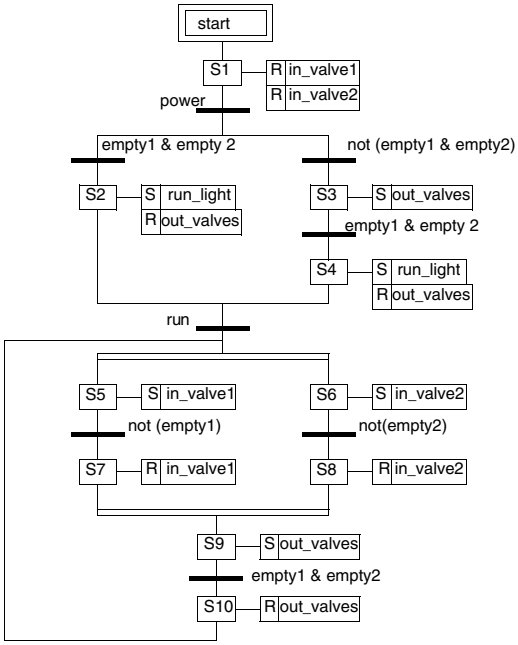


FIGURE 25.12 An SFC program for tank level control.

After the *run* transition is fired the flow of execution splits into both the left and right branches, as indicated by the two horizontal lines. The left branch fills one tank, while the right branch fills the other tank independently. When both branches are complete the flow of execution rejoins at the second set of horizontal lines, and then activates step *S9*. After step *S10* the flow of execution returns to the point after the *run* transition.

The SFC programming method differs from other programming methods in that the program is not expected to run completely in a single scan, while all others must run completely in each scan.

25.3.2 Interfacing

The installation and interfacing requirements for PLCs are driven by the need to protect people and equipment by failing safely. A typical wiring diagram for a PLC application is shown in Figure 25.13. At the top of the diagram a transformer is used to step down a higher supply voltage. This is immediately followed by a power disconnect and fuses. The power is then split into left and right rails, much like the ladder diagrams discussed earlier. Line 10 shows a master power control for the system. This includes a normally open start button and a normally closed stop button. These switches control a master control relay (MCR) *C1*. Notice that if power is supplied to the coil *C1*, it will close the contacts *C1* on the same run and hold *C1* on until the stop button is pushed. Another set of contacts is used on the left rail to disconnect power from the inputs to the PLC and the DC power supply. This control circuitry external to the PLC is required so that the stop buttons of a control system are able to directly disconnect the power. This is often required by law.

In this example the PLC is powered with 120 V AC, connected between the power rails. There are two 120 V AC inputs from normally open push buttons. The 24 V DC power supply is input to the *V+* on the output terminals of the PLC, which will then switch output power to solenoid *S1* and indicator light *L1*.

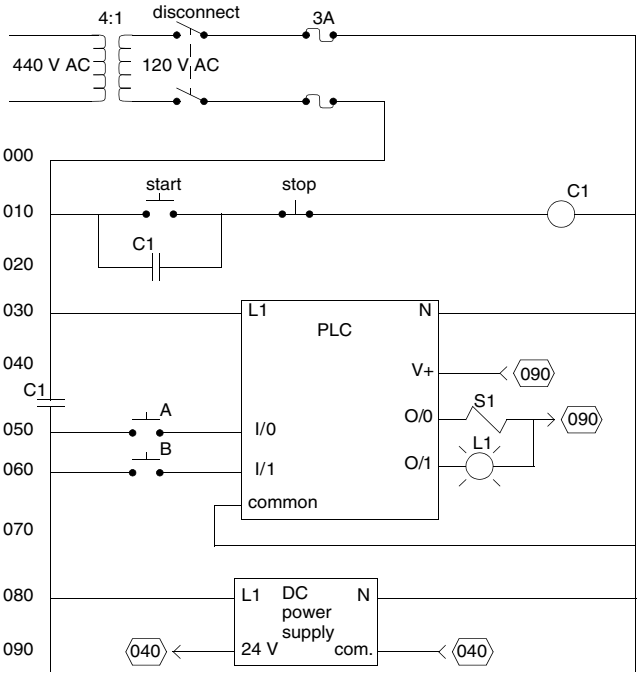


FIGURE 25.13 A PLC wiring example.

25.3.3 Advanced Capabilities

PLCs are often used in applications that go beyond basic logic solving. Some advanced programming and input/output (I/O) functions are listed below:

Calculations—The ability to do basic scientific calculations. Lower end PLCs only use integer math, while higher end PLCs also provide floating point math.
Analog I/O—Continuous voltage and current values can be input and output.
Feedback control—Proportional integral derivative (PID) controller calculations are provided as function blocks and can be used with analog I/O.
Communications—The ability to transmit data as strings over serial ports or to transfer parts of the PLC memory using proprietary protocols.
ASCII strings—Functions to manipulate ASCII strings.
System—Fault detection, status monitoring, interrupt routines, etc.
Fuzzy logic—Some PLCs include fuzzy set functions for nonlinear control problems.

At a minimum PLCs use communications for programming. But in many applications PLCs are used to communicate with other devices. In the past, most communications were based on proprietary, or closed, standards. More recently a few open communication standards have been developed and are supported by many vendors; these include Profibus, DeviceNet, CanBus, and ModBus. There has also been a trend to use more universal communication standards such as RS-232, RS-422, RS-485, and Ethernet. An example of an automation system is shown in Figure 25.14. An RS-232 connection is used between a laptop computer (e.g., COM1) and PLC1 for programming. DH+ is used to connect PLC1, PLC2, and the HMI; it is a proprietary communication standard developed by Allen-Bradley. An operator can use the human machine interface (HMI) to display data and accept operator input and communicate these values directly to both PLCs. Devicenet, an open automation standard, is used to connect PLC2 to a welding controller.

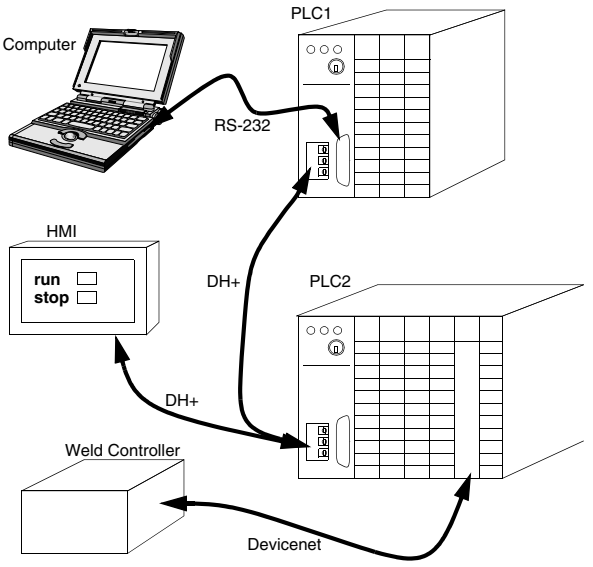


FIGURE 25.14 PLC communication example.

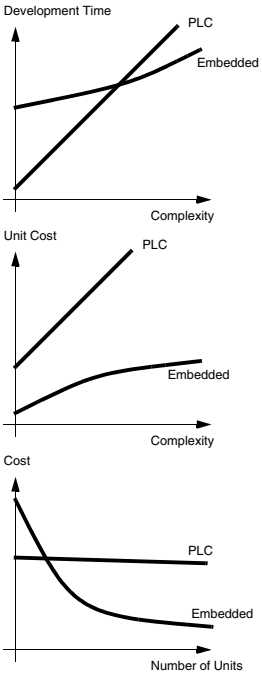


FIGURE 25.15 Relative trade-offs between control solutions.

25.4 Conclusion

PLCs and embedded controllers are complementary technologies and, when applied strategically, they will both provide low cost and reliable solutions to control problems. Figure 25.15 shows the relative trade-offs between the controllers. In general, an embedded controller requires more initial development time than a PLC for a simple system. As the system grows more complex, the embedded controller benefits

from the existence of software libraries and design tools. When using a PLC the cost of the purchased hardware will always be higher per unit. The development costs for an embedded computer will usually be higher, but these become minimal when amortized over a large number of units. As a result, embedded controllers are typically selected for applications that will be mass-produced and allow a greater development time, such as a toy robot. PLCs are often selected for applications that only require a few controllers and are to be completed in a relatively short time, such as the production machines to make a toy.

References

1. Bryan, L.A., Bryan, E.A., *Programmable Controllers,* Industrial Text and VideoCompany, 1997.
2. Filer, R., Leinonen, G., *Programmable Controllers and Designing Sequential Logic,* Saunders College Publishing, 1992.
3. Lewis, R.W., *Programming Industrial Control Systems using IES1131-3,* The Institution of Electrical Engineers, 1998.
4. Petruzella, F., *Programmable Logic Controllers,* 2nd Ed., McGraw-Hill, 1998.
5. *Programmable Controllers—Part 3: Programming Languages,* IEC 61131-3 Ed. 1.0, 1993.
6. Stenerson, J., *Fundamentals of Programmable Logic Controllers, Sensors and Communications,* Prentice-Hall, 1998.
7. Webb, J.W., Reis, R.A., *Programmable Logic Controllers, Principles and Applications,* Prentice-Hall, 1995.

26

Graphical System Design for Embedded Systems

Shelley Gretlein
National Instruments, Inc.

26.1 Introduction

Nearly 50% of embedded system designs are late or never to market and nearly 30% fail after release [1]. Many of the challenges are a direct result of embedded systems becoming more complex as average code size has increased nearly an order of magnitude over the past 5 years [2]. In addition, with embedded systems becoming more common place, many domain experts, such as machine builders, test engineers, or control engineers, need embedded technologies to develop their systems, yet do not currently have the expertise required to develop embedded systems.With the increase in system complexity and with an ever-increasing number of nonembedded experts requiring this technology, there is a strong need for a new approach to embedded design. Graphical system design is a revolutionary approach to solving design challenges that blends intuitive graphical programming and flexible commercial off-the-shelf (COTS) hardware to help engineers and scientists more efficiently design, prototype, and deploy embedded systems. Figure 26.1 illustrates the embedded system development process. The graphical system design streamlines the embedded development process from design to prototype to deployment. Using the graphical system design approach will enable you to use a single environment across all stages of design to increase productivity, save money, and bring embedded technology to the domain expert.

26.2 Graphical Programming for Embedded Design

Many embedded systems run autonomously and need to execute many tasks in parallel with specific timing requirements. Consider a machine control system that needs to control a linear stage, rotate multiple shafts, control lighting, and read in video data; in a system like this, there are multiple processes that must happen deterministically, in real-time, and in parallel. Using traditional, text-based tools such

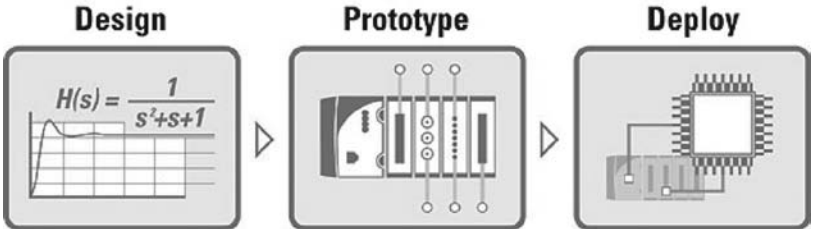


FIGURE 26.1 Graphical system design streamlines the embedded development process from design to prototype to deployment.

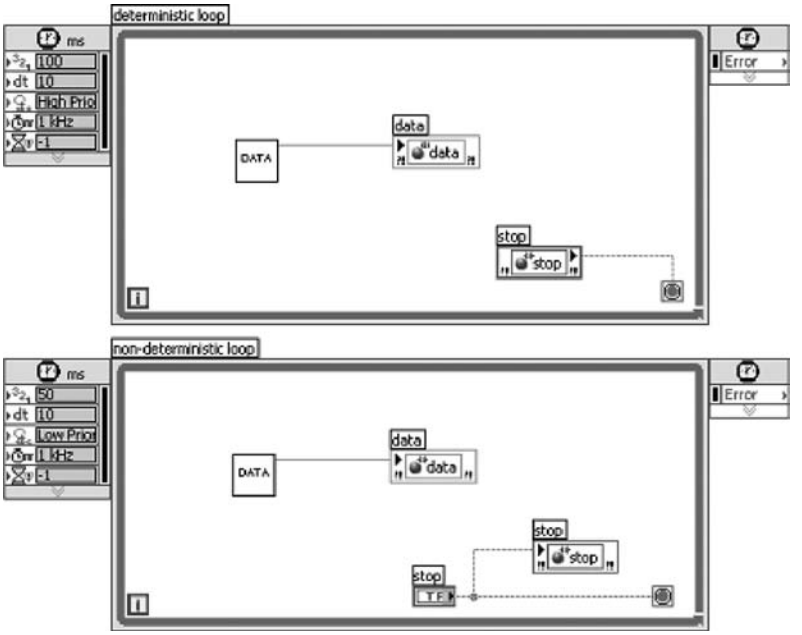


FIGURE 26.2 Parallel timed loops in the graphical programming environment of LabVIEW intuitively display parallel tasks.

as C for these applications quickly escalates in complexity. Graphical programming facilitates complex programming and timing models easily because of native functionality. More than 20 years ago, NI created components and technology in the formof the LabVIEW graphical development environment. LabVIEW natively incorporates timing into code using coding structures to implement timing, and representing parallelism is simply a matter of drawing another loop as shown in Figure 26.2.

To implement this level of timing and parallelism in text code is a barrier to a broader audience of domain experts, where the graphical representation is simply much more clear and accessible to scientists and engineers. If the graphical programming paradigm was extended to chips including FPGAs and microprocessors, it would be relatively painless to manage the parallel architecture of the silicon using the same consistency and scalability. Another key requirement for embedded system design is that the software platform addresses the various algorithm design views common in real-time embedded design. Various design views as models of computation for embedded software platforms are described in Reference 3. These models of computation match the way system designers view their system to help

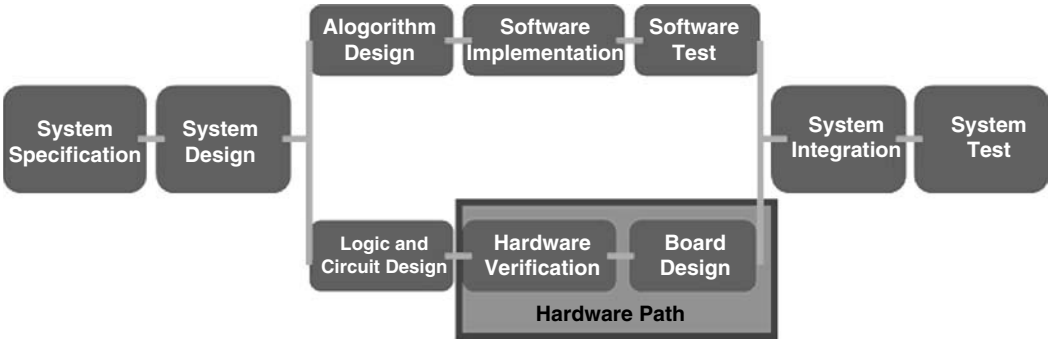


FIGURE 26.3 Typical embedded system software and hardware design flow showing separate software and hardware design flows.

minimize the complexity of translating system requirements into a software design. Over the past several years, LabVIEW has expanded to incorporate multiple models of computation to better meet the needs of embedded system designers and their varied skill sets. LabVIEW now includes text-based math, continuous time simulation, state diagrams, and graphical dataflow models to represent a variety of algorithms. LabVIEW also includes interactive tools to enhance the design experience for digital filters, control models, and digital signal processing algorithms to further ease designs in these vertical applications. You can implement algorithms on flexible COTS hardware platforms to significantly decrease your time to first prototype.

26.3 Customizable Off-the-Shelf Prototyping Platforms

As previously mentioned, with the many designs that are late or never released to market or worse, designs failing after release, something needs to be done to get more high-quality products out sooner. One way to address both of these issues is to prototype the systems better by integrating real-world signals and real hardware into the design process sooner so that high-quality designs are iterated on and problems are found (and fixed) earlier.

Consider the design flow in Figure 26.3. The hardware path is a viable location to search for efficiencies and ways to shorten the design process. Today, if you are creating custom hardware for final deployment, it is difficult to have the software and hardware developed in parallel, as the software is never tested on representative hardware until the process reaches the system integration step. In addition, you do not want the software development to be purely theoretical, because waiting until the system integration test to include I/O and test the design with real signals may mean that you discover problems too late to meet design deadlines.

Most designers currently use a solution such as an evaluation board to prototype their systems. However, these boards often only include a few analog and digital I/O channels and rarely include vision, motion, or ability to synchronize I/O. In addition, designers often have to waste time developing custom boards for sensors or specialized I/O, just to complete a proof of concept. Using flexible COTS prototyping platforms instead can truly streamline this process, as shown in Figure 26.4, and can eliminate much of the work required for hardware verification and board design. Much like PCs today, where anyone can go to an electronics store and plug components such as memory, motherboards, and peripheral together to create a PC, graphical system design strives to achieve the same standardization for prototyping platforms.

For most systems, a prototyping platform must incorporate the same components of the final deployed system. These components are often a real-time processor for executing algorithms deterministically, programmable digital logic for high-speed processing or interfacing the real-time processor to other

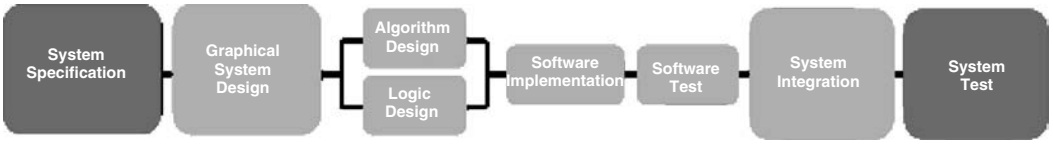


FIGURE 26.4 Stream-lined development flow with graphical system design.

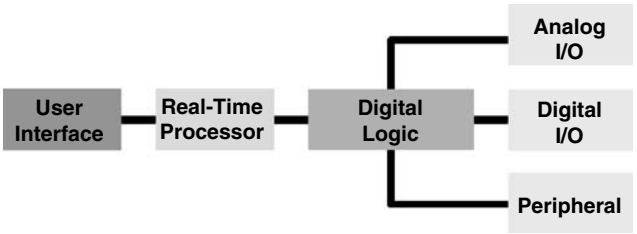


FIGURE 26.5 Typical components of an embedded system.

components, and varied types of I/O and peripherals, as shown in Figure 26.5. Finally, as with any system, if the off-the-shelf I/O does not serve all your needs, the platform should also be extensible and customizable when needed.

National Instruments offers several types of prototyping platforms including NI CompactRIO that contain all the basic building blocks of an embedded system. The controller contains a 32-bit processor running a real-time operating system. The CompactRIO backplane contains an FPGA that can implement high-speed processing and actually configure and provide interfaces to the I/O modules, which include options for analog input and output, digital input and output, and counter/timer functionally. Each of the modules includes direct connectivity to sensors and actuators as well as built-in signal conditioning and isolation. A module development kit is also available, which allows developers to expand the platform to include custom modules with all plugging into this COTS framework. In addition, CompactRIO is industrially packaged (-40°C to 70°C, 50G shock) with a small footprint (3.5 in. $\times$ 3.5 in. $\times$ 7.1 in.) and low power requirements (7–10 W typical), making it ideal for not only prototyping but also deployment of in-vehicle, machine control, and on-board predictive maintenance applications.

26.4 Custom Deployment Options

As previously mentioned, hardware devices, such as NI CompactRIO, can be employed for prototyping. It can be used for deployment because of its packaging, durability, and cost. However, there may be times when size or power may drive the need for smaller, custom board designs. To address that need, designers can preserve their software investment by using the LabVIEW Embedded Development Module to deploy code to any 32-bit processor. The LabVIEW Embedded Development Module combines all the graphical development benefits with the out-of-the-box analysis functions, integrated I/O, and interactive graphical debugging. This module is able to target any 32-bit microprocessor and provides a framework that can openly integrate a variety of existing C-based, third party tool chains and operating systems to target custom board designs. Once integrated, users can develop 100% graphically and interactively debug their application. By generating code to integrate with all existing targets in the market, users can have ultimate flexibility and breadth in target options. This new technology empowers a broader range of scientists, engineers, and domain experts to design their algorithms, develop the application, program the logic, prototype the system, and then deploy it to the target of choice.

26.5 Conclusion

The time has come for a new approach to electronic system design. Graphical system design brings a software platform combined with a hardware platform that offers the opportunity to significantly reduce development cost and time to market. A software platform that integrates multiple models of computation minimizes the time to implement specifications into a design. A flexible COTS hardware prototyping platform that supports the software platform and offers customizable components minimizes the time to first prototype, because the time and money required for designing custom hardware is eliminated. In addition, prototyping with real I/O results in a higher quality design—eliminating the design failures we see today. Finally, consistent graphical software from your design to your prototyping platform to your final deployment target maximizes code reuse and eases the transition to final deployment.

References

1. Embedded Software Development: Issues and Challenges. July 2003.
2. Rich Sevcik, "Taking the Frustration Out of Embedded Design," TechOnline, 2002, http://www.techonline.com/community/related_content/21543
3. Edward A. Lee, *Advances in Computers,* M. Zelkowitz, Ed., Vol. 56, Academic Press, London, 2002

27

Field-Programmable Gate Arrays

Daniel R. Fay
Daniel A. Connors
University of Colorado at Boulder

27.1 Introduction

Field-programmable Gate Arrays, or FPGAs, are a type of integrated circuit (IC) that contain logic components and interconnects that can be programmed after the device is manufactured in a semiconductor fabrication laboratory [1]. In short, an FPGA is a type of gate array that is programmable multiple times in the field rather than once at manufacturing time. FPGAs can host complex logic designs that include both combinational logic (AND, OR, NOT, XOR, etc.) as well as sequential logic (latches, flip–flops, etc). Most FPGAs also contain memory elements in the form of flip–flops or block memories. Modern FPGA-based chips contain up to hundreds of thousands of gates, and there are a variety of FPGA architectures on the market. In addition, there are emerging FPGA design trends to include not only programmable logic blocks but also programmable interconnects and switches between the blocks, and fixed hardware such as block memories, digital signal processing (DSP) cores, and processors. Overall, FPGA programmability allows them to be used in a variety of applications: prototyping ICs, in low-volume applications where the nonrecurring engineering (NRE) cost of designing and manufacturing a custom IC is too high, as glue logic for linking together other ICs, and for reconfigurable computing, which uses FPGAs as optimized computing engines. FPGAs provide this versatility through configurable logic and a hierarchy of flexible, programmable interconnects.

FPGAs [2] trace their roots to programmable logic devices, or PLDs, that emerged during the 1980s. These devices, known as programmable logic arrays (PLAs) and programmable array logics (PALs), allowed the designer to implement simple combinational and sequential logic. The PLA is composed of a set of programmable AND planes that connect to a set of programmable OR planes. The resulting connection of AND and OR planes creates the potential for any logic functions to be synthesized as its

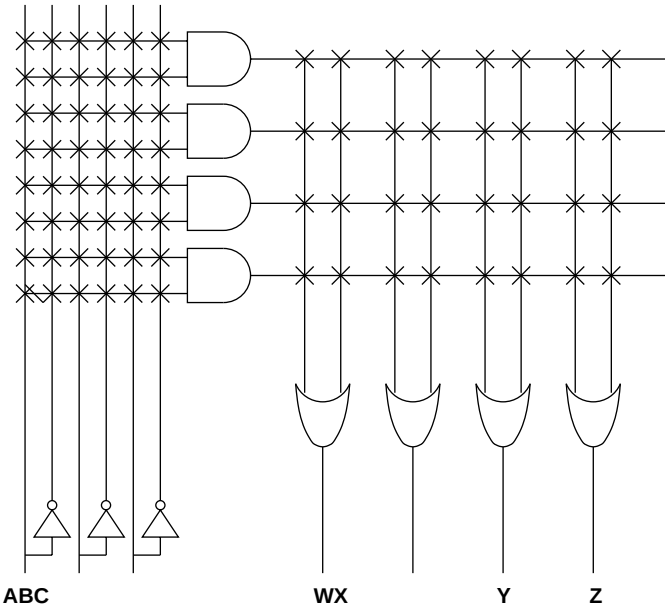


FIGURE 27.1 The respective AND and OR planes of a programmable logic array design.

sum of products (or alternatively the product of sums) form. Figure 27.1 displays the respective AND and OR planes of a PLA design. The original PLA-based designs were far more limited than FPGAs, with their functionality restricted to registering a few levels of simple sum-of-products Boolean expressions. Moreover, FPGA-generation of programmable logic, including Complex PLDs, or CPLDs, expanded these limited capabilities by adding more programmable logic and by allowing more flexible interconnect. As such, CPLDs can be used to implement functionality like feedback loops between logic blocks and even simple microcontrollers.

27.2 Field-Programmable Gate Array Architecture

At the heart of every FPGA architecture lies two basic resources: configurable logic blocks (CLBs) and routing resources. FPGAs organize the CLBs into a two-dimensional array with rows and columns of CLBs separated by routing channels. While the CLBs in modern FPGAs are often quite complicated, they all share similar basic functionality. To implement arbitrary combinational logic, they contain a lookup table (LUT), which is an $N \times N$ memory array that can implement any N-input Boolean function. The second component is a storage element such as a D flip–flop. Finally, there is also at least a multiplexer (MUX) that allows the storage element to be bypassed. These final two components allow the CLB to implement sequential logic like state machines. Figure 27.2 displays the basic view of a LUT-based CLB in modern FPGA systems.

Commercial FPGAs typically include more complex and specialized logic within their CLBs. Manufacturers often augment CLBs with multiple LUTs, fast carry chains for implementing low-latency arithmetic, and/or more sophisticated storage capabilities: some vendors' CLBs, for example, allow the LUT memory to be used as conventional read/write memory. Many FPGAs also contain various special–purpose blocks, such as memory, fast multipliers, DSP blocks, high-speed I/O, and even full-fledged microprocessors.

FPGAs can hold their configuration in different ways; however, the most common way for them to hold their configuration is using static RAM (SRAM). As compared to other configuration storage techniques, SRAM-programmed FPGAs enjoy high logic densities and can be rapidly reprogrammed

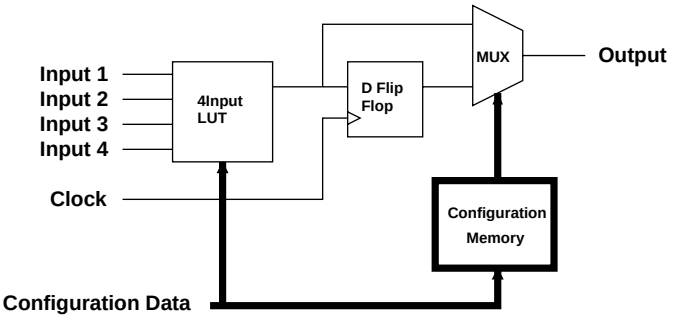


FIGURE 27.2 The basic view of a LUT-based CLB in modern FPGA systems.

multiple times; however, they lose their configuration if the FPGA loses power. Some FPGAs hold their configuration information using some kind of electrically erasable programmable read-only memory (EEPROM). FPGAs tend to have a lower logic density than SRAM-programmed FPGAs, and can only be reprogrammed a limited number (about 10,000–1,000,000) of times, but they can lose power and still retain their programming information. The final way FPGAs can store their configuration information is through either fuses or antifuses. Like EEPROM-programmed FPGAs, they retain their configuration information even after losing power; however, they can be programmed only once.

Most FPGAs have abundant clock resources consisting of dedicated low-skew logic and routing for distributing multiple clock signals around the FPGA. In addition, many FPGAs contain embedded clock managers that contain phase-locked loops (PLLs) for generating multiple arbitrary clock signals from a single external clock source.

27.3 Field-Programmable Gate Array Design Flow

To design for an FPGA, the designer first describes the behavior of the circuit. Once entered, the designer, using an electronic design automation (EDA) tool, generates a technology-mapped netlist. Place-androute tools then take this netlist and fit it to the target FPGA. After place-and-route is complete, the designer then uses the tools to generate a programming file to load on the FPGA. Timing analysis, simulation, and in-circuit logic analysis are used to debug the design throughout this process. Figure 27.3 illustrates a high-level overview of the design process using FPGAs including components of design entry, synthesis, verification, and simulation.

Design entry for FPGAs can be done in any number of ways [3]. One way is for the designer to enter the FPGA's design as a circuit schematic. Schematic entry is useful for designs that can be easily described using basic logic elements such as logic gates, flip–flops, multiplexers, encoders, and decoders. For larger projects, hardware description language (HDL) entry is usually a better choice. These languages allow the designer to describe an FPGA's functionality at a much higher level of abstraction. HDLs are like standard programming languages such as C/C++ and Java, with added extensions to support hardware-specific notations such as timesteps, interconnects, and concurrency. While there exist a variety of HDLs, the most widely used HDLs (and the ones the FPGA tools typically support) are VHDL or Verilog HDL. Many newer FPGA tools help increase designer productivity by allowing the designer to describe the design's higher levels of abstraction than those afforded by conventional HDLs. Some tools, such as Impulse-C, Handel-C, NAPA-C, and SystemC, allow the designer to describe the hardware in a modified version of the widely used C programming language. Other languages are used as well, including extensions to existing HDLs such as SystemVerilog and SystemVHDL. Tools also exist for converting DSP algorithms programmed in MATLAB into FPGA-synthesizable hardware.

Many toolchains provide libraries of predefined circuits, called IP cores. These cores are pretested and preoptimized for the target FPGA, which facilitates rapid development of complex FPGA designs. When

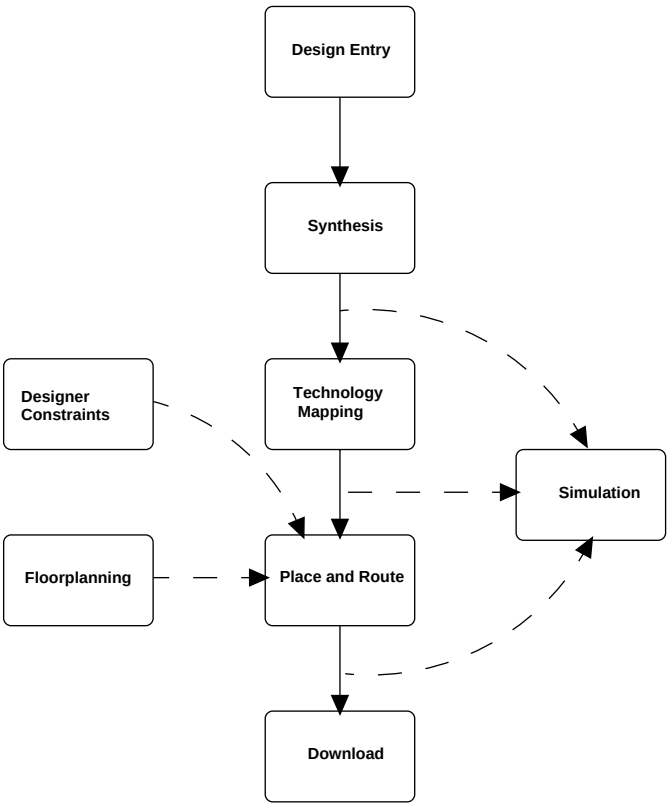


FIGURE 27.3 High-level overview of the design process using FPGAs.

the majority of the design consists of linking up predefined IP cores using standardized interconnect, the resulting design is called a system on a programmable chip (SoPC), which is analogous to the system on a chip (SoC) design flow for application-specific integrated circuits (ASICs).

Before testing the design while it is running on an actual FPGA, the FPGA designer debugs the design using a software simulator. Software simulation is first done after entering the design, to locate errors in the behavior of the circuit. After synthesis turns the circuit's behavioral description into a lower-level description, the simulation is rerun on the mapped netlist to look for bugs the synthesis tools might produce. After place and route, the design is tested again for tool-induced bugs. The final stage of testing involves testing the design while it actually runs on the FPGA. Like any other IC, FPGAs can be debugged with tools like logic analyzers and oscilloscopes. Many FPGA tools, however, also allow the designer to synthesize designs with full-speed logic analyzers. These tools synthesize logic analyzers onto the FPGA logic, allowing for full-speed, detailed analysis of the FPGA's internal circuitry.

The synthesis process outlined above consists of several key stages. First, the tools parse design entry information and convert it to an intermediate format understandable by the rest of the synthesis tools. Synthesis tools usually provide syntax checking at this stage. The tools then map the design to the functionality of the FPGA's logic blocks by packing Boolean logic functions to fit within the LUTs, by inferring arrays of storage as memory blocks, and so forth. Once the design is mapped to a technologyspecific netlist, the place and route phase begins. During this stage, the tools decide where to place the mapped blocks, with the route stage subsequently deciding how to interconnect the placed CLBs. During place and route, the tools can make tradeoffs between fast logic or logic that uses as few CLBs as possible. The designer can provide timing and area constraints to help guide the tools. It is also frequently possible for the designer to intervene at this stage to provide custom placement of the CLB mappings. Known

as floorplanning, this technique allows the designer to manually place certain functionality to either improve synthesis time (the tools only need to place and route parts of the overall design), speed, or FPGA utilization. After completing the place and route stage, the tools proceed to generate a programming bitstream to load onto the FPGA. To protect the designer's intellectual property, some FPGA vendors allow this bitstream to be encrypted.

Once the tools finish generating the programming bitstream, it is then loaded onto the FPGA. FPGAs can be programmed in any of three ways. The first is to load the bitstream using a programming cable that connects the FPGA to computer that loads in the bitstream onto the array. This programming method is most frequently used during development/prototyping. The second method, which is used-more often for shipping products, is to have the FPGA read its configuration from an external PROM or Flash memory chip on power up. Finally, on FPGAs with internal nonvolatile storage, the FPGA can be programmed to hold a configuration indefinitely.

27.4 Different Uses for Field-Programmable Gate Arrays

Field-programmable gate arrays can be put to a variety of uses. One of these is for ASIC prototyping. For ASIC prototyping, a high-level description of the ASIC is synthesized to the FPGA to check for functionality bugs as well as for basic timing checks. For simple ASICs, the entire design can be synthesized onto a single FPGA. More complicated designs, however, may require multiple FPGAs. Products such as QuickTurn and IKOS provide large-scale hardware emulation systems made up of many interconnected FPGAs for full circuit emulation of a large design.

FPGAs are also commonly used as ASIC replacements. They are desirable in low-volume applications where the high NRE costs cannot be amortized over enough units to justify the ASIC's lower per-unit cost over an FPGA. FPGAs are also useful in situations where the ASIC's design is subject to change; instead of having to qualify and manufacture an entire new set of ASICs to implement a change, the FPGA can be modified simply by reprogramming it.

FPGAs are also very useful in education of computer engineering.With the inherent attribute of being programmable, an FPGA can be used as a "programmable breadboard" that allows students to try out different designs. Their abundant logic resources allow students to implement elaborate projects with complexity ranging from hundreds of logic gates to embedded computer systems replete with multiple microprocessors, external I/O, and large external memories. A large selection of boards designed for education exist, which are made by the major FPGA vendors as well as by third-party companies. These boards usually contain an FPGA with access to external memory, switches and push buttons, light emitting diodes (LEDs), display screens, as well as expansion ports for attaching other hardware.

A growing use of FPGAs is to use them for reconfigurable computing. Reconfigurable computing leverages the vast parallel computational resources of FPGAs to accelerate certain classes of computational problems. These include DSP, bioinformatics, graphics/visualization, simulation, searching, video compression/decompression, and encryption/decryption. In general, any application that involves large amounts of computation but does not have very complex control flow (e.g., has few if-else statements and typically does not call a lot of procedures) is a candidate for FPGA acceleration. Most FPGAs do not implement floating-point arithmetic well, so applications that do a lot of floating-point calculations need to be reimplemented to only use integer arithmetic if they are to be efficiently implemented on an FPGA.

While it is possible for a reconfigurable computer to consist solely of FPGAs, virtually all reconfigurable computers contain one or more conventional microprocessors for running parts of the program that are ill-suited for running on the FPGA. A major way reconfigurable computers differentiate themselves is by how tightly coupled the FPGA logic is to the CPU. FPGA logic that is more closely coupled with the CPU logic reduces the communication latency between the CPU and the FPGA; however, these setups often have less FPGA logic available than more-decoupled designs. Generally, low-latency CPU–FPGA communication is essential when the FPGA part of the algorithm is relatively small or requires frequent communication with the CPU part of the algorithm. Production reconfigurable computers can take on

a variety of configurations. In one configuration, the FPGA fabric is completely integrated with the processor itself. In this configuration, programs implement special complex instructions on the FPGA fabric. Another type of reconfigurable computer treats the FPGA as a tightly-coupled coprocessor. In this setup, the FPGA coprocessor shares memory with the main processor, allowing fast communication via direct memory access (DMA). Other systems take a traditional computer system and add FPGAs as expansion cards in the same way as someone might add a network card or video card.

27.5 Embedding Microprocessors and Microcontrollers onto Field-Programmable Gate Arrays

In recent years, it has become possible for designers to integrate microprocessor/microcontroller designs onto an FPGA. Many FPGAs now contain enough logic resources to implement a full-fledged microprocessor/microcontroller onto the FPGA. Known as "Soft CPU Cores," these parts are a standard part of many FPGA vendors' toolchains and contain microprocessor cores specially optimized for efficient implementation on the FPGA. Designers can typically tailor the soft microprocessor for its intended use by adding/removing resource-intensive hardware components such as caches, floating-point units, and barrel shifters. In addition to soft microprocessor cores, some FPGAs actually contain embedded microprocessor blocks. Since these microprocessor blocks are entirely custom logic, they can perform much faster, have more features, and use less FPGA area than a typical soft microprocessor core; however, the designer is limited to the hard microprocessor cores available within a given FPGA family.

Designers use integrated microprocessors/microcontrollers in a variety of ways. One way is to use them to implement complicated statemachines, as it is often easier towrite a statemachine as a software program than it is to express the statemachine as a state diagramand then synthesize a custom finite statemachine. Integrated microprocessor/microcontroller cores on FPGAs are also useful as an area-saving feature; instead of spending precious FPGA hardware resources implementing infrequently used, performance insensitive functionality, these features can instead be implemented as a software program running on the microprocessor/microcontroller. An example of thiswould be awireless router; while the signal processing and packet processing features are implemented on the FPGA fabric for high performance, the web server used for configuration purposes is run as a software program on the microprocessor/microcontroller.

FPGA-integrated microprocessors/microcontrollers usually have various interfaces that allow the core to interface with the rest of the FPGA. Many contain special high-speed interface blocks that allow for connecting custom accelerator blocks and coprocessors, such as fast Fourier transform (FFT) blocks or floating-point units. In addition, these cores usually have slower buses for interfacing with memory and/or I/O. Figure 27.4 illustrates the integration of programmable components with design blocks in a modern FPGA.

27.6 Reliability and Power Considerations

Serious and growing concerns with digital logic, particularly that used in extreme environments such as space, are soft error upsets, or SEUs. SEUs occur most often when a high-energy particle, such as a cosmic ray, hits a transistor and reverses its state. While the SEU does not permanently damage the transistor, it can lead to incorrect behavior in the circuit. While SEUs can occur in any kind of modern digital logic, they are of a special concern when using SRAM-based FPGAs. If an SEU flips one of the configuration bits on an SRAM-based FPGA, it can seriously alter the functionality of the FPGA's logic, with the error persisting until the FPGA gets reconfigured.

There are several ways to mitigate these problems. First, all critical state storage and interconnect buses should be protected with at least parity to detect the errors. If on-the-fly error detection *and correction* is desired, the circuits should also have some sort of error-correcting code (ECC) implemented on the state and interconnects. Another redundancy technique that can be used in lieu of or in conjunction with

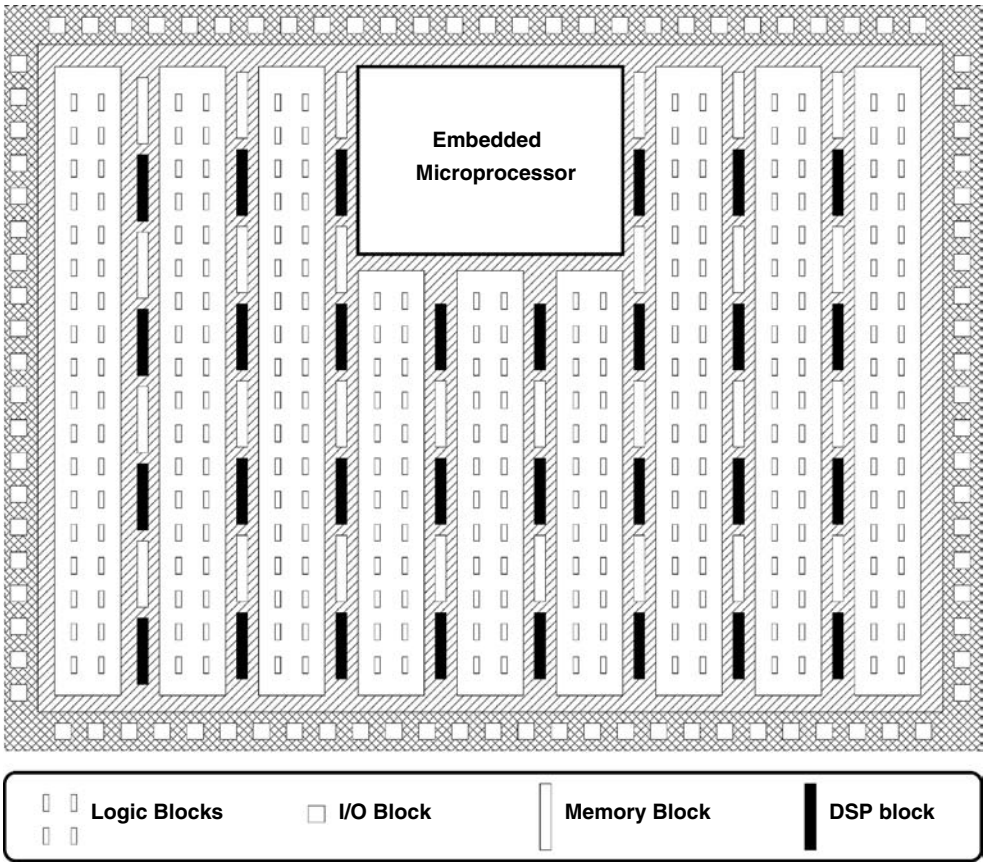


FIGURE 27.4 Integration of programmable components with design blocks in a modern FPGA.

the previously stated techniques is triple modular redundancy (TMR). TMR consists of triplicating all the on-chip logic. At regular stages throughout the design, the results of all three copies are compared using a minority voter scheme. If one of the results differs from the other two, the two matching results are then passed on downstream.

All the above techniques are effective ways to deal with SEU errors on any form of digital logic. One problem remains for SRAM-based FPGAs, however; as time goes on, the number of errors within the configuration storage grows until even ECC and TMR techniques cannot correct for the faulty logic. To mitigate this, the FPGA's configuration must be periodically refreshed to scrub out built-up configuration errors.

Power consumption is also an important consideration when designing with FPGAs. Overall, an FPGA's power needs tend to be significantly higher than for a custom ASIC; however, an FPGA typically uses less power than the equivalent algorithm implemented as a software program on a microprocessor or microcontroller. Relative to an ASIC, an FPGA design's power consumption can be easily estimated because of its premade, precharacterized logic blocks. Tools that will estimate power consumption using a simulation of the FPGA exist.

27.7 Similar Devices

Besides FPGAs and traditional PLDs, there exist a variety of other quasiprogrammable ICs; one of these is a mask programmable gate array, or MPGA. These parts start out as partially finished ICs; they come

out of the factory with their transistors finished in a regular array as well as specifying the first few interconnect layers. To finalize the product, the customer connects the finished transistors in a desired pattern by specifying the remaining interconnect layers. Such a chip plays the middle ground between the low NRE/high unit cost of FPGAs and the high NRE/low unit cost of ASICs. They accomplish this by achieving densities close to ASICs; however, the customer only needs to design a few of the uppermost (and typically the lowest cost) of the masks in the mask set.

Structured ASICs are closely related to MPGAs. Structured ASICs bring the advantages of IP blockbased SoC design to MPGAs by combining mask-programmable logic with nonprogrammable blocks such as microprocessors, embedded memories, memory controllers, and high-speed I/O blocks. Some FPGA vendors provide special structured ASICs that exactly replicate the functionality of their FPGAs, to make it easier for customers to port their design from an FPGA over to a structured ASIC.

There is an analog equivalent to digital FPGAs: field-programmable analog arrays (FPAAs). These parts contain configurable analog blocks interfaced with programmable interconnect. Each configurable analog block typically contains at least one operational amplifier ("Op-Amp") along with various passive electronic components such as resistors and capacitors.

References

1. Brown, S.D., Francis, R.J., Rose, J., and Vranesic, S.G., *Field Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
2. Trimberger, S.M., *Field-Programmable gate Array Technology*, Kluwer Academic Publishers, 1994.
3. Salcic Z. and Smailagic, A., *Digital Systems Design and Prototyping Using Field Programmable Logic*, Kluwer Academic Publishers, 1997.

28

Graphical Programming for Field-Programmable Gate Arrays: Applications in Control and Mechatronics

Jeannie Sullivan Falcon
Michael Trimborn
National Instruments, Inc.

28.1 Introduction

Embedded control systems require a high degree of reliability and determinism. However, the costs of designing and testing these control systems can be very high because of application complexity and stringent performance requirements. In addition, reducing time-to-market for product development is essential for many companies. For researchers, the use of high-level software tools in a unified environment can abstract away many implementation details. This facilitates the rapid investigation of advanced algorithms and control approaches on real-world systems.

Field-programmable gate arrays (FPGAs) are being used for an increasing number of embedded control and monitoring systems. They are less expensive than ASICs in terms of development and can also be reprogrammed in the field. FPGAs also have advantages over microprocessors in that all of the control logic is implemented in hardware. This leads to a significant advantage in terms of speed. Also, multiple tasks can run in parallel because each task will utilize a different set of gates on the FPGA. FPGAs can be used for control, signal processing, digital communication, health monitoring, and custom timing and triggering.

Programming for FPGAs has previously required knowledge of VHDL or Verilog [1]. VHDL and Verilog are taught in electrical engineering programs but not in mechanical, aerospace, biomedical, or chemical engineering programs. This has limited the proliferation of FPGAs in control and mechatronic systems. Figure 28.1 shows a LabVIEW FPGA example for pulse width modulation (PWM) output generation. Figure 28.2 shows VHDL for the same task [2].

National Instruments (NI) has recently developed several new design tools and a hardware architecture that can be used to speed the design and implementation of embedded control systems utilizing FPGAs. The design tools are based on LabVIEWFPGA—agraphical programming language for FPGAs. The hardware architecture combines an FPGA with a microprocessor and is available for desktop computers, PXI/CompactPCI industrial computers, as well as an NI embedded control system called CompactRIO.

Built-in LabVIEW FPGA functions are available for control and signal processing. Control systems with digital inputs (such as encoders) and digital outputs (such as PWM signals) can be built on the FPGA with rates up to 20 MHz. Control systems with analog inputs and outputs can be built with rates up to 500 kHz. Parallelism is inherent because separate FPGA gates are used for each loop on a LabVIEW FPGA diagram. Thus, the addition of control or monitoring tasks will not slow the overall FPGA application. The total number of tasks is limited, however, by the number of gates of the FPGA.

This paper will present a background on FPGAs, discuss programming methods and system architectures, and describe the LabVIEW FPGA module for graphical programming of FPGAs. Several applications in control and mechatronics will also be discussed.

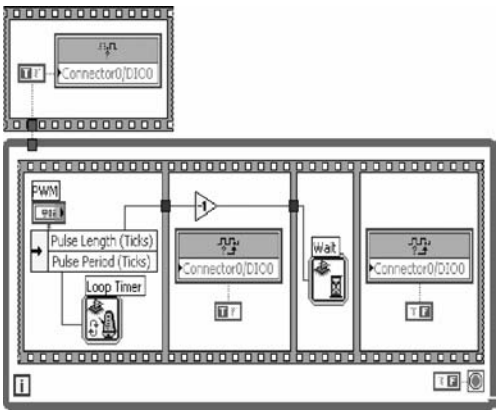
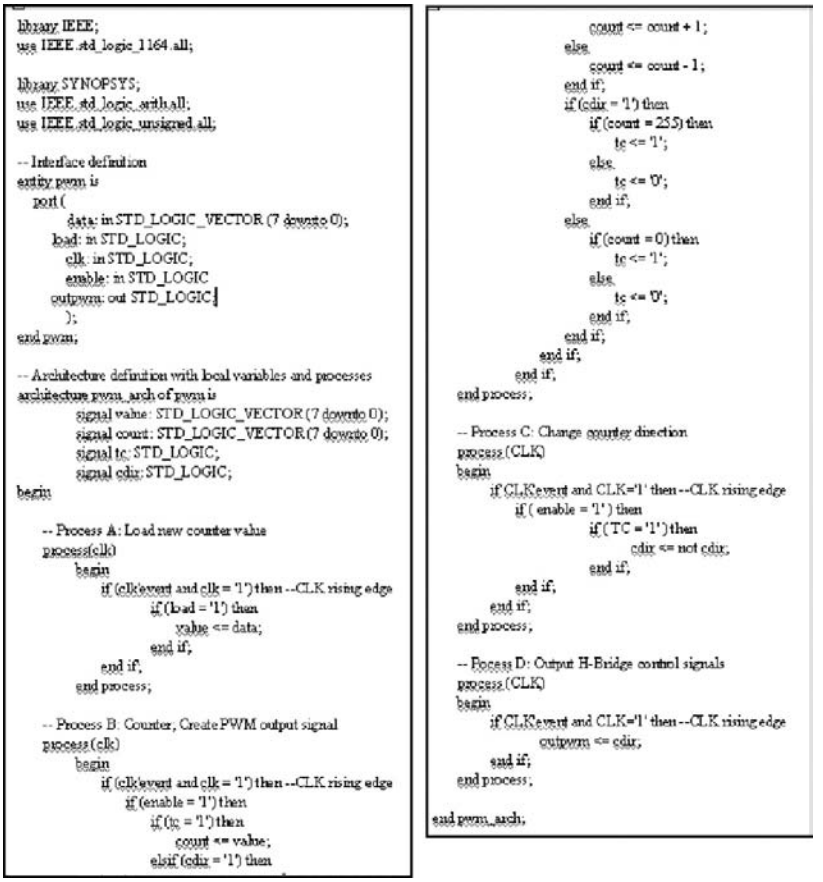


FIGURE 28.1 LabVIEW FPGA program for PWM output.



```
library IEEE;
use IEEE.std_logic_1164.all;

library SYNOPSYS;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- Interface definition
entity pwm is
  port (
      data: in STD_LOGIC_VECTOR (7 downto 0);
    load: in STD_LOGIC;
      clk: in STD_LOGIC;
      enable: in STD_LOGIC
    outpwm: out STD_LOGIC
      );
end pwm;

-- Architecture definition with local variables and processes
architecture pwm_arch of pwm is
        signal value: STD_LOGIC_VECTOR (7 downto 0);
        signal count: STD_LOGIC_VECTOR (7 downto 0);
        signal tc: STD_LOGIC;
        signal cdir: STD_LOGIC;
begin

    -- Process A: Load new counter value
    process(clk)
        begin
            if (clk'event and clk = '1') then --CLK rising edge
                    if (load = '1') then
                        value <= data;
                    end if;
            end if;
        end process;

    -- Process B: Counter, Create PWM output signal
    process (clk)
        begin
            if (clk'event and clk = '1') then --CLK rising edge
                if (enable = '1') then
                    if (tc = '1') then
                        count <= value;
                    elsif (cdir = '1') then
                        count <= count + 1;
                    else
                        count <= count - 1;
                    end if;
                    if (cdir = '1') then
                        if (count = 255) then
                            tc <= '1';
                        else
                            tc <= '0';
                        end if;
                    else
                        if (count = 0) then
                            tc <= '1';
                        else
                            tc <= '0';
                        end if;
                    end if;
                end if;
            end if;
    end process;

    -- Process C: Change counter direction
    process (CLK)
    begin
        if CLK'event and CLK='1' then --CLK rising edge
            if ( enable = '1' ) then
                    if ( TC = '1') then
                        cdir <= not cdir;
                    end if;
            end if;
        end if;
    end process;

    -- Process D: Output H-Bridge control signals
    process (CLK)
    begin
        if CLK'event and CLK='1' then --CLK rising edge
                outpwm <= cdir;
        end if;
    end process;

end pwm_arch;
```

FIGURE 28.2 VHDL example for PWM output.

28.2 Field-Programmable Gate Array (FPGA) Background

28.2.1 FPGA Definition and Market

An FPGA is a chip that consists of many unconfigured logic gates. Unlike the fixed, vendor-defined functionality of an application-specific integrated circuit (ASIC) chip, an FPGA can be configured and reconfigured for different applications. FPGAs are used in applications where the cost of developing and fabricating an ASIC is prohibitive, or the hardware must be reconfigured after being placed into service. Because FPGAs can be used for implementation of custom algorithms in hardware, they offer benefits such as precise timing and synchronization, rapid decision making, and simultaneous execution of parallel tasks. Today, FPGAs appear in such devices as electronic instruments, consumer electronics, automobiles, aircraft, copy machines, and application-specific computer hardware.

The major manufacturers of FPGAs are Xilinx [3] and Altera [4] and, together, they comprise more than 80% of the FPGA market [5].

28.2.2 Programming Methods

Historically, programming FPGAs has been limited to engineers who have in-depth knowledge of VHDL or other low-level design tools, and requires overcoming a very steep learning curve.

Other FPGA design options are available. Celoxica [6] offers comprehensive C/C++ design tools for FPGAs. Mentor Graphics [7] creates FPGA code using C++.

AccelChip [8] offers MATLAB® programming capability for FPGAs including tools for floating-point to fixed-point conversion. Xilinx [3] provides a graphical programming option using Simulink® as part of their Xilinx System Generator for DSP.

28.2.3 LabVIEW FPGA Graphical Programming

28.2.3.1 LabVIEW FPGA Introduction

The NI LabVIEW FPGA Module extends the LabVIEW graphical development environment to reconfigurable FPGAs on various hardware platforms [9]. With the LabVIEW FPGA Module, the operation of the FPGA device as well as the connected analog and digital I/O can be configured by programming in LabVIEW. The graphical dataflow and parallelism inherent in LabVIEW are useful for intuitively programming control tasks, algorithms, signal processing, and logic.

Once the FPGA is targeted in the LabVIEW development environment, LabVIEW displays only the functions that can be implemented in the FPGA (Figure 28.3). The LabVIEW FPGA Module functions palette includes typical LabVIEWstructures and functions, such as while loops, for loops, case structures, and sequence structures, as well as a dedicated set of LabVIEW FPGA-specific functions for math, signal generation and analysis, simple control (including PID), comparison logic, array and cluster manipulation, occurrences, analog and digital I/O, and timing. A combination of these functions can be used to embed logic and intelligence onto the FPGA-based device.Math and control in FPGA are limited to integer-math operations. However, simple control blocks such as PID have been developed with pseudo-floating point capability.

The LabVIEW block diagram is implemented in hardware, which provides direct control over the reconfigurable hardware I/O. The I/O signals can be analyzed and manipulated in ways that are not possible with fixed I/O hardware. Figure 28.4 shows the block diagram for a simple rising-edge counter. Following the data flow makes it very easy to understand the behavior of this counter. A while loop

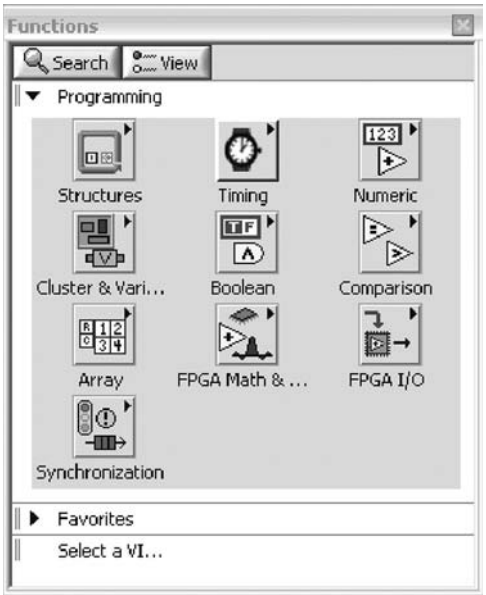


FIGURE 28.3 LabVIEW FPGA Functions Palette showing programming structures, timing, math, logic, control, synchronization, and I/O functions.

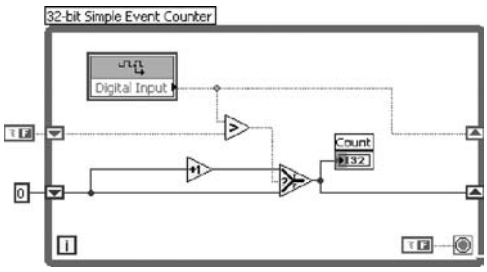


FIGURE 28.4 LabVIEW FPGA program for 32-bit counter.

structure is used to continuously read from a digital line and compare the line status from one iteration to the next. When the digital line changes state, the counter is incremented by one. The counter size is defined by the I32 data type of the counter indicator, making this a 32-bit counter.

The uniqueness of LabVIEW FPGA is thatmultiple, parallel counters can be created through additional while loops placed on the LabVIEW block diagram. These will consume completely different sections of the FPGA, resulting in true parallel process. In addition, the hardware can be reconfigured as needed. For example, the digital I/O lines connected to the FPGA can be apportioned as needed into any combination of encoder inputs, counter inputs, PWM outputs, or serial communication protocols. If different digital I/O is needed for a new application, the FPGA is simply reprogrammed with LabVIEW-FPGA to provide different I/O configuration.

The hardware configuration on the FPGA can be augmented with in-line signal processing, health monitoring, fault detection, logic, and feedback control. Each additional task consumes additional FPGA gates until the total number of gates on the FPGA device is consumed.

28.2.4 Development Flow

After creating the LabVIEW FPGA VI, the code must be compiled. Similar to other FPGA development tools, compile time for an FPGA VI can range fromminutes to several hours, depending on the complexity of the code and the specifications of the development system. To maximize development productivity, a bit-accurate emulation mode can be used, so the logic of the design can be verified before initiating the compile process. When targeting the FPGA Device Emulator, LabVIEW accesses I/O from the device and executes the VI logic on the Windows development computer. In this mode, it is possible to use the same debugging tools available in LabVIEW for Windows, such as execution highlighting, probes, and breakpoints.

Once the LabVIEW FPGA code is compiled, a LabVIEW host VI can be created to integrate the hardware into the rest of the test and control system. Figure 28.5 illustrates the development process for creating an FPGA application. The host VI uses controls and indicators on the FPGA VI front panel to transfer data between the FPGA on the RIO device and the host processing engine. These front panel objects are represented as data registers within the FPGA. The host computer can be either a PC or PXI controller running Windows or a PC, PXI controller, Compact Vision System, or CompactRIO controller running a real-time operating system (RTOS).

28.2.5 Combined FPGA/MicroProcessor Architectures

National Instruments provides multiple hardware platforms with FPGA technology. Figure 28.6 shows the PXI/cPCI Industrial Computer platform, with plug-in FPGA boards as an option, as well as the CompactRIO programmable automation controller with an FPGA on the backplane of the chassis. NI also provides programmable plug-in FPGA boards for desktop computers. An additional hardware platform with a programmable FPGA is the NI Compact Vision System with distributed machine vision and control capability.

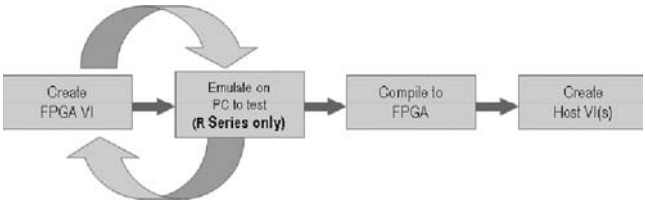


FIGURE 28.5 Application development flow showing FPGA compilation step.

FIGURE 28.6 PXI/cPCI Industrial Computer and NI CompactRIO automation controller.

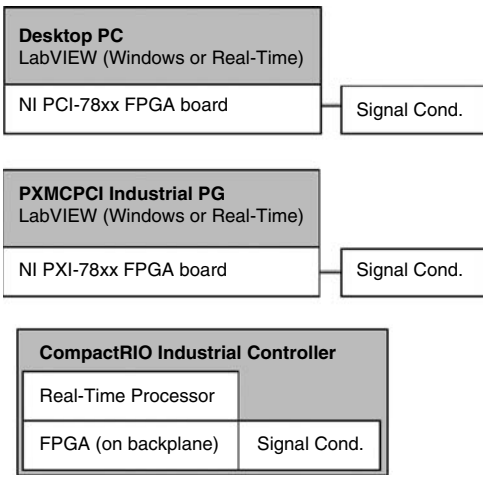


FIGURE 28.7 Hardware architecture diagram for FPGA/processor platforms.

The shared hardware architecture among the platforms is illustrated in Figure 28.7. The desktop PC and the PXI/cPCI Industrial PC can run Windows or LabVIEW Real-Time on the main floating-point processor. LabVIEW Real-Time allows LabVIEW diagrams to run on top of a commercial real-time operating system for multiple types of targets. In both the desktop and the PXI system, the plug-in FPGA board is connected to the processor via the PCI bus. Plug-in FPGA boards are available with both balanced analog I/O and multiple digital lines.

The CompactRIO architecture shown in Figure 28.7 is similar to that of the desktop and the PXI system with FPGA technology. The floating-point processor in the CompactRIO controller is tied to the FPGA on the backplane via an internal PCI bus. The FPGA is, in turn, connected to the analog and digital I/O modules in a star topology. Analog modules contain both conversion circuitry and signal conditioning while digital modules have signal conditioning alone.

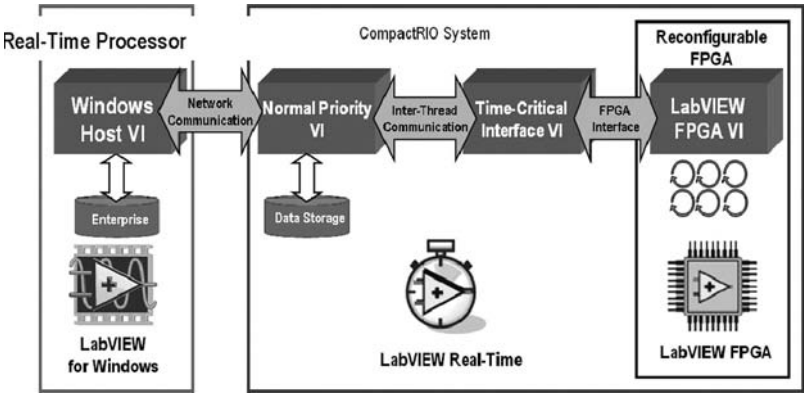


FIGURE 28.8 Software application architecture diagram for FPGA/processor platforms.

Because of the similar hardware architecture, a common software application architecture can be used for all three platforms. This software architecture is shown in Figure 28.8 and includes a host VI (program) running on a Windows PC, a real-time VI running on the processor, and an FPGA VI running completely in hardware on the FPGA.

With this software and hardware architecture, it is possible to quickly develop various hierarchical control strategies. Some lower level control loops can take place completely in the FPGA. Higher level control loops can be written for the floating-point processor in the system. In addition, various tasks such as custom startup and shutdown behavior, in-line signal processing, and custom timing, synchronization, and digital communication can be implemented in the FPGA. Time-critical parallel tasks such as health monitoring and fault detection can be developed on the FPGA as well.

While processors run computations in sequence, FPGAs can run computations in parallel. New processor architectures involving multiple cores are being released, but the FPGA is still the best choice for parallel computation. If multiple control loops or tasks are implemented in a processor, then the overall loop rate will decrease. Multiple control loops and tasks that are implemented in an FPGA can execute independently so that extremely high loop rates can be achieved. The limiting factor for FPGAs is the space available or the number of gates.

28.2.6 Integration of FPGA Graphical Programming with Simulation and Control

A simple analog input and analog output operation written with LabVIEWFPGA is shown in Figure 28.9. The analog I/O will run at the maximum rate allowed by the FPGA since a timing VI has not been added to the program. The timing is then limited only by the conversion rate for I/O on the FPGA-based device.

Figure 28.10 shows the front panel for this VI when the analog output is wrapped back to the analog input. The user can adjust the slider for the analog output on the front panel and see the result in the graph of the analog input.

A simple control program can be developed for the real-time processor using the LabVIEW Simulation Module. Figure 28.11 shows the simulation loop with a signal generator VI writing a sine wave to the analog output. The analog input is read back through the FPGA and communicated to the processor.

Many other dynamic system modeling and control blocks are available in LabVIEW Simulation Module including linear continuous and discrete-time models, as well as both graphical and textual blocks for nonlinear systems.

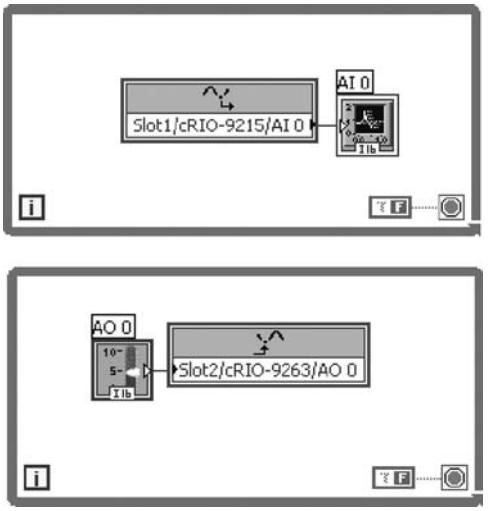


FIGURE 28.9 Simple analog input and output VI written in LabVIEW FPGA for single-point I/O.

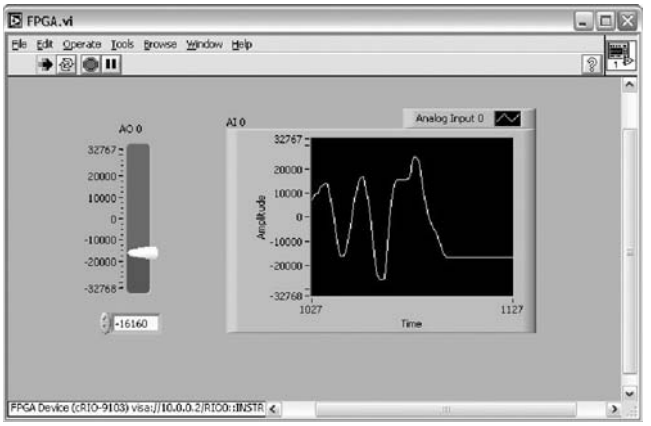


FIGURE 28.10 Front panel of analog input and output VI written in LabVIEW FPGA.

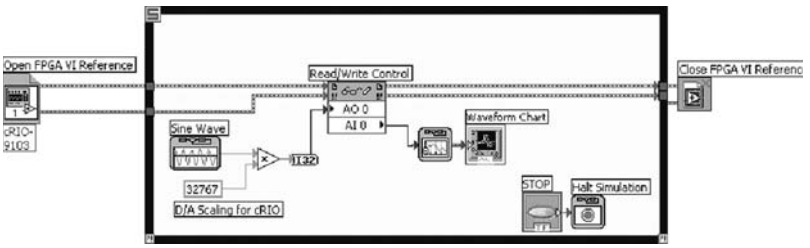


FIGURE 28.11 Simple LabVIEW Simulation Module VI that communicates with LabVIEW FPGA VI through an interface node.

28.3 Applications

28.3.1 High-Speed Control on a FPGA for Micro-Electromechanical Systems Device

Using LabVIEW FPGA and the NI PXI-7831R module, Kurfess and Degertekin implemented parallel, extremely high speed control (120 kHz) of an array of micro-electromechanical systems (MEMS) devices [10]. MEMS devices are produced in large quantities on a single silicon wafer, and, even when packaged or built into another device, they are often spatially arranged in arrays. Kurfess and Degertekin developed a novel measurement system that can test multiple MEMS devices in an array simultaneously rather than stepping and scanning a single measurement device over the MEMS array. The new measurement system is itself an array of MEMS microinterferometers. Each microinterferometer in the array is a position-sensing grating interferometer (μPSGI) capable of measuring out-of-plane distances with subnanometer accuracy at frequencies more than 20 MHz. The microinterferometer is 1000 times smaller than typical, bench-sized laser interferometers. Initial results demonstrate mapping of microscale membranes vibrating in the 2 MHz range (Figure 28.12).

Kurfess and Degertekin implemented the microinterferometer in both single and array units using the LabVIEW FPGA Module and NI reconfigurable I/O hardware, which makes the transition from monitoring and controlling a single interferometer to an array of interferometers a simple task. They used the NI PXI-7831R reconfigurable I/O module to provide analog I/O along with high-speed PID control. They applied the control to a deformable grating within the interferometer sensor to minimize the effects of sample misalignment and external disturbances and improve the sensitivity of the microinterferometer during operation. They used the pseudo–floating-point PID example in the LabVIEW FPGA Module as a basis for the control implementation. With the high-speed hardware timing capabilities of the LabVIEW FPGA Module, Kurfess and Degertekin easily achieved the required 120 kHz bandwidth for the control loop.

In this application, the maximum bandwidth was determined by the conversion speed of the A/D and D/A circuitry on the PXI-7831R module. Higher bandwidth control can be achieved if the digital lines on the plug-in FPGA board are used for both sensors and actuators. An example would be encoders or tachometers for sensors and PWM commands to amplifiers or drives for actuators. Also, the CompactRIO system includes options for high bandwidth analog-to-digital and digital-to-analog conversion circuitry.

28.3.2 Single-Point I/O on FPGA for Haptics

Kopp and O'Malley [11] used LabVIEW Real-Time and LabVIEW FPGA for control and high-speed single-point I/O in a haptics experiment focused on creating stiff "virtual walls." A limiting factor for

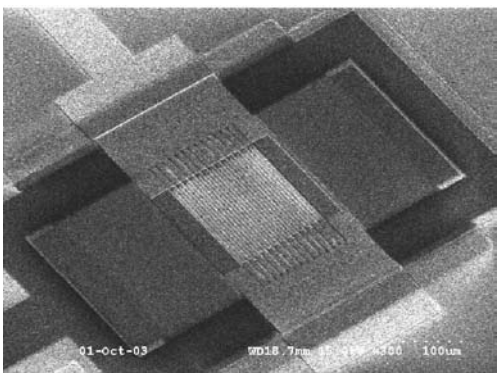


FIGURE 28.12 SEM image of MEMS microinterferometer.

stiffness in virtual walls is the sampling rate of the control system for the haptic device. If the desired stiffness is too high in relation to the sampling rate, then the virtual wall ceases to be passive and exhibits active behavior in the form of vibrations and, at too high of a stiffness, instability. The device used in the experiment was the Phantom Premium 1.0 haptic device from SensAble Technologies [12]. Using LabVIEW FPGA and a PXI-7831R FPGA board, the researchers were able to achieve a sampling rate of 20 kHz. The Windows/GHOST software and hardware platform that shipped with the Phantom device had a maximum sampling rate of 1 kHz. The increased sampling rate for the FPGA-based system resulted in a maximum wall stiffness of 500 N/m compared to a maximum wall stiffness of 25 N/m for the Windows/GHOST system.

28.3.3 Phase-Locked Loop for Scanning Probe Microscope

Rychen and Vancura [13] developed a compact, high-speed control system for a scanning probe microscope (SPM) using LabVIEW FPGA, LabVIEW Real-Time, and a PXI system. The 1.26 GHz real-time processor ran several algorithms such as raster scan generators, the tip-sample distance controller, and auxiliary feedback controllers. A digital phase-locked loop (PLL) was implemented with LabVIEWFPGA resulting in a demodulation bandwidth of more than 10 kHz, using the analog I/O lines to interface to the external components. With a redesign involving the use of digital I/O lines, 24 MS/s is possible with a frequency resolution of 0.1 μHz. The fully digital case will yield high-phase accuracy compared to the analog circuitry. In addition, the demodulation bandwidth may be adjusted into the millihertz range for the highest frequency resolution.

28.3.4 Model-Free Adaptive Control for FPGA

CyboSoft [14] has developed a model-free adaptive (MFA) control toolset for LabVIEW. Used in conjunction with NI SoftMotion on CompactRIO, one option in the toolset provides high-speed positioning control using LabVIEW FPGA with a minimum sampling period of 25 μs. The toolset has been successfully demonstrated with high precision piezoelectric stages with significant load changes.

28.3.5 Motorcycle Engine Control Prototyping

Dase [15] from Drivven Inc. [16] used LabVIEW FPGA, CompactRIO, and custom I/O modules to build an engine control prototyping system. Engine control is a challenging application involving loop times on the order of milliseconds and precision fuel and spark timing on the order of microseconds. Dase was able to completely replace the ECU on a 2004 Yamaha YZF-R6 motorcycle using the CompactRIO system such that experienced riders could not identify significant differences between the factory ECU control and the prototype control.

28.4 Additional Tools for FPGA Development

28.4.1 SoftMotion Position Control Loop on FPGA

The NI SoftMotion Development Module, shown in Figure 28.13, includes enhanced real-time and FPGA algorithms for motion control and built-in inputs for a variety of feedback devices including quadrature encoders and sine cosine encoders. Table 28.1 shows the servo update rates that can be obtained using NI SoftMotion on a variety of hardware targets including FPGA targets.

28.4.2 Digital Filter Design Toolkit

The NI Digital Filter Design Toolkit provides functions and interactive tools for design, analysis, and implementation of digital filters within LabVIEW FPGA. The toolkit includes interactive design methods for both classical filter design and filter design based on pole-zero placement (Figure 28.14). A control

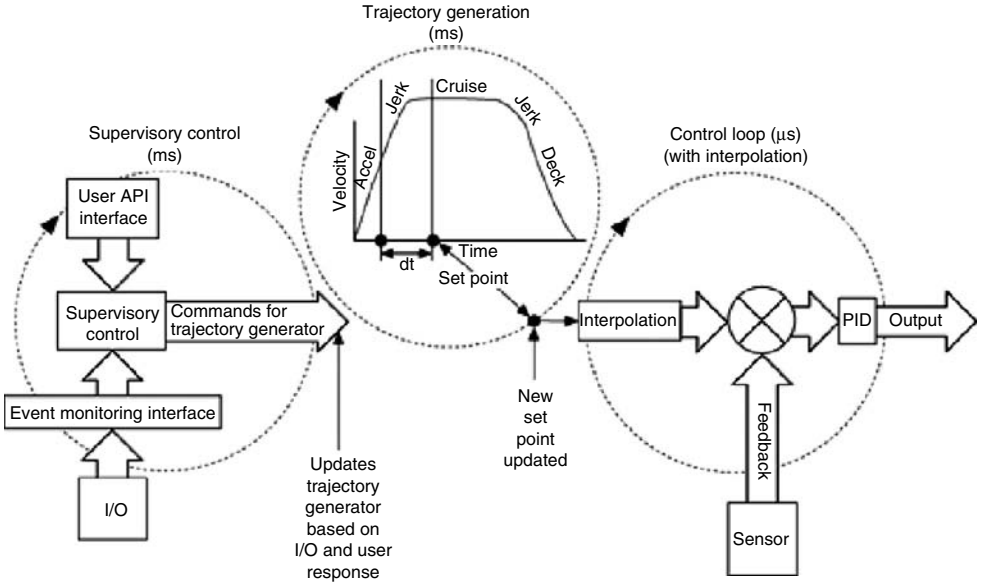


FIGURE 28.13 Architecture of NI SoftMotion Development Module.

TABLE 28.1 Servo Update Rates for NI Softmotion on Various Platforms

Hardware Platform	Servo Update Rate
CompactRIO	5 μS
PCI/PXI FPGA	5 μS
PCI/PXI Motion Control	62.5 μS
PCI/PXI M Series DAQ	1 ms
Compact FieldPoint	5 ms

law that can be represented as a discrete-time transfer function can be implemented on the FPGA using LabVIEW FPGA code generated from the Digital Filter Design Toolkit.

28.4.3 State Diagram Editor on FPGA

The LabVIEW State Diagram Toolkit can be used to help build state machines on an FPGA. The toolkit includes a state diagram editor for drawing the logic that defines an application. An example of this is shown in Figure 28.15. As states and transitions are added to the state machine diagram, LabVIEW generates code to match each change. The logic for the statemachines is represented in the LabVIEW code by a series of while loops and case statements. The toolkit can also insert comments in the LabVIEW block diagram to indicate where additional code can be added. The entire LabVIEW block diagram can then be implemented on an FPGA using the LabVIEW FPGA module.

28.5 Conclusion

Graphical programming for FPGAs can be used by control and mechatronic system designers and researchers to quickly develop robust and high-performance applications. The combination of FPGAs with floating-point processors allows engineers to intelligently distribute computation, control, and signal

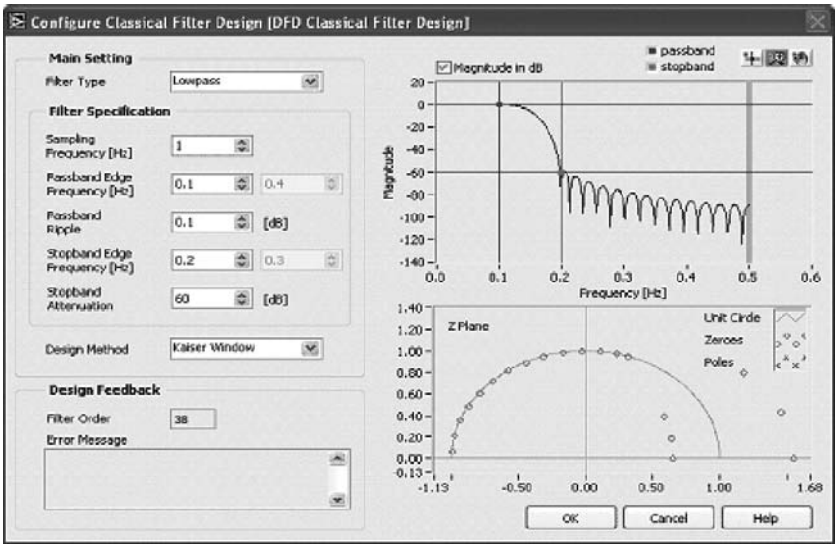


FIGURE 28.14 Interactive filter design using LabVIEW Digital Filter Design Toolkit.

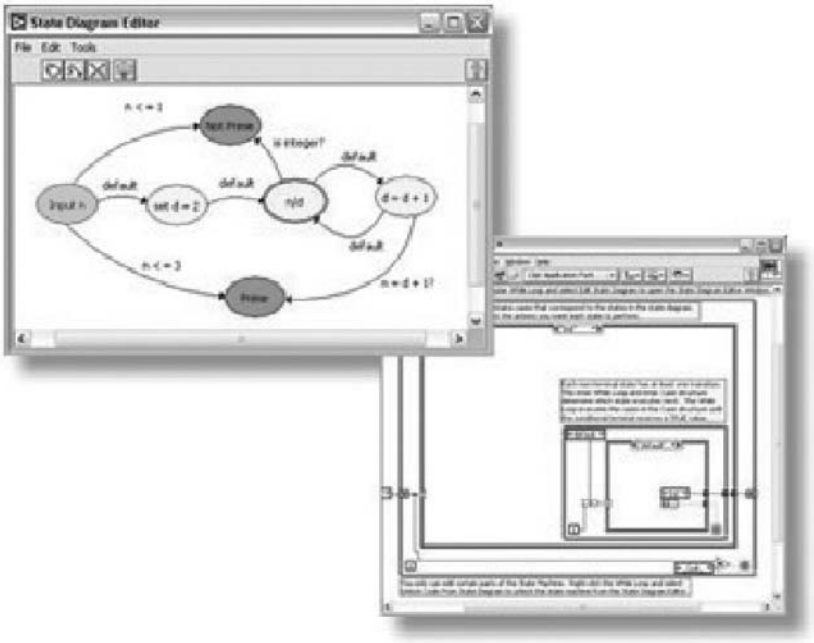


FIGURE 28.15 Architecture of NI SoftMotion Development Module.

processing functions. LabVIEW FPGA is a comprehensive tool for graphical programming FPGAs and includes functions for timing, logic, synchronization, high-speed control, analog, and digital I/O. The NI SoftMotion Module provides motion-specific functionality for combined real-time processor/FPGA systems. The NI Digital Filter Design Toolkit can be used in conjunction with LabVIEW FPGA to quickly prototype and implement digital filters on an FPGA.

References

1. C.Maxfield, *The DesignWarrior's Guide to FPGA's – Devices, Tools and FLows*, Oxford, UK: Newnes Press, 2004, pp. 153–172.
2. T. Salcudean, Project Integrated Program, Department of Electrical and Computer Engineering, University of British Columbia, http://www.ece.ubc.ca/~tims.
3. Xilinx, Inc., San Jose, CA, http://www.xilinx.com
4. Altera Corp., San Jose, CA, http://www.altera.com
5. M. Santarini (November 2005) "Xilinx keeps pace through innovation," *Movers and Shakers 2005*, 6th Edn., Reed Business Information, Available: http://www.reed-electronics.com/moversandshakers/article/CA6277496.html
6. Celoxica Ltd., Oxfordshire, UK, http://www.celoxica.com
7. Mentor Graphics Corp.,Wilsonville, OR, http://www.mentor.com
8. AccelChip, Milpitas, CA, http://www.accelchip.com
9. National Instruments Corp., FPGA portal, http://www.ni.com/fpga
10. B. Kim, M. Schmittdiel, F. L. Degertekin, and T. Kurfess, "Deformable diffraction grating for scanning micro interferometer arrays," *SPIE Photonic West Conference*, 2004.
11. E. Kopp and M. O'Malley, "Improving haptic simulations using LabVIEW FPGA," *Proceedings of the 23rd Annual Houston Conference on Biomedical Engineering Research*, February 9–10, 2006.
12. Sensable Technologies,Woburn, MA, http://www.sensable.com
13. J. Rychen and T. Vancura, Nanonis Gmbh, Switzerland, "Building high-precision control systems with LabVIEW7 Express," www.ni.com/pdf/csma/us/jorg_rychen_labviewsevenexpress.pdf
14. CyboSoft, General Cybernation Group, Inc., Rancho Cordova, CA, www.cybosoft.com
15. C. Dase, "Drivven Prototypes FPGA-Based Engine Control System Using NI CompactRIO," http://sine.ni.com/csol/cds/item/vw/p/id/538/nid/124100
16. Drivven Inc., San Antonio, TX, http://www.drivven.com

III

Software and Data Acquisition

29

Introduction to Data Acquisition

Craig Anderson
National Instruments, Inc.

29.1 Introduction

Data acquisition is the gathering of signals from real-world measurement sources and the digitizing of those signals for storage, analysis, and presentation on a personal computer (PC). Light, temperature, pressure, and torque are a few of the many different types of signals that can interface to a data acquisition system. In addition to acquiring data, data acquisition systems also are used to generate electrical signals.

These signals can either intelligently control mechanical systems or provide a stimulus so that the data acquisition system can measure the response. A data acquisition system provides a way to empirically test designs, theories, and real-world systems for validation or research. Figure 29.1 illustrates a typical computer-based data acquisition module.

The design and the production of amodern car, for instance, rely heavily on data acquisition. Engineers will first use data acquisition to test the design of the car's components. The frame can be monitored for mechanical stress, wind noise, and durability. The vibration and temperature of the engine can be acquired to evaluate the design quality. The researchers and engineers can then use this data to optimize the design of the first prototype of the car. The prototype can then bemonitored under many different conditions on a test track while information is collected through data acquisition. After required iterations of design change and data acquisition, the car is ready for production. Data acquisition devices can monitor the machines that assemble the car, and they can ensure that the assembled car meets the necessary specifications.

At first, data acquisition devices stood alone and were manually controlled by an operator. When the PC emerged, data acquisition devices and instruments could be connected to the computer through a serial port, parallel port, or some custom interface. A computer program could control the device automatically and retrieve data from the device for storage, analysis, or presentation. Now, instruments

FIGURE 29.1 Examples of typical, computer-based data acquisition devices. Plug-in devices are also commonly available for USB, PCI, and PCI Express buses.

and data acquisition devices can be integrated into a computer through high-speed communication links such as PCI, PCI Express, Ethernet, IEEE 1394, or USB busses for tighter integration between the power and .exibility of the computer and the instrument or device.

Modern data acquisition systems are composed of transducers and sensors to measure a physical phenomenon, the resulting electrical signals, hardware to prepare the signals for measurement, measurement hardware, and data acquisition software.

29.2 Transducers and Sensors

Data acquisition begins with the physical phenomenon to be measured. This physical phenomenon could be the temperature of a room, the intensity of a light source, the pressure inside a chamber, the force applied to an object, or many other things. An effective data acquisition system can measure all these different phenomena.

A transducer is a device that converts a physical phenomenon into a measurable electrical signal, such as voltage or current. The ability of a data acquisition systemto measure different phenomena depends on the transducers that convert the physical phenomena into signals measurable by the data acquisition hardware. Transducers are synonymous with sensors in data acquisition systems. There are specific transducers for many different applications, such as measuring temperature, pressure, or fluid flow.

A common example of a transducer is a thermocouple. A thermocouple uses the material properties of dissimilar metals to convert a temperature into a voltage. As the temperature increases, the voltage produced by the thermocouple increases. A software program can then convert the voltage reading back into a temperature for analysis, presentation, and data logging. Many sensors produce currents instead of voltages. A current is often advantageous because the signal will not be corrupted by small amounts of resistance in the wires connecting the transducer to the data acquisition device. A disadvantage of current-producing transducers, though, is that most data acquisition devices measure voltage, not current.

Generally, the data acquisition devices that can measure current use a very small resistance of a known value to convert the known current into a readable voltage. Ultimately, the device isthen still acquiring a voltage.

29.3 Signals

The appropriate transducer converts the physical phenomena into measurable signals. However, different signals need to be measured in different ways. For this reason, it is important to understand the different types of signals and their corresponding attributes. Signals can be categorized into two groups; analog and digital.

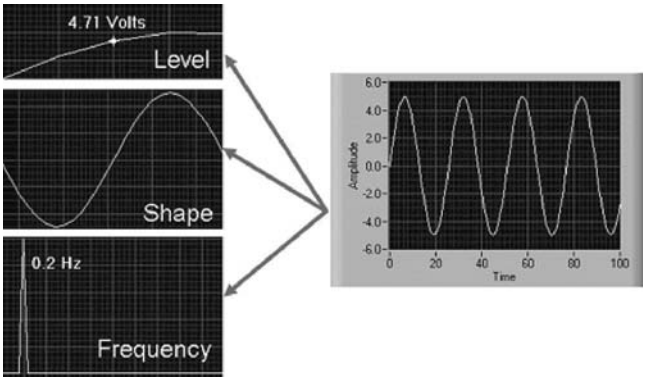


FIGURE 29.2 Primary characteristics of an analog signal.

29.3.1 Analog Signals

An analog signal can be at any value with respect to time. A few examples of analog signals include voltage, temperature, pressure, sound, and load. The three primary characteristics of an analog signal are level, shape, and frequency (Figure 29.2).

29.3.1.1 Level

Because analog signals can take on any value, the level gives vital information about the measured analog signal. The intensity of a light source, the temperature in a room, and the pressure inside a chamber are all examples that demonstrate the importance of a signal's level. When measuring a signal's level, the signal generally doesnot change quickly with respect to time. The accuracy of the measurement, however, is very important. A data acquisition system that yields maximum accuracy should be chosen to aid in analog level measurements.

29.3.1.2 Shape

Some signals are named after their speci.c shape—sine, square, sawtooth, and triangle. The shape of an analog signal can be as important as the level, because measuring the shape of an analog signal allows further analysis of the signal, including peak values, dc values, and slope. Signals where shape is of interest generally change rapidly with respect to time and therefore require faster sampling of the analog signal than is required for a level measurement. The analysis of heartbeats, video signals, sounds, vibrations, and circuit responses are some applications involving shape measurements.

29.3.1.3 Frequency

Analog signals can be categorized by their frequency. Unlike the level or shape of the signal, frequency cannot be directly measured. The signal must be analyzed using software to determine the frequency information. Thisanalys isisus ually done using an algorithm known asthe Fourier transform.

When frequency is the most important piece of information, it is important to consider both accuracy and acquisition speed. Although the acquisition speed for acquiring the frequency of a signal is less than the speed required for obtaining the shape of a signal, the signal must still be acquired fast enough so that the pertinent information is not lost while the analog signal is being acquired. The Nyquist sampling theorem states that an analog signalmust be sampled two times faster than the highest frequency component in the signal source in order to preserve frequency information. To preserve shape and frequency information, sampling should be performed 10 times faster than the highest frequency component in the signal source. Speech analysis, telecommunication, and earthquake analysis are some examples of common applications where the frequency of the signal must be known.

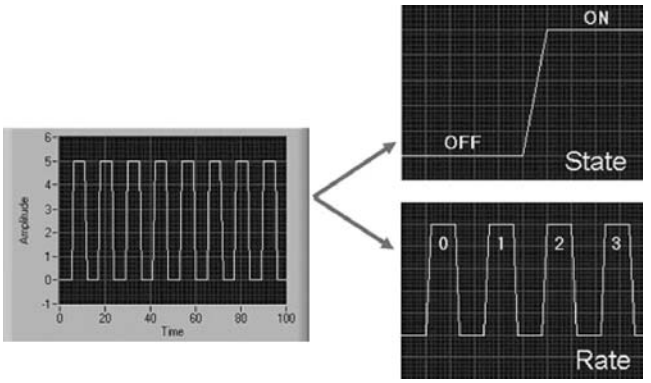


FIGURE 29.3 Primary characteristics of a digital signal.

29.3.2 Digital Signals

A digital signal cannot take on any value with respect to time. Instead, a digital signal has two possible levels—high and low. Digital signals generally conform to certain speci.cations that de.ne the characteristics of the signal. Digital signals are commonly referred to as transistor–transistor logic (TTL). TTL speci.cationsdictat e that a digital signal islo w when the level fallsw ithin 0–0.8 V, and high when it falls between 2 and 5 V. The useful information that can be measured from a digital signal includes the state and the rate, ass een in Figure 29.3.

29.3.2.1 State

Digital signals cannot take on any value with respect to time. The state of a digital signal is essentially the level of the signal—on or off, high or low.Monitoring the state of a switch—open or closed—is a common application showing the importance of knowing the state of a digital signal. The most commonly used digital levels are TTL and TTL-compatible CMOS. These are both very common 5-V standards for digital hardware.

Digital transfer rates to and from data acquisition hardware vary from unstrobed to high speed. Unstrobed digital input and output involves setting digital lines and monitoring states by software command. Thisformof digital input and output isals o known ass tatic or immediate digital I/O.The maximum speed of an unstrobed I/O is highly dependent on the computer hardware, the operating system, and the application program. Pattern digital I/O referst o inputsand outputsof digital patternsunder the control of a clock signal. The speed at which the data can be sent or received depends on the amount of data, the characteristics of the data acquisition hardware, and the computer speed.

29.3.2.2 Rate

The rate of a digital signal de.nes how it changes state with respect to time. One example of measuring a digital signal's rate is determining how fast a motor shaft spins. Unlike frequency, the rate of a digital signal measures how often a portion of the signal occurs. A software algorithm is not required to determine the rate of a signal. Instead, a digital signal's rate is typically calculated using counter/timers. Counter/timers are capable of measuring or producing time-critical digital pulses. These pulses, such asthe digital input and output, are generally TTL or TTL-compatible CMOS. These components are used for measuring or producing a number of time-critical signals, including event counting, pulse train generation, frequency-shift keying, and quadrature encoder monitoring. The two main characteristics of a counter/timer are counter size and maximum source frequency. The counter size is generally represented in bitsand determinesho w high a counter can count. For instance, a 32-bit counter can count $2^{32} - 1 = 4{,}294{,}967{,}295$ eventsbefor e it returnsthe count value back to zero. The maximum source frequency

represents the speed of the fastest signal the counter can count. An 80-MHz counter can count events that are asfas t as12.5 nsapar t. An "event" isactually the rising or falling edge of a digital signal.

29.4 Signal Conditioning

No real situation will have perfect signals, be completely free of noise, or be guaranteed to be electrically safe. Most real signals require some sort of preparation (conditioning) before being gathered by a data acquisition system. Signal conditioning hardware provides a method to remove—as much as possible—unwanted componentsof a digital or analog signal.

Measurement hardware, particularly for high-frequency analog signals, is usually equipped with an antialiasing filter. Thisisa low-pass.lt er that blocks frequencies above the desired frequency range and increasesthe measurement accuracy. Digital and counter/timer lines are also commonly fitted with filters that remove signal spikes that could otherwise be mistakenly counted as a rising or falling edge. Isolation is another type of signal conditioning that separates the measurement hardware circuitry from the signal being measured. Thisisdone to remove large differencesin electric potential between the measurement hardware and the signal, and it protectsoper atorsand measurement hardware from damage, given a large surge in voltage or current. Other types of signal conditioning include ampli.cation, attenuation, multiplexing, excitation, and cold-junction compensation.

29.5 Data Acquisition Hardware

The heart of a data acquisition device is a digital-to-analog converter (DAC), an analog-to-digital converter (ADC), or some combinationsof the two. An ADC hasa .nite list of binary valuesthat represent analog voltage ranges. The ADC's purpose is to select a value from this list that is closest to an actual voltage at a specified time. The value is then transferred in binary format to a computer. Alternatively, a DAC can produce an analog voltage froma list of binary values. The voltage generated by a basicDAC stays the same until it receivesanother value from the computer. In order to acquire and produce analog waveforms, the DAC and ADC must activate at precise intervals. Consequently, measurement hardware has timing circuitry to produce a pulse train of a constant frequency to control the ADC and DAC.

The data that ist ransferred from the ADC and to the DAC travelst o the computer over a bus. A bus isa group of electrical conductorsthat transfer information inside a computer. Some common examples of a busar e PCI and USB. The buscan carry both control information and binary measurement data to and from measurement hardware. One of the most important considerations in selecting a bus is bus transfer rate, usually expressed in megabytes per second (Mb/s). A single analog value could require less than 1 byte or asm uch as4 bytes, depending on the type of measurement hardware. The busiss hared among multiple devices, so data acquisition devices often have onboard memory to serve as a holding place for data when the bus is not available. In very fast data acquisition routines, the memory can hold all the data, and at the end of the acquisition, all the data can be transferred to the computer for processing.

When data isac quired at high speedson multiple channels, it isoft en important to understand the phase relationship from one signal to the next. If the signals need to be generated or acquired on a single data acquisition device, simultaneously sampling data acquisition hardware provides a dedicated ADC or DAC and a shared clock for all channels. If the signals are generated or acquired on multiple data acquisition devices, there are a number of ways to synchronize the systems and preserve relative phase relationships. One way is to share the ADC and DAC clock between the data acquisition devices. The real-time system integration (RTSI) bus can connect multiple devices together to share timing circuitry among multiple devices. Phase-lock looping (PLL) is a more sophisticated synchronization method. A reference signal is supplied to all the data acquisition devices, and the internal clocks stay in phase with the reference signal. Consequently, the phase relationship can be reserved even if different measurement

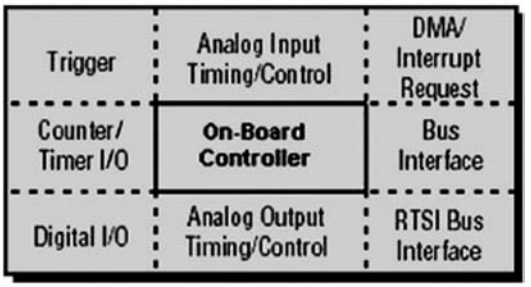


FIGURE 29.4 The components of a typical data acquisition system.

hardware is using different sampling or update speeds. Figure 29.4 is a diagram showing the components of a typical data acquisition system.

The difference between an actual analog voltage and the closest voltage from the list of binary values is called the quantization error. In a perfect digitizing measurement system that is free of noise, the quantization errorwould solely explain any difference between the actual voltage and the measured voltage. No measurement hardware and no environment, however, are perfect. The accuracy of an instrument describes the amount of uncertainty when considering quantization error, unavoidable system noise, and hardware imperfections. Accuracy is sometimes confused with precision. Precision refers to the amount of deviation inmultiple measurements connected to a constant and level signal source. Even if an instrument is precise, it could still be inaccurate if the readings were consistent but signi.cantly different than the actual value of the signal.

The accuracy of a data acquisition system can change with temperature, time, and usage. Data acquisition hardware can store onboard correction constants for offset and gain errors. An offset error is a constant difference between the measured and actual voltage, regardless of the voltage level. A gain error increases linearly as the measured voltage increases. Some data acquisition hardware also includes an accurate voltage source on board that can be periodically used as a reference to correct the gain and offset error parameters.

29.6 Data Acquisition Software

The final piece of a data acquisition systemto understand is the software. Software transforms the PC and the data acquisition hardware into a complete data acquisition, analysis, and presentation tool. Without software to control or drive the hardware, the data acquisition device will not function properly. The driver software is a set of commands that a programmer can incorporate into a program. The driver software is usually supplied by the manufacturer of the hardware and can be used in a variety of programming languages. Driver software is the layer of software that allows easy communication with the hardware. It forms the middle layer between the application software and the hardware. Driver software also prevents a programmer from having to do register-level programming or complicated commands in order to access the hardware functions.

Application software adds analysis and presentation capabilities to the driver software. To choose the right application software, evaluate the complexity of the application, the availability of con.gurationbased software that fits the application, and the amount of time available to develop the application. If the application isc omplex or there isno existing program, use a development environment. A programmer can use a programming language and an application development environment to build an application from the driver software such as the one shown in Figure 29.5, or use a con.guration-based programwith preset functionality. The application is then ready for an end user to easily control and acquire data from the hardware—a custom instrument built specifically for the user's needs.

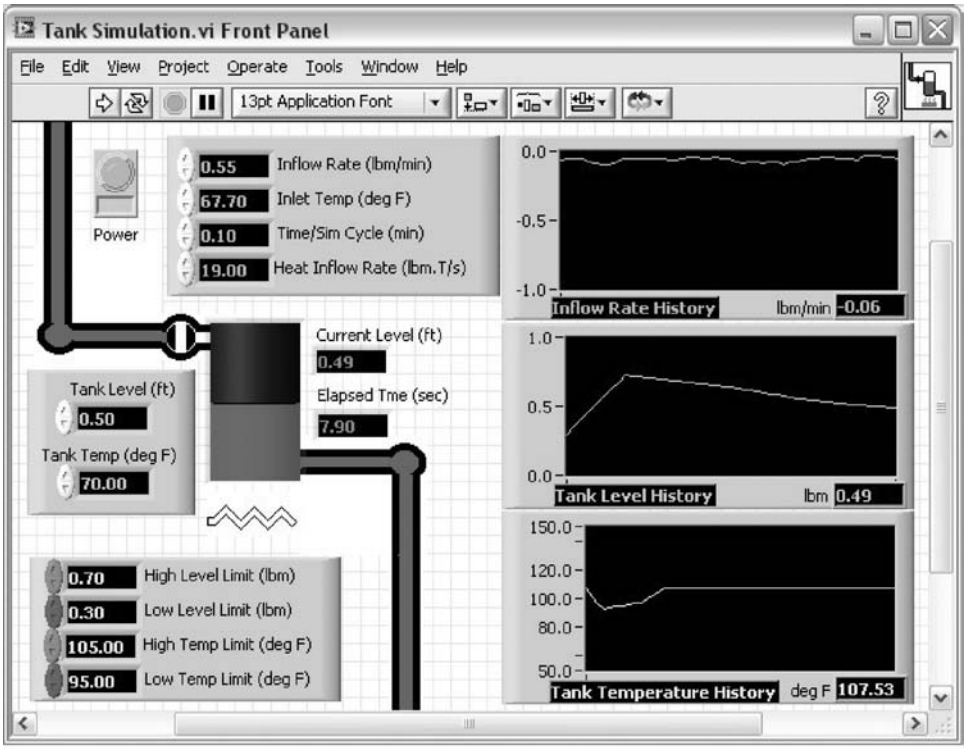


FIGURE 29.5 An example of an application built from LabVIEWsoftware.

29.7 Summary

Data acquisition is the process of collecting signals fromreal-world measurement sources and the digitizing of those signals for storage, analysis, and presentation. Data acquisition systems can also be used to generate signals that either intelligently control mechanical systems or provide a stimulus so that the data acquisition system can measure the response. Initially, data acquisition devices were stand-alone and controlled manually by an operator. Today, data acquisition devices and instruments are connected to computers and are composed of transducers and sensors for measuring physical phenomena, hardware to prepare the signals for measurement, measurement hardware, and data acquisition software.

29.8 Related Information

More information about PC-based data acquisition systems is available from National Instruments in the form of white papers, application notes, customer solutions, and product information. Visit http://www.ni.com and search for "data acquisition" to view available information.

30

Measurement Techniques: Sensors and Transducers

Cecil Harrison
University of Southern Mississippi

30.1 Introduction

An automatic control system is said to be *error actuated* because the **forward path** components (*comparator, controller, actuator*, and *plant* or *process*) respond to the error signal (Figure 30.1). The error signal is developed by comparing the measured value of the **controlled output** to some **reference input**, and so the accuracy and precision of the controlled output are largely dependent on the accuracy and precision with which the controlled output is measured. It follows then that measurement of the controlled output, accomplished by a system component called the **transducer**, is arguably the single most important function in an automatic control system.

A transducer senses the magnitude or intensity of the controlled output and produces a proportional signal in an energy form suitable for transmission along the feedback path to the comparator. [The term proportional is used loosely here because the output of the transducer may not always be directly proportional to the controlled output; that is, the transducer may not be a linear component. In linear systems, if the output of the transducer (the measurement) is not linear, it is linearized by the signal conditioner.] The element of the transducer which senses the controlled output is called the *sensor*; the remaining elements of a transducer serve to convert the sensor output to the energy form required by the **feedback path**. Possible configurations of the feedback path include:

- Mechanical linkage
- Fluid power (pneumatic or hydraulic)
- Electrical, including optical coupling, RF propagation, magnetic coupling, or acoustic propagation

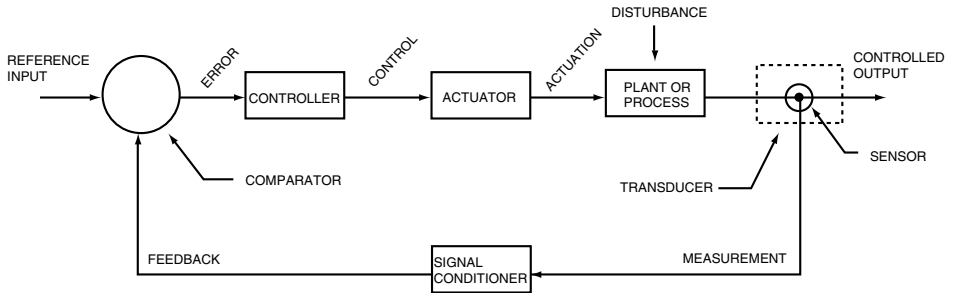


FIGURE 30.1 Functional block diagram of a canonical (standard) automatic control system.

Electrical signals suitable for representing measurement results include:

- DC voltage or current amplitude
- AC voltage or current amplitude, frequency, or phase (CW modulated)
- Voltage or current pulses (digital)

In some cases, representation may change (e.g., from a DC amplitude to digital pulses) along the feedback path.

The remainder of this discussion pertains to a large number of automatic control systems in which the feedback signal is electrical and the feedback path consists of wire or cable connections between the feedback path components. The transducers considered hereafter sense the controlled output and produce an electrical signal representative of the magnitude, intensity, or direction of the controlled output.

The **signal conditioner** accepts the electrical output of the transducer and transmits the signal to the comparator in a form compatible with the reference input. The functions of the signal conditioner include:

- Amplification/attenuation (scaling)
- Isolation
- Sampling
- Noise elimination
- Linearization
- Span and reference shifting
- Mathematical manipulation (e.g., differentiation, division, integration, multiplication, root finding, squaring, subtraction, or summation)
- Signal conversion (e.g., DC–AC, AC–DC, frequency–voltage, voltage–frequency, digital–analog, analog–digital, etc.)
- Buffering
- Digitizing
- Filtering
- Impedance matching
- Wave shaping
- Phase shifting

In cases in which part or all of the required signal conditioning is accomplished within the transducer, the transducer output may be connected directly to the comparator. [Connection of the transducer output directly to the comparator should not be confused with unity feedback. Unity feedback occurs when the cascaded components of the feedback path (transducer and signal conditioner) have a combined transfer function equal to 1 (unity).] In a digital control system, many of the signal conditioning functions listed here can also be accomplished by software.

Transducers are usually considered in two groups:

- **Motion** and **force transducers**, which are mainly associated with **servomechanisms**
- **Process transducers**, which are mainly associated with **process control** systems

As will be seen, most process transducers incorporate some sort of motion transducer.

30.2 Motion and Force Transducers

This section discusses those transducers used in systems that control motion (i.e., *displacement, velocity,* and *acceleration*). Force is closely associated with motion, because motion is the result of unbalanced forces, and so force transducers are discussed concurrently. The discussion is limited to those transducers that measure *rectilinear* motion (straight line motion within a stationary frame of reference) or *angular* motion (circular motion about a fixed axis). Rectilinear motion is sometimes called *linear* motion, but this leads to confusion in situations where the motion, though along a straight line, really represents a mathematically nonlinear response to input forces. Angular motion is also called *rotation* or *rotary motion* without ambiguity.

The primary theoretical basis for motion transducers is found in rigid-body mechanics. From the equations of motion for rigid-bodies (Table 30.1), it is clear that if any one of displacement, velocity, or

TABLE 30.1 Equations of Motion

Continuous	Discrete $\Delta t = t_i - t_{i-1}$
Rectilinear displacement:	
$x(t) = \int v(t)\,dt$	$x_i = x_{i-1} + \frac{v_i + v_{i-1}}{2} \cdot (\Delta t)$
$= \iint a(t)\,dt$	$= 2x_{i-1} - x_{i-2} + \frac{a_i + 2a_{i-1} + a_{i-2}}{2} \cdot \frac{(\Delta t)}{2}$
Angular displacement:	
$\theta(t) = \int \omega(t)\,dt$	$\theta_i = \theta_{i-1} + \frac{\omega_i + \omega_{i-1}}{2} \cdot (\Delta t)$
$= \iint \alpha(t)\,dt$	$= 2\theta_{i-1} - \theta_{i-2} + \frac{\alpha_i + 2\alpha_{i-1} + \alpha_{i-2}}{2} \cdot \frac{(\Delta t)}{2}$
Rectilinear velocity:	
$v(t) = \frac{d}{dt}x(t)$	$v_i = \frac{x_i - x_{i-1}}{\Delta t}$
$= \int a(t)\,dt$	$= v_{i-1} + \frac{a_i + a_{i-1}}{2} \cdot (\Delta t)$
Angular velocity:	
$\omega(t) = \frac{d}{dt}\theta(t)$	$\omega_i = \frac{\theta_i - \theta_{i-1}}{\Delta t}$
$= \int \alpha(t)\,dt$	$= \omega_{i-1} + \frac{\alpha_i + \alpha_{i-1}}{2} \cdot (\Delta t)$
Rectilinear acceleration:	
$a(t) = \frac{d}{dt}v(t)$	$a_i = \frac{v_i - v_{i-1}}{\Delta t}$
$= \frac{d^2}{dt^2}x(t)$	$= \frac{x_i - 2x_{i-1} + x_{i-2}}{(\Delta t)^2}$
Angular acceleration:	
$\alpha(t) = \frac{d}{dt}\omega(t)$	$\alpha_i = \frac{\omega_i - \omega_{i-1}}{\Delta t}$
$= \frac{d^2}{dt^2}\theta(t)$	$= \frac{\theta_i - 2\theta_{i-1} + \theta_{i-2}}{(\Delta t)^2}$

acceleration is measured, the other two can be derived by mathematical manipulation of the signal within an analog signal conditioner or within the controller software of a digital control system.

Position is simply a location within a frame of reference; thus, any measurement of displacement relative to the frame is a measurement of position, and any displacement transducer whose input is referenced to the frame can be used as a position transducer.

30.2.1 Displacement (Position) Transducers

Displacement transducers may be considered according to application as *gross* (large) displacement transducers or sensitive (small) displacement transducers. The demarcation between gross and sensitive displacement is somewhat arbitrary, but may be conveniently taken as approximately 1 mm for rectilinear displacement and approximately 10′ arc (1/6°) for angular displacement. The predominant types of gross displacement transducers (Figure 30.2) are:

- Potentiometers (Figure 30.2a)
- Variable differential transformers (VDT) (Figure 30.2b)
- Synchros (Figure 30.2c)
- Resolvers (Figure 30.2d)
- Position encoders (Figure 30.2e)

Potentiometer-based transducers are simple to implement and require the least signal conditioning, but potentiometers are subject to wear due to sliding contact between the wiper and the resistance element and may produce noise due to wiper bounce (Figure 30.2a). Potentiometers are available with strokes ranging from less than 1 cm to more than 50 cm (rectilinear) and from a few degrees to more than 50 turns (rotary).

VDTs are not as subject to wear as potentiometers, but the maximum length of the stroke is small, approximately 25 cm or less for a linear VDT (LVDT) and approximately 60° or less for a rotary VDT (RVDT). VDTs require extensive signal conditioning in the form of phase-sensitive demodulation of the AC signal; however, the availability of dedicated VDT demodulators in integrated circuit (IC) packages mitigates this disadvantage of the VDT.

Synchros are rather complex and expensive three-phase AC machines, which are constructed to be precise and rugged. Synchros are capable of measuring angular differences in the positions (up to ±180°) of two continuously rotating shafts. In addition, synchros may function simultaneously as reference input, output measurement device, feedback path, and comparator (Figure 30.2c).

Resolvers are simpler and less expensive than synchros, and they have an advantage over RVDTs in their ability to measure angular displacement throughout 360° of rotation. In Figure 30.2d, which represents one of several possibilities for utilizing a resolver, the signal amplitude is proportional to the cosine of the measured angle at one output coil and the sine of the measured angle at the other. Dedicated ICs are available for signal conditioning and for conversion of resolver output to digital format. The same IC, when used with a *Scott-T transformer*, can be used to convert synchro output to digital format.

Position encoders are highly adaptable to digital control schemes because they eliminate the requirement for digital-to-analog conversion (DAC) of the feedback signal. The code tracks are read by track sensors, usually wipers or electro-optical devices (typically infrared or laser). Position encoders are available for both rectilinear and rotary applications, but are probably more commonly found as shaft encoders in rotary applications. Signal conditioning is straightforward for *absolute* encoders (Figure 30.2e), requiring only a decoder, but position resolution depends on the number of tracks, and increasing the number of tracks increases the complexity of the decoder. *Incremental* encoders require more complex signal conditioning, in the form of counters and a processor for computing position. The number of tracks, however, is fixed at three (Figure 30.2f). Position resolution is limited only by the ability to render finer divisions of the code track on the moving surface.

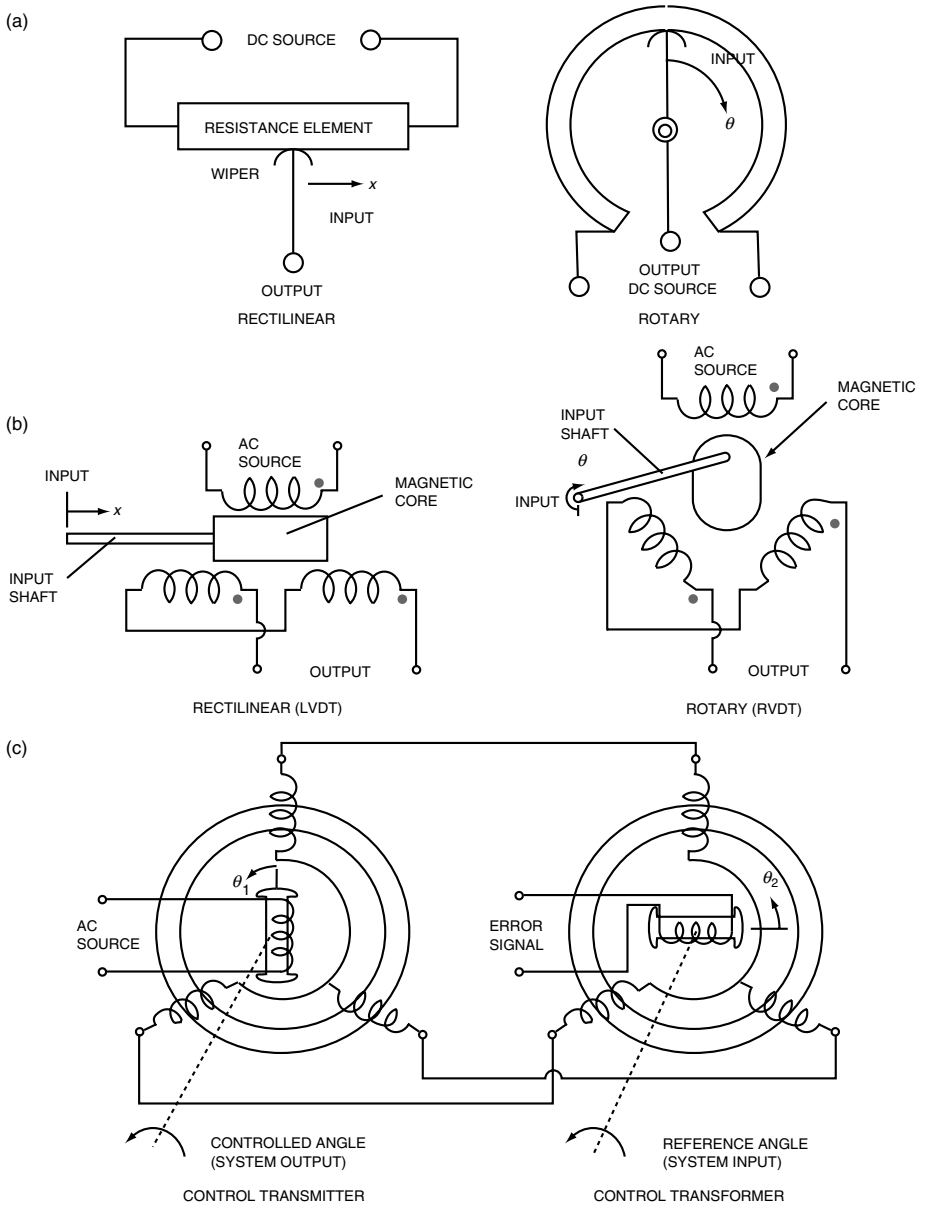


FIGURE 30.2 Gross displacement transducers: (a) potentiometers, (b) variable differential transformers (VDT), (c) synchros (typical connection), (d) resolvers (typical connection), (e) absolute position encoders, (f) code track for incremental position encoder.

Although gross displacement transducers are designed specifically for either rectilinear or rotary motion, a *rack* and *pinion*, or a similar motion converter, is often used to adapt transducers designed for rectilinear motion to the measurement of rotary motion, and vice versa.

The predominant types of *sensitive* (small) displacement transducers (Figure 30.3) are:

- Differential capacitors
- Strain gauge resistors
- Piezoelectric crystals

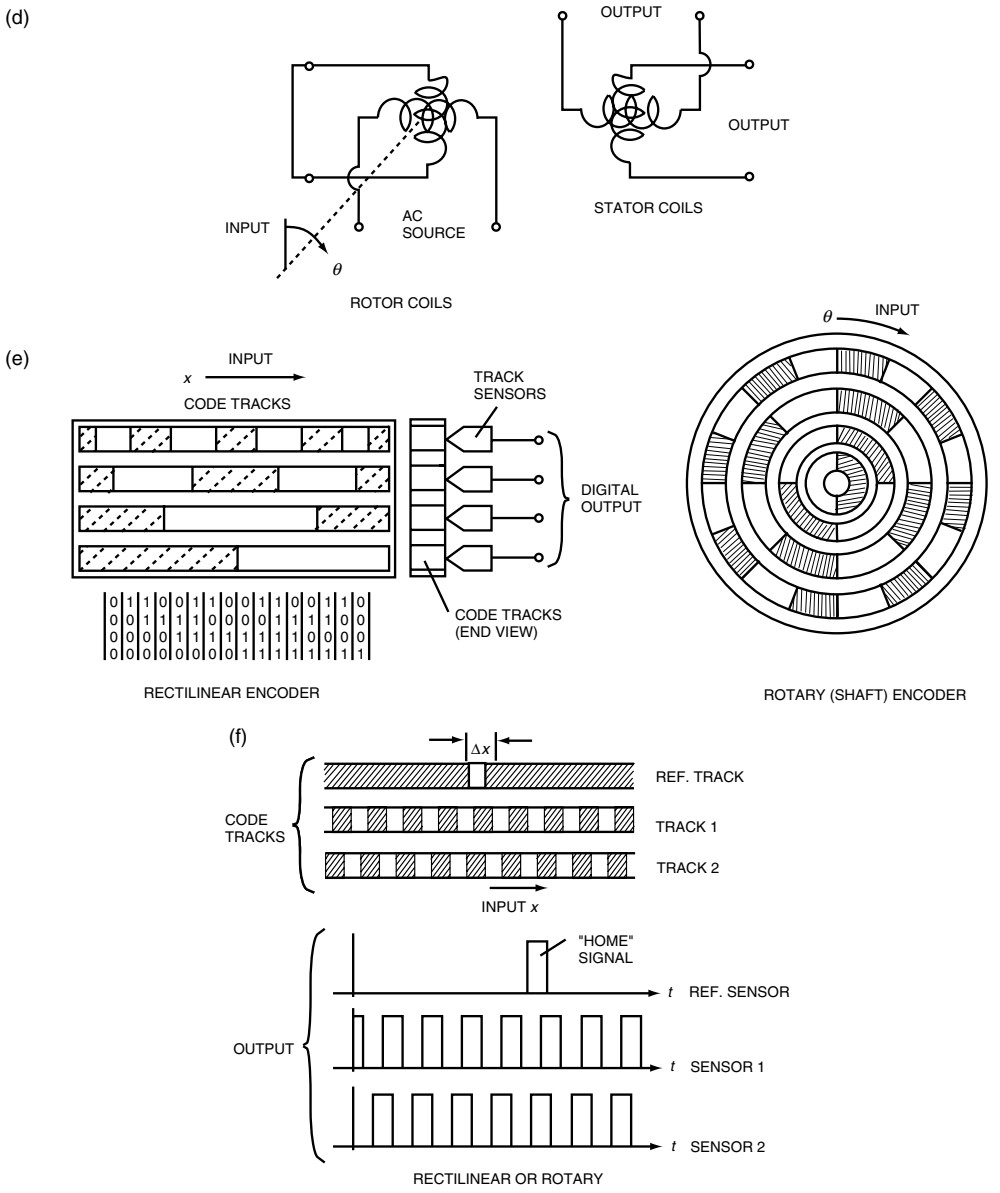


FIGURE 30.2 (*Continued*)

Figure 30.3a provides a simplified depiction of a differential capacitor used for sensitive displacement measurements. The motion of the input rod flexes the common plate, which increases the capacitance of one capacitor and decreases the capacitance of the other. In one measurement technique, the two capacitors are made part of an impedance bridge (such as a Schering bridge), and the change in the bridge output is an indication of displacement of the common plate. In another technique, each capacitor is connected to serve as tuning capacitor for an oscillator, and the difference in frequency between the two oscillators is an indication of displacement.

A strain gauge resistor is used to measure elastic deformation (strain) of materials by bonding the resistor to the material (Figure 30.3b) so that it undergoes the same strain as the material. The resistor is

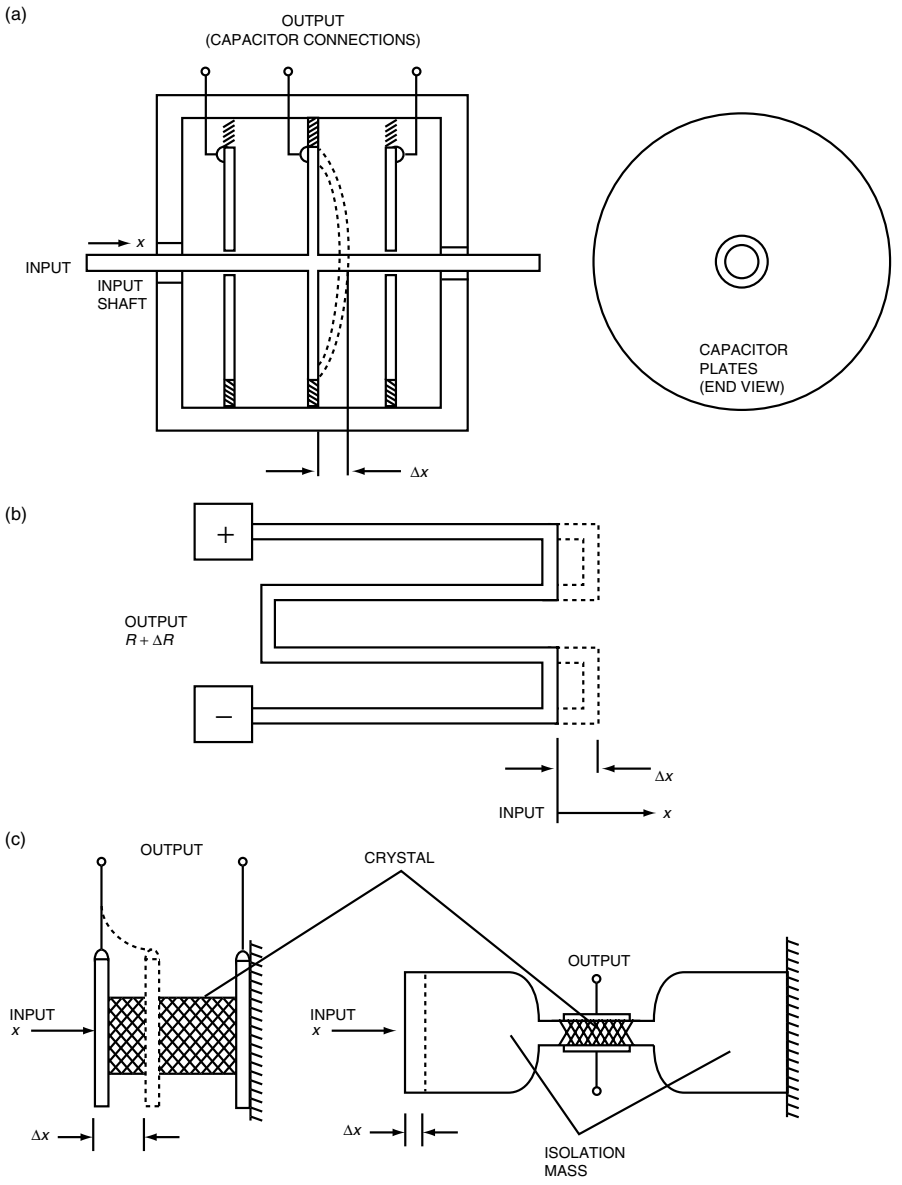


FIGURE 30.3 Sensitive displacement transducers: (a) differential capacitor, (b) strain gauge resistor, (c) piezoelectric crystals.

usually incorporated into one of the several bridge circuits, and the output of the bridge is taken as an indication of strain.

The piezoelectric effect is used in several techniques for sensitive displacement measurements (Figure 30.3c). In one technique, the input motion deforms the crystal by acting directly on one electrode. In another technique, the crystal is fabricated as part of a larger structure, which is oriented so that input motion bends the structure and deforms the crystal. Deformation of the crystal produces a small output voltage and also alters the resonant frequency of the crystal. In a few situations, the output voltage is taken directly as an indication of motion, but more frequently the crystal is used to control an oscillator, and the oscillator frequency is taken as the indication of strain.

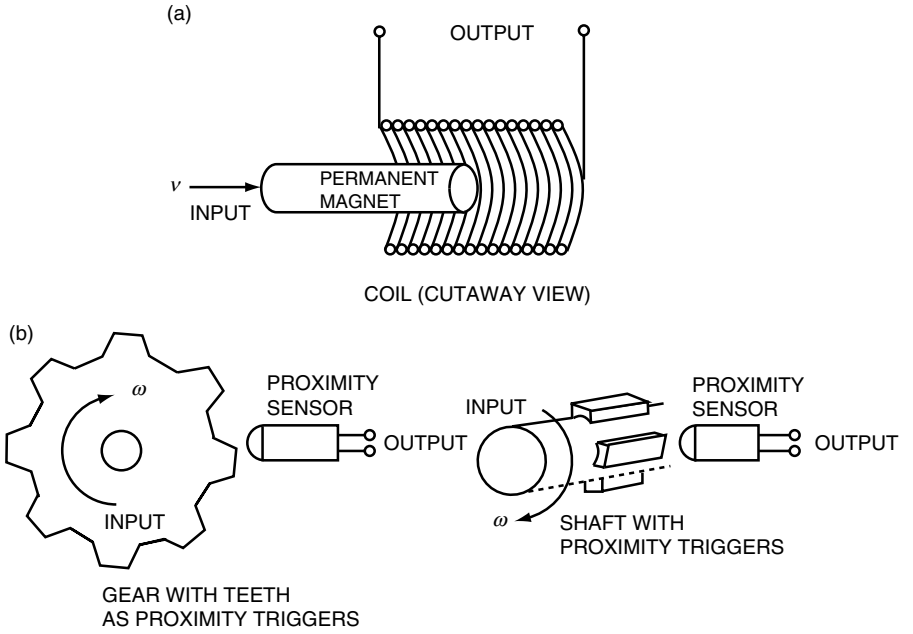


FIGURE 30.4 Velocity transducers: (a) magnet and coil, (b) proximity sensors.

30.2.2 Velocity Transducers

As stated previously, signal conditioning techniques make it possible to derive all motion measurements—displacement, velocity, or acceleration—from a measurement of any one of the three. Nevertheless, it is sometimes advantageous to measure velocity directly, particularly in the cases of short-stroke rectilinear motion or high-speed shaft rotation. The analog transducers frequently used to meet these two requirements are:

- Magnet-and-coil velocity transducers (Figure 30.4a)
- Tachometer generators

A third category of velocity transducers, *counter-type velocity transducers* (Figure 30.4b), is simple to implement and is directly compatible with digital controllers.

The operation of magnet-and-coil velocity transducers is based on Faraday's law of induction. For a solenoidal coil with a high length-to-diameter ratio made with closely spaced turns of fine wire, the voltage induced into the coil is proportional to the velocity of the magnet. Magnet-and-coil velocity transducers are available with strokes ranging from less than 10 mm to approximately 0.5 m.

A tachometer generator is, as the name implies, a small AC or DC generator whose output voltage is directly proportional to the angular velocity of its rotor, which is driven by the controlled output shaft. Tachometer generators are available for shaft speeds of 5000 rpm, or greater, but the output may be nonlinear and there may be an unacceptable output voltage ripple at low speeds.

AC tachometer generators are less expensive and easier to maintain than DC tachometer generators, but DC tachometer generators are directly compatible with analog controllers and the polarity of the output is a direct indication of the direction of rotation. The output of an AC tachometer generator must be demodulated (i.e., rectified and filtered), and the demodulator must be phase sensitive in order to indicate direction of rotation.

Counter-type velocity transducers operate on the principle of counting electrical pulses for a fixed amount of time, then converting the count per unit time to velocity. Counter-type velocity transducers

rely on the use of a proximity sensor (*pickup*) or an incremental encoder (Figure 30.2f). Proximity sensors may be one of the following types:

- Electro-optic
- Variable reluctance
- Hall effect
- Inductance
- Capacitance

Two typical applications of counter-type velocity transducers are shown in Figure 30.4b.

Since a digital controller necessarily includes a very accurate electronic clock, both pulse counting and conversion to velocity can be implemented in software (i.e., made a part of the controller program). Hardware implementation of pulse counting may be necessary if time-intensive counting would divert the controller from other necessary control functions. A special-purpose IC, known as a *quadrature decoder/counter interface*, can perform the decoding and counting functions and transmit the count to the controller as a data word.

30.2.3 Acceleration Transducers

As with velocity measurements, it is sometimes preferable to measure acceleration directly, rather than derive acceleration from a displacement or velocity measurement. The majority of acceleration transducers may be categorized as *seismic* accelerometers because the measurement of acceleration is based on measuring the displacement of a mass called the *seismic element* (Figure 30.5). The configurations shown in Figures 30.5a,b require a rather precise arrangement of springs for suspension and centering of the seismic mass. One of the disadvantages of a seismic accelerometer is that the seismic mass is displaced during acceleration, and this displacement introduces nonlinearity and bias into the measurement. The force-balance configuration shown in Figure 30.5c uses the core of an electromagnet as the seismic element. A sensitive displacement sensor detects displacement of the core and uses the displacement signal in a negative feedback arrangement to drive the coil, which returns the core to its center position. The output of the force-balance accelerometer is the feedback required to prevent displacement rather than displacement per se.

A simpler seismic accelerometer utilizes one electrode of a piezoelectric crystal as the seismic element (Figure 30.5d). Similarly, another simple accelerometer utilizes the common plate of a differential capacitor (Figure 30.3a) as the seismic element.

30.2.4 Force Transducers

Force measurements are usually based on a measurement of the motion, which results from the applied force. If the applied force results in gross motion of the controlled output, and the mass of the output element is known, then any appropriate accelerometer attached to the controlled output produces an output proportional to the applied force ($F = Ma$). A simple spring-balance scale (Figure 30.6a) relies on measurement of displacement, which results from the applied force (weight) extending the spring.

Highly precise force measurements in high-value servomechanisms, such as those used in pointing and tracking devices, frequently rely on gyroscope precession as an indication of the applied force. The scheme is shown in Figure 30.6b for a gyroscope with gimbals and a spin element. A motion transducer (either displacement or velocity) on the precession axis provides an output proportional to the applied force. Other types of gyroscopes and precession sensors are also used to implement this force measurement technique.

Static force measurements (in which there is no apparent motion) usually rely on measurement of strain due to the applied force. Figure 30.6c illustrates the typical construction of a common force transducer called a *load cell*. The applied force produces a proportional strain in the S-shaped structural member, which is measured with a sensitive displacement transducer, usually a strain gauge resistor or a piezoelectric crystal.

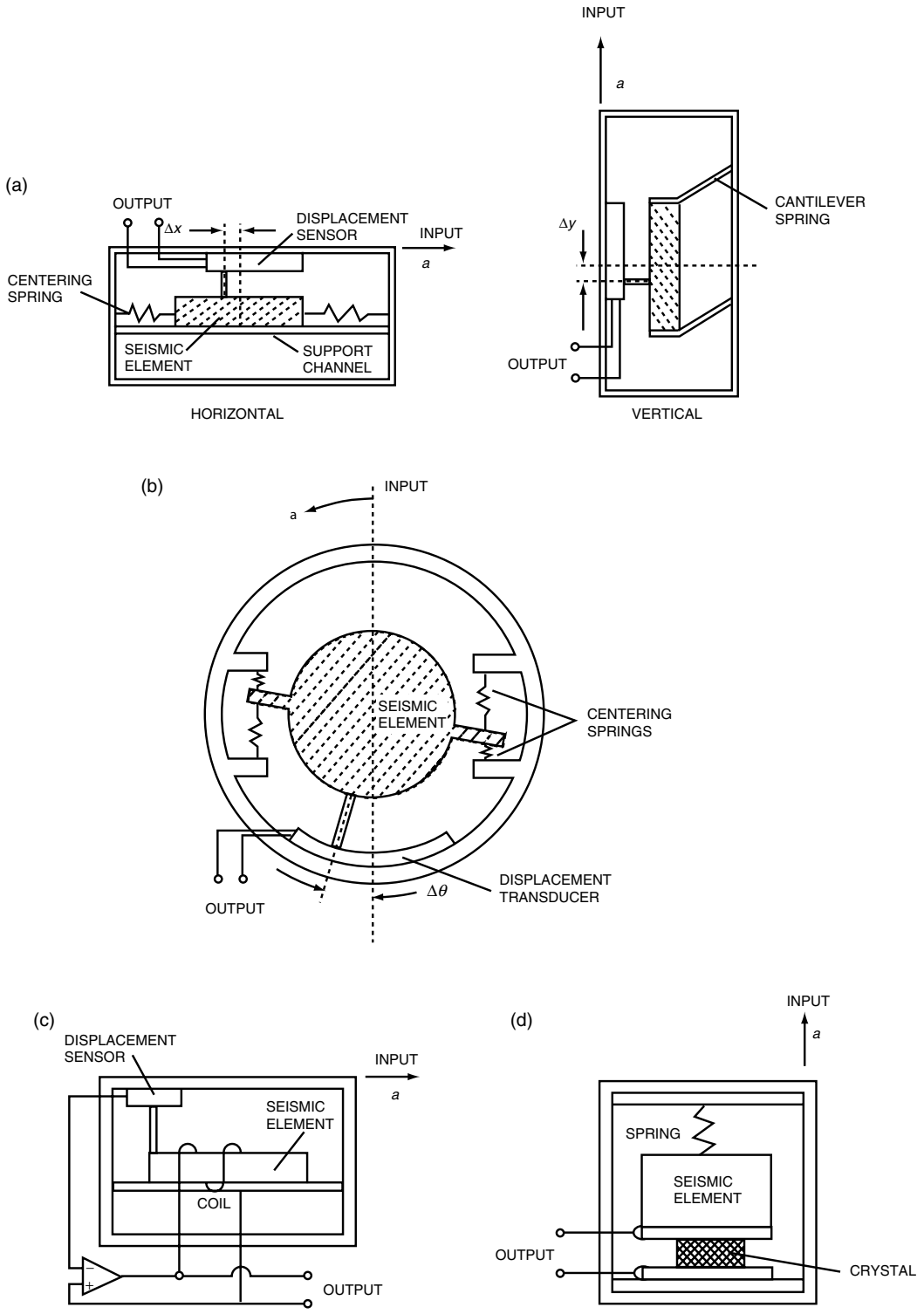


FIGURE 30.5 Seismic accelerometers: (a) rectilinear acceleration transducers, (b) rotary accelerometer, (c) force-balance accelerometer, (d) piezoelectric accelerometer.

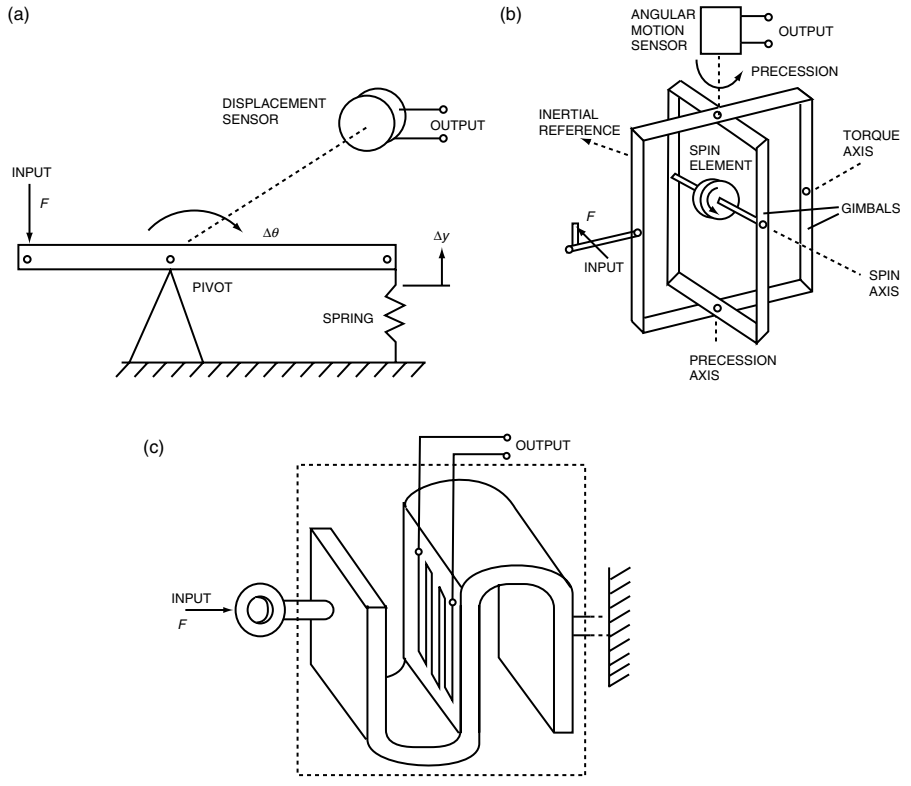


FIGURE 30.6 Force transducers: (a) spring scale, (b) gyroscope, (c) load cell.

30.3 Process Transducers

This section discusses transducers used in measuring and controlling the *process variables* most frequently encountered in industrial processes, namely,

- Fluid pressure
- Fluid flow
- Liquid level
- Temperature

30.3.1 Fluid Pressure Transducers

Most fluid pressure transducers are of the *elastic* type, in which the fluid is confined in a chamber with at least one elastic wall, and the deflection of the elastic wall is taken as an indication of the pressure. The *Bourdon tube* and the *bellows* are examples of elastic pressure transducers, which are used in laboratory-grade transducers and in some industrial process control applications. The fluid pressure transducer depicted in Figure 30.7, which uses an elastic *diaphragm* to separate two chambers, is the type most frequently encountered in industrial process control. Diaphragms are constructed from one of a variety of elastic materials ranging from thin metal to polymerized fabric.

For gross pressure measurements, the displacement of the diaphragm is sensed by a potentiometer or LVDT; for more sensitive pressure measurements, any one of the three sensitive displacement sensors described earlier is used. In the most common configuration for sensitive pressure transducers, a strain gauge resistor with a rosette pattern is bonded to the diaphragm. In another configuration, the outer

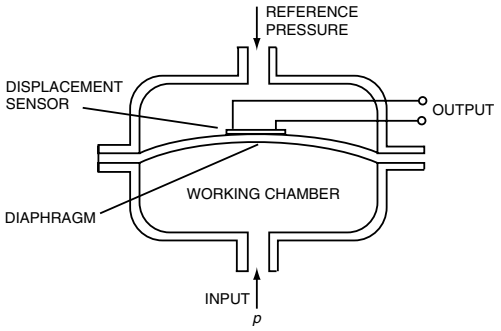


FIGURE 30.7 Diaphragm pressure transducer.

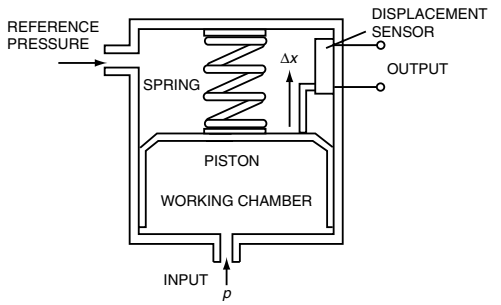


FIGURE 30.8 Piston-and-spring transducer.

walls of the pressure sensor serve as capacitor plates and the diaphragam serves as the common plate of a differential capacitor. In a very sensitive and highly integrated configuration, the diaphragm is a silicon wafer with a piezoresistive strain gauge and signal conditioning circuits integrated into the silicon.

High-vacuum (very low pressure) measurements, usually based on observations of viscosity, thermal conductivity, acoustic properties, or ionization potential of the fluid, will not be included in this discussion. Transducers used in high-pressure hydraulic systems [70 MPa (10,000 psi) or greater] are usually of the *piston and spring* type (Figure 30.8).

In either of the pressure transducers, the output is actually a measure of the difference in pressure between the working chamber and the reference chamber of the transducer (i.e., $p_{OUT} = p - p_{REF}$). The measurement is called:

- An *absolute pressure* if the reference chamber is sealed and evacuated (i.e., $p_{REF} = 0$ and $p_{OUT} = p$)
- A *gauge pressure* if the reference chamber is vented to the atmosphere (i.e., $p_{OUT} = p - p_{ATM}$)
- A *differential pressure* if any other pressure is applied to the reference chamber

30.3.2 Fluid Flow Transducers (Flowmeters)

Flowmetering, because of the number of variables involved, encompasses a wide range of measurement technology and applications. In industrial processes, the term *fluid* is applied not only to gases and liquids, but also to flowable mixtures (often called *slurries* or *sludges*) such as concrete, sewage, or wood pulp. Control of a fluid flow, and hence the type of measurement required, may involve *volumetric* flow rate, *mass* flow rate, or flow direction. Gas flows may be *compressible*, which also influences the measurement technique. In addition, the condition of the flow—whether or not it is homogenous and clean (free of suspended particles)—has a bearing on flowmeter technology. Another factor to be considered is flow velocity; slow moving laminar flows of viscous material require different measurement techniques than

those used for high-velocity turbulent flows. Still another consideration is confinement of the flow. Whereas most fluid flow measurements are concerned with *full flow* through *closed channels* such as ducts and pipes, some applications require measurements of *partial flow* through *open channels* such as troughs and flumes. Only the most widely used flowmeters are considered here.

The major categories of flowmeters are:

- Differential pressure, constriction-type (venturi, orifice, flow nozzle, elbow (or pipe bend), and pitot static) (Figure 30.9)
- Fluid-power (gear motors, turbines, and paddle wheels) (Figure 30.10)

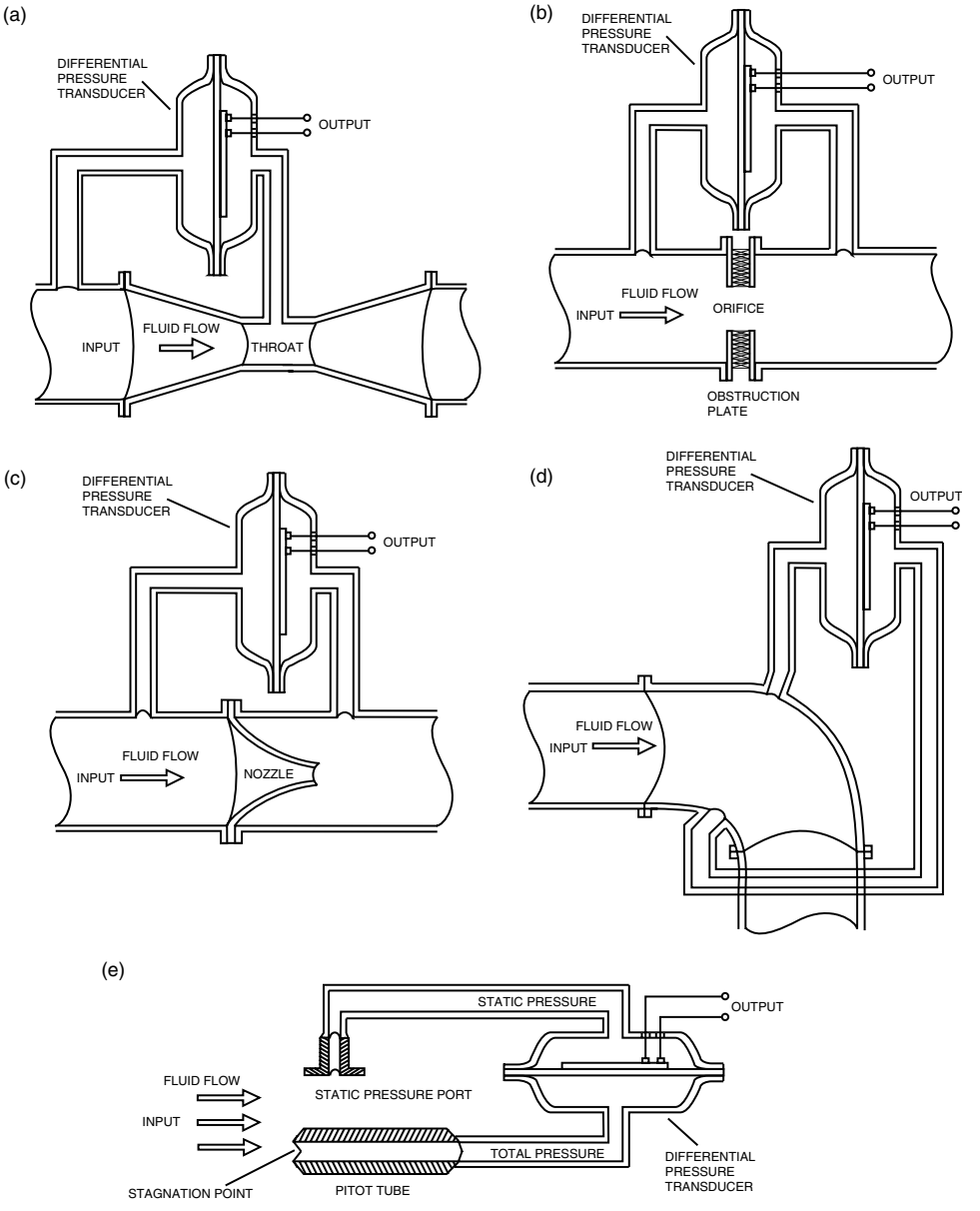


FIGURE 30.9 Differential pressure flowmeters: (a) Venturi flowmeter, (b) orifice flowmeter, (c) nozzle flowmeter, (d) pipebend (elbow) flowmeter, (e) pitot-static-flowmeter.

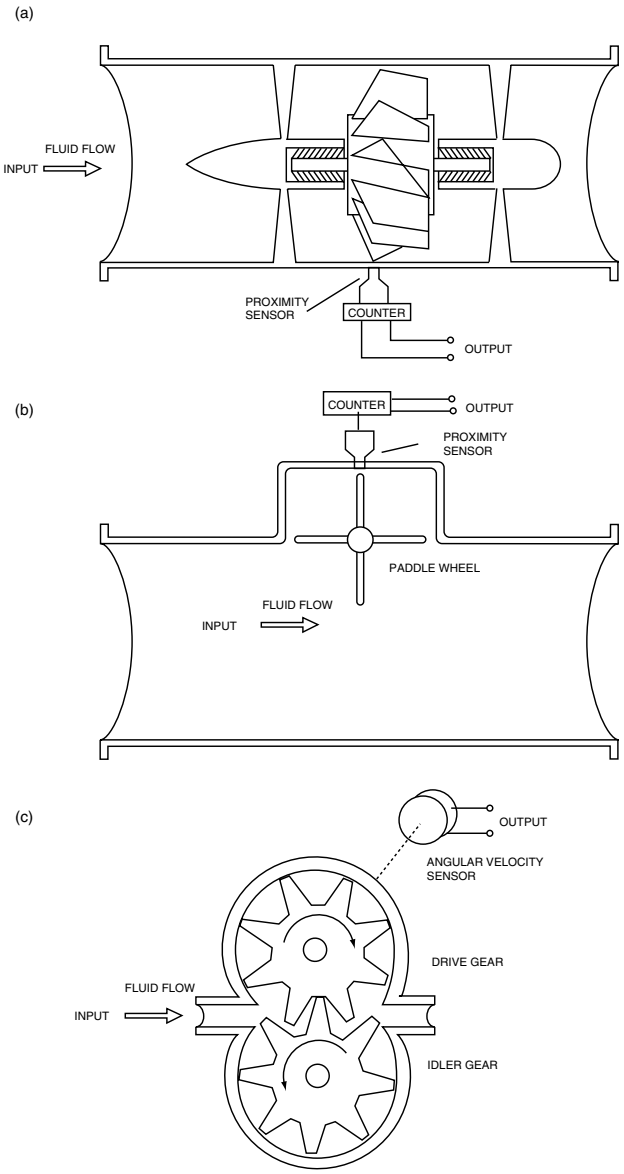


FIGURE 30.10 Fluid power flowmeters: (a) turbine flowmeter, (b) paddle wheel flowmeter, (c) gear motor flowmeter.

- Ultrasound (Figure 30.11)
- Vortex shedding (Figure 30.12)
- Thermal anemometer (Figure 30.13)
- Electromagnetic (Figure 30.14)
- Rotameter (variable-area in-line flowmeter) (Figure 30.15)

Differential pressure flowmeters are suited to high- and moderate-velocity flow of gas and clean, low-viscosity liquids. Venturi flowmeters (Figure 30.9a) are the most accurate, but they are large and expensive. Orifice flowmeters (Figure 30.9b) are smaller, less expensive, and much less accurate than venturi flowmeters.

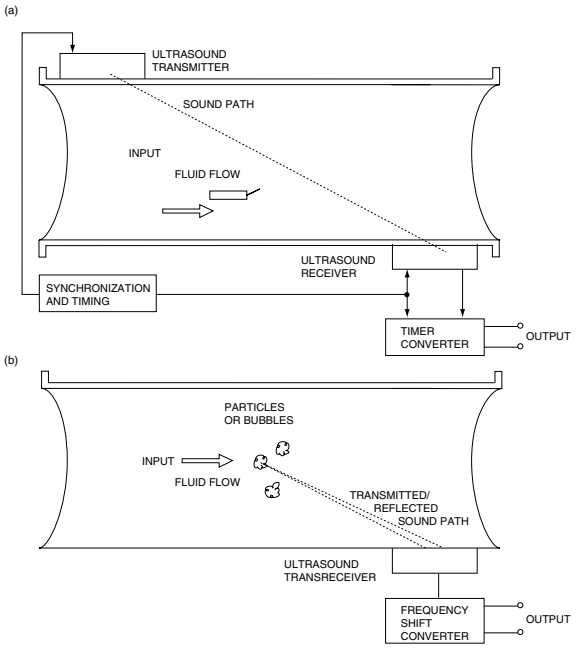


FIGURE 30.11 Ultrasound flowmeters: (a) transmission-type ultrasound flowmeter, (b) doppler ultrasound flowmeter.

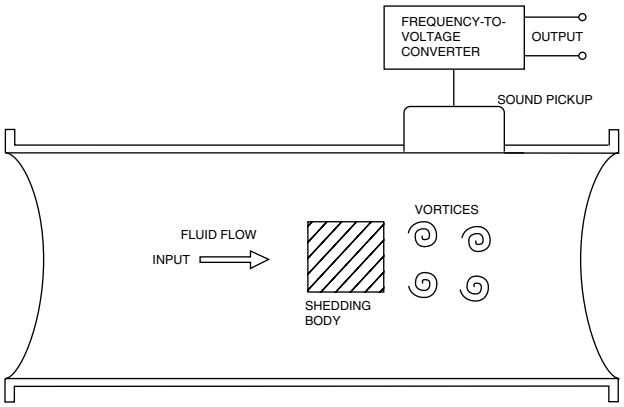


FIGURE 30.12 Vortex-shedding flowmeter.

Nozzle flowmeters (Figure 30.9c) are a compromise between venturi and orifice flowmeters. Pipe-bend flowmeters (Figure 30.9d), which can essentially be installed in any bend in an existing piping system, are used primarily for gross flow rate measurements. Pitot-static flowmeters (Figure 30.9e) are used in flows which have a large cross-sectional area, such as in wind tunnels. Pitot-static flowmeters are also used in freestream applications such as airspeed indicators for aircraft.

Fluid-power flowmeters are used in low-velocity, moderately viscous flows. In addition to industrial control applications, turbine flowmeters (Figure 30.10a) are sometime used as speed indicators for ships or boats. Paddle wheel flowmeters (Figure 30.10b) are used both in closed- and open-flow applications such as liquid flow in flumes. Since a fluid-power gear motor (Figure 30.10c) is a constant volume device, motor shaft speed is always a direct indication of fluid flow rate.

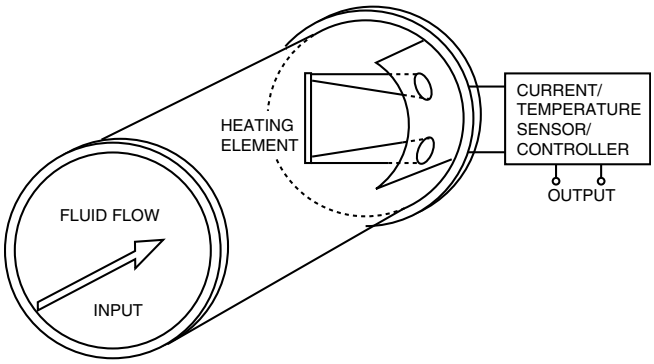


FIGURE 30.13 Thermal anemometer.

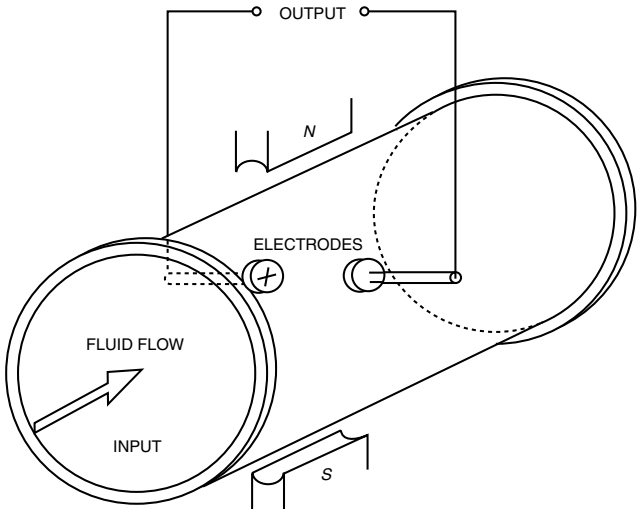


FIGURE 30.14 Electromagnetic flowmeter.

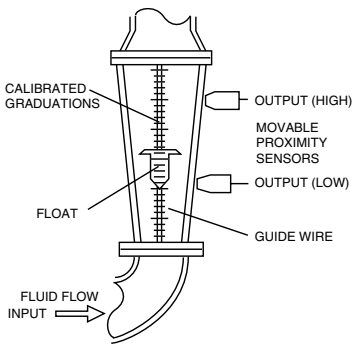


FIGURE 30.15 Variable-area in-line flowmeter (rotameter).

Ultrasound flowmeters of the transmission type (Figure 30.11a), which are based on the principle that the sound transmission speed will be increased by the flow rate of the fluid, are used in all types of clean, subsonic flows. Doppler flowmeters (Figure 30.11b) rely on echoes from within the fluid, and are thus only useful in dirty flows that carry suspended particles or turbulent flows that produce bubbles. Ultrasound flowmeters are nonintrusive devices, which can often be retrofitted to existing duct or pipe systems.

Vortex shedding flowmeters (Figure 30.12) introduce a *shedding body* into the flow to cause production (*shedding*) of *vortices*. The sound accompanying the production and collapse of the vortices is monitored and analyzed. The dominant frequency of the sound is indicative of the rate of vortex production and collapse, and hence an indication of flow rate. Vortex shedding flowmeters are useful in low-velocity, nonturbulent flows.

Thermal anemometers (Figure 30.13) are used in low-velocity gas flows with large cross-sectional area, such as in heating, ventilation, and air conditioning (HVAC) ducts. Convection cooling of the heating element is related to flow rate. The flow rate measurement is based either on the current required to maintain a constant temperature in the heating element, or alternatively on the change in temperature when the current is held constant.

Electromagnetic flowmeters (Figure 30.14) are useful for slow moving flows of liquids, sludges, or slurries. The flow material must support electrical conduction between the electrodes, and so in some cases it is necessary to ionize the flow upstream from the measurement point in order to use an electromagnetic flowmeter.

Variable-area in-line flowmeters (Figure 30.15), or *rotameters*, are sometimes referred to as sight gauges because they provide a visible indication of the flow rate. These devices, when fitted with proximity sensors (such as capacitive pickups) that sense the presence of the float, can be used in on–off control applications.

30.3.3 Liquid Level Transducers

Liquid-level measurements are relatively straightforward, and the transducers fall into the categories of *contact* or *noncontact*. Measurements may be *continuous*, in which the liquid level is monitored continuously throughout its operating range, or *point*, in which the liquid level is determined to be above or below some predetermined level.

The contact transducers encountered most frequently are:

- Float
- Hydrostatic pressure
- Electrical capacitance
- Ultrasound

The noncontact transducers encountered most frequently are:

- Capacitive proximity sensors
- Ultrasound
- Radio frequency
- Electro-optical

Float-type liquid level transducers are available in a wide variety of configurations for both continuous and point measurements. One possible configuration is depicted in Figure 30.16 for continuous measurement and for both single- and dual-point measurements.

Hydrostatic pressure liquid level transducers may be used in either vented or pressurized applications (Figure 30.17). In either case the differential pressure is directly proportional to the weight of the liquid column, since the differential pressure transducer accounts for surface pressure.

Capacitance probes (Figure 30.18a) are widely used in liquid level measurements. It is possible, when the tank walls are metal, to use a single bare or insulated metal rod as one capacitor plate and the tank walls as the other. More frequently, capacitance probes consist of a metal rod within a concentric cylinder

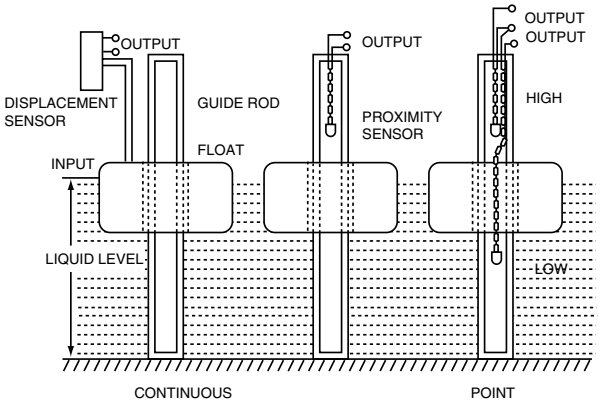


FIGURE 30.16 Float-type liquid level transducers.

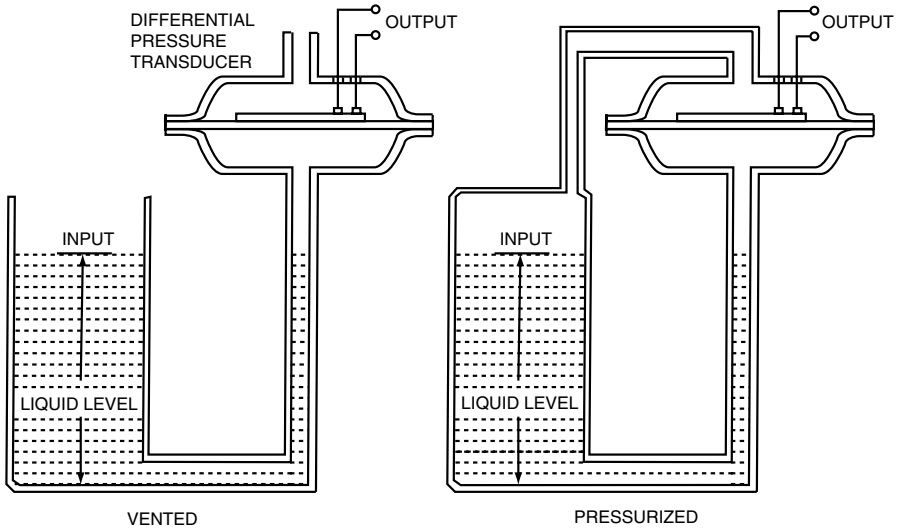


FIGURE 30.17 Hydrostatic pressure liquid level transducers.

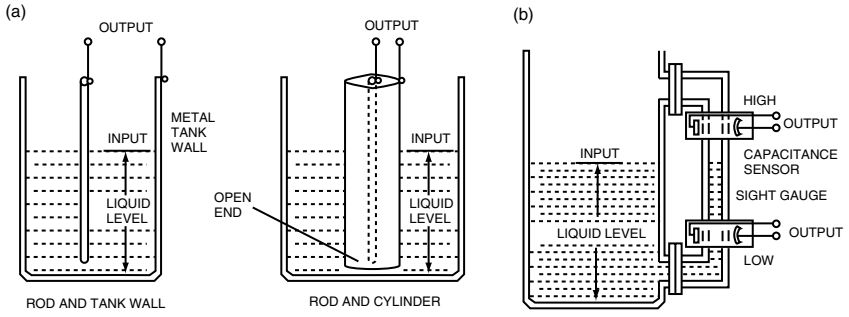


FIGURE 30.18 Capacitive-type liquid level transducers: (a) capacitive probes, (b) capacitive switches.

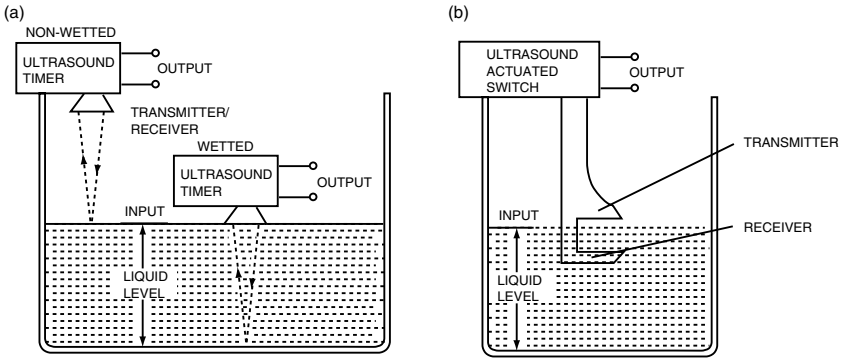


FIGURE 30.19 Ultrasound liquid level transducers: (a) echo-ranging liquid level transducer, (b) ultrasound switch.

open at the ends, which makes the transducer independent of the tank construction. An interesting application of this type of capacitance probe is as aircraft fuel quantity indicators. Capacitance switches can be utilizes as depicted in Figure 30.18b to provide noncontact point measurements of liquid level.

Ultrasound echo ranging transducers can be used in either *wetted* (contact) or *nonwetted* (noncontact) configurations for continuous measurement of liquid level (Figure 30.19a). An interesting application of wetted transducers is as depth finders and fish finders for ships and boats. Nonwetted transducers can also be used with bulk materials such as grains and powders. Radio-frequency and electro-optic liquid level transducers are usually noncontact, echo ranging devices that are similar in principle and application to the nonwetted ultrasound transducer.

Ultrasonic transducers can also be adapted to point measurements by locating the transmitter and the receiver opposite one another across a gap (Figure 30.19b). When liquid fills the gap, attenuation of the ultrasound energy is markedly less than when air fills the gap. The signal conditioning circuits utilize this sharp increase in the level of ultrasound energy detected by the receiver to activate a switch.

30.3.4 Temperature Transducers

Temperature measurement is generally based on one of the following physical principles:

- Thermal expansion
- Thermoelectric phenomena
- Thermal effect on electrical resistance
- Thermal effect on conductance of semiconductor junctions
- Thermal radiation

(Strictly speaking, any device used to measure temperature may be called a thermometer, but more descriptive terms are applied to devices used in temperature control.)

Bimetallic switches (Figure 30.20) are widely used in on–off temperature control systems. If two metal strips with different *coefficients of thermal expansion* are bonded together while both strips are at the same temperature, the bimetallic structure will bend when the temperature is changed. Although these devices are often called *thermal cutouts*, implying that they are used in normally closed switches, they can be fabricated in either normally closed or normally open configurations. The bimetallic elements can also be fabricated in coil or helical configurations to extend the range of motion due to thermal expansion.

Thermocouples are rugged and versatile temperature sensors frequently found in industrial control systems. A thermocouple consists of a pair of dissimilar metal wires twisted or otherwise bonded at one end. The *Seebeck effect* is the physical phenomena that accounts for thermocouple operation, so thermocouples are known alternatively as *Seebeck junctions*. The potential difference (*Seebeck voltage*) between the free ends of the wire is proportional to the difference between the temperature at the junction and

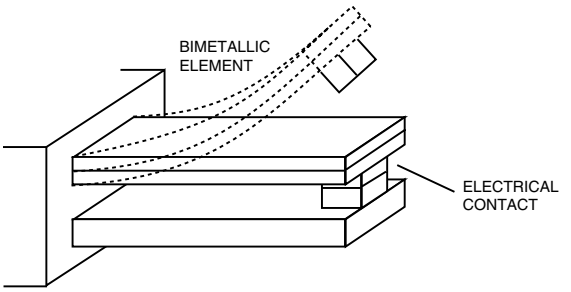


FIGURE 30.20 Bimetallic thermal switch.

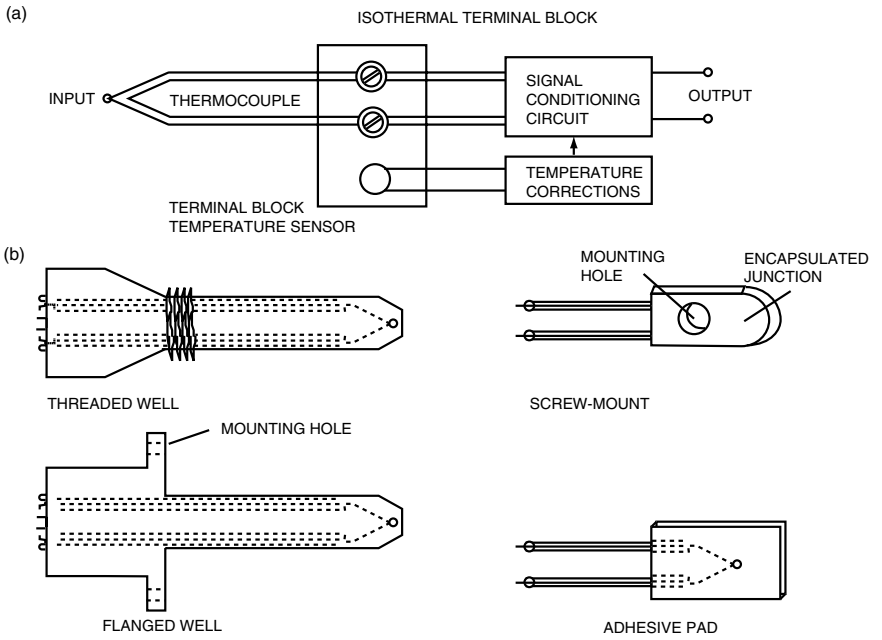


FIGURE 30.21 Thermocouples: (a) typical thermocouple connection, (b) some thermocouple accessories and configurations.

the temperature at the free ends. Thermocouples are available for measurement of temperature as low as −270°C and as high as 2300°C, although no single thermocouple covers this entire range. Thermocouples are identified as type B, C, D, E, G, J, K, N, R, S, or T, according to the metals used in the wire.

Signal conditioning and amplification of the relatively small Seebeck voltage dictates that the thermocouple wires must be connected to the terminals of a signal conditioning circuit. These connections create two additional Seebeck junctions, each of which generates its own Seebeck voltage, which must be canceled in the signal conditioning circuit. To implement cancellation to the corrections the following are necessary (Figure 30.21a):

- The input terminals of the signal conditioning circuit must be made of the same metal.
- The two terminals must be on an *isothermal terminal block* so that each Seebeck junction created by the connection is at the same temperature.
- The temperature of the terminal block must be known.

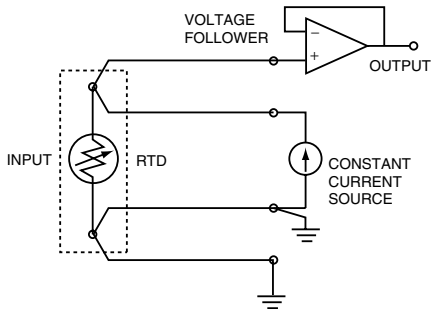


FIGURE 30.22 Typical resistance temperature detector (RTD) application.

The first two requirements are met by appropriate construction of the signal conditioning circuit. The third requirement is met by using a reference temperature sensor, probably an IC temperature transducer of the type described later.

Thermocouple and thermocouple accessories are fabricated for a variety of applications (Figure 30.21b). Protective shields (thermowells) are used to protect thermocouple junctions in corrosive environments or where conducting liquids can short circuit the thermocouple voltage; however, exposed (bare) junctions are used wherever possible, particularly when a fast response is essential.

Resistance temperature detectors (RTD) are based on the principle that the electrical *resistivity* of most metals increases predictably with temperature. Platinum is the preferred metal for RTDs, although other less expensive metals are used in some applications. The resistivity of platinum is one of the standards by which temperature is measured. The relatively good linearity of the resistivity of platinum over a wide temperature range (–200 to 800°C) makes platinum RTDs suitable for stable, accurate temperature transducers, which are easily adapted to control systems applications.

The disadvantage of the RTD is that the temperature-sensitive element is a rather fragile metal filament wound on a ceramic bobbin or a thin metal film deposited on a ceramic substrate. RTD elements are usually encapsulated and are rarely used as bare elements. The accessories and application packages used with RTDs are similar to those used with thermocouples (Figure 30.21b).

Most platinum RTDs are fabricated so as to have a nominal resistance of 100 Ω at 0°C. The *resistance temperature coefficient* of platinum is approximately 3–4 mΩ/Ω/°C, so resolution of the temperature to within 1°C for a nominal 100-Ω RTD element requires resolution of the absolute resistance within 0.3–0.4 Ω. These resistance resolution requirements dictate use of special signal conditioning techniques to cancel the lead and contact resistance of the RTD element (Figure 30.22). The circuit depicted in Figure 30.22 is a variation of a 4-*wire ohmmeter*. Most RTDs are manufactured with four leads to be compatible with such circuits.

Thermistors are specially prepared metal oxide semiconductors that exhibit a strong *negative* temperature coefficient, in sharp contrast to the weak positive temperature coefficient of RTDs. Nominal thermistor resistance, usually specified for 25°C, ranges from less than 1000 Ω to more than 1 MΩ, with sensitivities greater than 100 Ω/°C. Thus, the thermistor is the basis for temperature sensors that are much more sensitive and require less special signal conditioning than either thermocouples or RTDs. The tradeoff is the marked nonlinearity of the resistance-temperature characteristic. To minimize this problem, manufacturers provide packages in which the thermistor has been connected into a resistor network chosen to provide a relatively linear resistance-temperature characteristic over a nominal temperature range.

The development of thermistor technology has lead to the IC temperature sensor in which the temperature-sensitive junction(s) and the required signal conditioning circuits are provided in a monolithic package. The user is only required to provide a supply voltage (typically 5 V DC) to the IC in order to obtain an analog output voltage proportional to temperature. Thermistors and IC temperature sensors

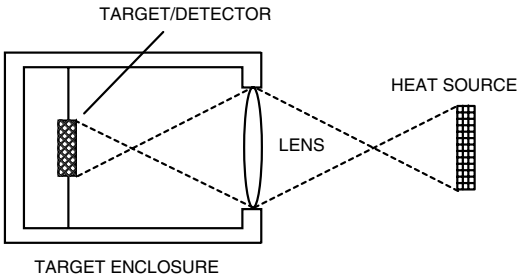


FIGURE 30.23 Schematic of the radiation thermometry scheme.

can be produced in very small packages, which permit highly localized temperature measurements. Some thermistors designed for biological research are mounted in the tip of a hypodermic needle. The shortcomings of both thermistor and IC temperature sensors are that they are not rugged, cannot be used in caustic environments, and are limited to temperatures below approximately 200°C.

Radiation thermometers are used for remote (noncontact) sensing of temperature in situations where contact sensors cannot be used. Operation is based on the principles of heat transfer through thermal radiation. Radiation thermometers focus the *infrared* energy from a heat source onto a *black body* (target) within the radiation thermometer enclosure (Figure 30.23). One of the contact temperature sensors described previously is incorporated into the target to measure the target temperature. The rise in temperature at the target is related to the source temperature. Typical radiation thermometers have standoff ranges (focal lengths) of 0.5–1.5 m, but instruments with focal length as short as 1 cm or as long as 10 m are available. Radiation thermometers are available for broadband, monochromatic, or two-color thermometry.

30.4 Transducer Performance

The operation of a transducer within a control system can be described in terms of its *static performance* and its *dynamic performance*. The static characteristics of greatest interest are:

- Scale factor (or sensitivity)
- Accuracy, uncertainty, precision, and system error (or bias)
- Threshold, resolution, dead band, and hysteresis
- Linearity
- Analog drift

The dynamic characteristics of greatest interest are:

- Time constant, response time, and rise time
- Overshoot, settling time, and damped frequency
- Frequency response

Static performance is documented through *calibration*, which consists of applying a known input (quantity or phenomenon to be measured) and observing and recording the transducer output. In a typical calibration procedure, the input is increased in increments from the lower range limit to the upper range limit of the transducer, then decreased to the lower range limit. The range of a component consists of all allowable input values. The difference between the upper and lower range limits is the *input span* of the component; the difference between the output at the upper range limit and the output at the lower range limit is the *output span*.

Dynamic performance is documented by applying a known change, usually a step, in the input and observing and recording the transducer output, usually with a strip recorder or a storage oscilloscope.

30.5 Loading and Transducer Compliance

A prime requirement for an appropriate transducer is that it be *compliant* at its input. Compliance in this sense means that the input energy required for proper operation of the transducer, and hence a correct measurement of the controlled output, does not significantly alter the controlled output. A transducer that does not have this compliance is said to *load* the controlled output. For example, a voltmeter must have a high-impedance input in order that the voltage measurement does not significantly alter circuit current and, hence, alter the voltage being measured.

Defining Terms

Controlled output: The principal product of an automatic control system; the quantity or physical activity to be measured for automatic control.

Feedback path: The cascaded connection of transducer and signal conditioning components in an automatic control system.

Forward path: The cascaded connection of controller, actuator, and plant or process in an automatic control system.

Motion transducer: A transducer used to measure the controlled output of a servomechanism; usually understood to include transducers for static force measurements.

Plant or process: The controlled device that produces the principal output in an automatic control system.

Process control: The term used to refer to the control of industrial processes; most frequently used in reference to control of temperature, fluid pressure, fluid flow, and liquid level.

Process transducer: A transducer used to measure the controlled output of an automatic control system used in process control.

Reference input: The signal provided to an automatic control system to establish the required controlled output; also called *setpoint*.

Servomechanism: A system in which some form of motion is the controlled output.

Signal conditioning: In this context, the term used to refer to the modification of the signal in the feedback path of an automatic control system; signal conditioning converts the sensor output to an electrical signal suitable for comparison to the reference input (setpoint); the term can also be applied to modification of forward path signals.

Transducer: The device used to measure the controlled output in an automatic control system; usually consists of a sensor or pickup and signal conditioning components.

References

Bateson, R.N. 1993. *Introduction to Control System Technology,* 4th ed. Merrill, Columbus, OH.

Berlin, H.M. and Getz, F.C., Jr. 1988. *Principles of Electronic Instrumentation and Measurement.* Merrill, Columbus, OH.

Buchla, D. and McLachlan, W. 1992. *Applied Electronic Instrumentation and Measurement.* Macmillan, New York.

Chaplin, J.W. 1992. *Instrumentation and Automation for Manufacturing.* Delmar, Albany, NY.

Doeblin, E.O. 1990. *Measurement Systems Application and Design,* 4th ed. McGraw-Hill, New York.

Dorf, R.C. and Bishop, R.H. 1995. *Modern Control Systems,* 7th ed. Addison-Wesley, Reading, MA.

O'Dell, T.H. 1991. *Circuits for Electronic Instrumentation.* Cambridge Univ. Press, Cambridge, England, UK.

Seippel, R.G. 1983. *Transducers, Sensors, and Detectors.* Reston Pub., Reston, VA.

Webb, J. and Greshock, K. 1993. *Industrial Control Electronics,* 2nd ed. Macmillan, New York.

Further Information

Manufacturers and vendors catalogs, data documents, handbooks, and applications notes, particularly the handbook series (current year) by Omega Engineering, Inc.:

The Flow and Level Handbook
The Pressure, Strain, and Force Handbook
The Temperature Handbook

Trade journals, magazines, and newsletters, particularly:

Instrumentation Newsletter (National Instruments)
Personal Engineering and Instrumentation News
Test and Measurement News (Hewlett Packard)
Test and Measurement World

31

A/D and D/A Conversion

Brian Betts
National Instruments, Inc.

31.1 Introduction

As computers began to gain popularity, engineers and scientists realized that computers could become a powerful tool for data acquisition and control. However, almost all real-world phenomena (such as light, pressure, velocity, temperature, etc.) are measured with sensors that produce analog signals, and computers, on the other hand, rely on digital signals. Therefore, many companies began to invest in advancements in analog-to-digital and digital-to-analog converters (ADC and DAC). These devices have become the keystone in every measurement device. This chapter will examine the ADC and DAC on a functional level as well as discuss important specifications of each.

31.2 Sampling

In order to convert an analog signal into a digital signal, the analog signal must first be sampled. Sampling involves converting one value of a signal at a particular interval of time. Generally, conversions happen uniformly in time. For example, a digitizing system may convert a signal every 5 ms, or sample at 200 k S/s. Although it is not necessary to uniformly sample a signal, doing so provides certain benefits that will be discussed later.

A typical sampling circuit contains two major components: a track-and-hold (T/H) circuit and the ADC. Since the actual conversion in the ADC takes some amount of time, it is necessary to hold constant the value of the signal being converted. At the instance the sample is to be taken, the T/H holds the sample value even if the signal is still changing. Once the conversion has been completed, the T/H releases the value it is currently storing and is ready to track the next value.

One aspect of sampling that cannot be avoided is that some information is lost because of the fact that an analog waveform actually has an infinite number of samples and there is no way to capture every value. The major pitfall associated with this fact is called undersampling or sampling too slow. If a 10-kHz sine wave is to be acquired and sampling only occurs at 5 kS/s, the true waveform will not be preserved. In fact, a waveform of a different frequency will result. The result of undersampling is often referred to as aliasing. According to the Nyquist theory, which deals with sampling, sampling should occur at a rate twice as high as the highest frequency component of the signal. In general, this theory just preserves the frequency of the signal; so, if the shape of the waveform is desired, sampling should probably be at least ten times as fast as the signal.

31.3 Analog-to-Digital Converter Specifications

31.3.1 Range

The input range of an ADC is the span of voltages over which the ADC can make a conversion. For example, a common range for ADC is 0–5 V, meaning that the ADC can convert an input that is within 0–5 V. The end points of the low and high end of the range are called –full-scale and +full-scale (they are also referred to as rails). If the –full-scale is equal to 0 V, then the range is referred to as unipolar, and if the two full-scale values have the same magnitude, for example, −5 to +5 V, then the range of the ADC is referred to as bipolar. If an input voltage falls outside the range, the ADC is said to be overranged. In this case, most ADCs will return a value of the endpoint closest to the sampled signal.

31.3.2 Resolution

The resolution of a digitizer is the smallest detectable change in voltage; however, the resolution of an ADC usually refers to the number of binary bits it produces. For example, a 12-bit ADC represents a converted analog value, using 12 digital bits. This same 12-bit ADC can resolve a value to one of $4096 (= 2^{12})$ different levels. Another common way to specify resolution is by decimal digits. A 6-digit voltmeter measuring on a 1-V scale could measure in 0.000001 V steps from −0.999999 to 0.999999 V.

31.3.3 Coding Convention

The different formats an ADC can use to represent its output are known as coding conventions. An ADC using binary coding produces all 0s at –full-scale and all 1s at +full-scale (e.g., a 3-bit converter would produce 000 through 111).

31.3.4 Linear Errors

Linear errors are the largest and most common errors in an ADC, and are easily corrected by simple calibration or by additions and multiplications by correction constants. Although linear errors do not distort the ADC transfer function, they can change the range over which the ADC correctly operates.

31.3.5 Nonlinear Errors

Unlike linear errors, nonlinear errors are more difficult to compensate for in either the analog or digital signal. The best way to mitigate nonlinear error is to choose a well-designed, well-specified ADC. Nonlinear errors are characterized by two different specifications: differential nonlinearity (DNL) and integral nonlinearity (INL).

DNL measures any irregularity in the code width (smallest detectable change) by comparing the actual change in value to the ideal value of one code width (or 1 LSB). INL measures the deviation from an ideal transfer line of the code transitions.

Another important specification of an ADC in regards to DNL is if any codes are missing. A missing code can be thought of as a code with a width of 0 LSB (or a DNL of −1). If a code is missing, the step size at that point in the transfer function is doubled, effectively cutting the local resolution of the ADC in half. Therefore, ADC datasheets will specify if the ADC has no missing codes.

Another way to capture the same information included in INL is a measurement called relative accuracy. Relative accuracy indicates how far away fromthe ideal the code transitions are (which is INL), and also includes how far any part of the transfer function, including quantization "staircase" error, deviates fromideal. In an ideal noiseless ADC, the worse case relative accuracy is always greater than the INL.

31.3.6 Aperture Errors

Aperture errors deal with the timing of the conversions themselves. All ADCs require some signal, generally a pulse train clock, to tell the ADC when to start a conversion. Inherently, some small amount of time will elapse between the ADC receiving this convert signal and the sample being held. This amount of time is called the aperture delay. Most ADCs have an aperture delay of just a few nanoseconds. However, most measurement devices have additional circuitry in front of the ADC, such as amplifiers, which have the effect of negating the aperture delay caused by the ADC. For example, if the ADC has a delay of 10 ns and the amplifier has a delay of 160 ns, the effective aperture delay of the system is −150 ns.

Another important time specification is jitter. Jitter (or aperture jitter) measures the difference in the amount of time between each sample. If a signal is sampled at 1 million samples per second (1 MS/s), the expected period between each sample would be exactly 1μ s. Jitter caused by the clock source, digital clock circuitry, or S/H circuitry could cause this 1μ s sample time to vary by as much as a few picoseconds to a nanosecond fromcy cle to cycle.

31.3.7 Noise

Noise limits the ADC resolution because an interfering waveform is present in the input signal as it is being converted. The most common source of noise in a signal is thermal noise. Thermal noise is caused by the random nature of electrical components. With higher temperatures and resistances in components, the thermal noise will increase. Other common sources of noise are electromagnetically coupled signals from nearby components, such as logic circuits and clocks. Generally, noise is specified in volts peak-to-peak or rms, or LSBs rms or peak-to-peak.

Quantization error, resulting from the bit-resolution limitation of the ADC, is sometimes referred to as quantization noise. Although quantization error is perfectly predictable with respect to the input signal, when a signal is fairly "busy" (meaning that each consecutive conversions do not result in many common bits of data) the quantization error becomes chaotic. When this occurs, the quantization error can be thought of as another source of random noise, whose statistical distribution is uniform from −0.5 to 0.5 LSB. In spectral analysis, this is sometimes the dominant source of noise.

Once noise reaches the ADC, there are ways to process the noise out of the signal, provided that the noise is an independent signal. One of the most common ways to decrease noise in a dc measurement is to acquire a number of points and average the values. If the noise is white random noise, which has equal energy density at all frequencies, averaging will reduce the amount of noise by the square root of the number of samples averaged. If the noise is interfering with a repetitive waveform, the noise can be reduced by measuring a number of waveforms, using a level trigger and then averaging the waveforms. Most noise specifications for an ADC are for quiet, low-impedance signals. To preserve the noise performance of the ADC, the user must connect signals to the inputs with shielded cabling that keeps signals away from any electromagnetic interference.

31.3.8 Dynamic Range

Dynamic range is the ratio of largest to smallest signal the ADC can represent. The dynamic range is found by taking a full-scale signal value and comparing that to the smallest detectable noise level of the ADC. The dynamic range is usually expressed in decibels (dB) and can be found by the following formula:

$$\text{Dynamic Range} = 20\log(S/N)$$

where S is large signal level and N is noise level. The noise level includes quantization noise of the ADC, which for an ideal ADC is equal to LSB rms. A full-scale sine wave has an amplitude of $2n - 1$ LSB or (n = number of bits of the ADC) . Therefore, an ideal ADC has a dynamic range of

$$\begin{aligned}\text{Dynamic Range} &= 20\log(2^{n-1}/\sqrt{2} \times 1/\sqrt{12}) \\ &= 6.0206n + 1.7609\end{aligned}$$

Since no ADC is ideal, the effective number of bits (ENOB) of an actual ADC can be calculated using the above equation. The ENOB represents the real-world resolution of an ADC, and can be found with the following equation using dynamic range:

$$\text{ENOB} = (\text{Dynamic Range} - 1.7609)/6.0206$$

For example, a 12-bit ADC with a dynamic range of 69 dB has an ENOB of 11.17 bits.

31.3.9 Types of Analog-to-Digital Converters

All ADCs accomplish the same fundamental task of taking an analog signal and converting it into a digital representation. Two crucial characteristics of an ADC are the speed at which conversions can be made and the resolution of the conversions. Because it takes an ADC longer to resolve input signals to higher resolutions, designing or specifying an ADC often becomes a trade-off of speed versus resolution. This makes it important to understand how the ADC will be used in order to match the correct converter for the application.

Despite the many different types of ADCs available, they all share some common characteristics. The heart of any ADC converter is the comparator. A comparator is a simple 1-bit ADC that has two analog inputs and one digital output. One of the analog input signals is a reference voltage that has some known value. The other inputs will either be greater than or less than the known input value, and that will turn into a digital value of 1 or 0. Some ADCs are actually composed of multiple comparators, but the basic theory for each one is the same.

31.3.10 Flash

Flash ADCs are the fastest ADCs available, obtaining speeds in multiple gigasamples per second. However, true to the speed versus resolution trade-off discussed, the flash converters generally have resolutions of 10 bits and below. A flash converter with n bits of resolution is composed of $2n - 1$ high-speed comparators operating in parallel, see Figure 31.1. A string of $2n - 1$ resistors between two voltage references supplies a set of uniformly spaced voltages that span the input range, one for each comparator. The input voltage is then compared to each level simultaneously. The comparators then output a 1 for all voltages below the input voltage, and a 0 for all voltages above the input voltage. These resulting digital values are then fed into a logic convert to output an n-bit value.

Because of the simplicity of the design, flash converters are fast, but as the resolution of the converter is increased, the number of comparators and resistors needed increases exponentially. Both the size and power needed to operate the converter also increase exponentially as a result of increased resolution, and

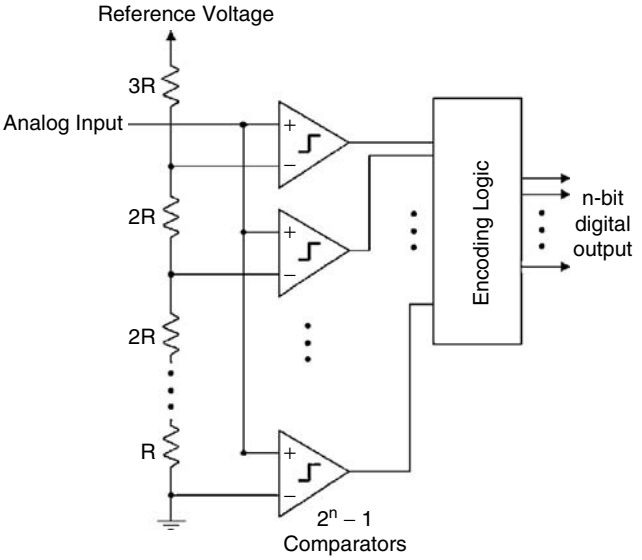


FIGURE 31.1 Flash ADC—a flash converter has $2n - 1$ comparators operating in parallel. It relies on the uniformity of the resistors for linearity.

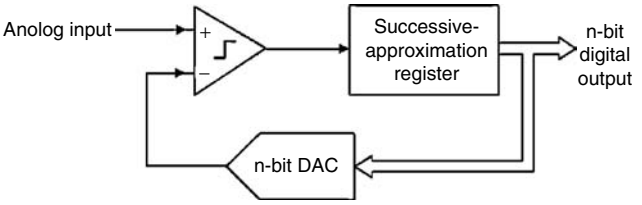


FIGURE 31.2 SAR ADC—a successive-approximation (SAR) converter has one comparator that iterates through a series of "guesses" to determine a digital representation of the signal.

this way the converters are limited in their resolution. However, because string resistors' values typically vary only a few percent fromone another, the differential linearity of the flash ADC is quite good.

31.3.11 Successive-Approximation Register

Successive-approximation register (SAR) ADCs are the most common ADCs, having resolutions of 8–18 bits and speeds up to 10 MS/s. These ADCs have a low cost, and generally have good integral linearity. The SAR ADC architecture contains a high-speed DAC in a feedback loop, see Figure 31.2. The SAR iterates the DAC through a series of levels, which are then compared to the input voltage. As the conversion progresses, the SAR builds the n-bit digital output as a result of these comparisons. When the SAR has finished, the output of the DAC is as close to the input signal as possible, and the digital input of the DAC becomes the output of the SAR ADC.

A good real-world analogy to an SAR is a balance scale. If an object of unknown mass is placed on one side and continues to test a combination of weights until the scale is balanced, the weight of the object can be obtained.

The speed of the SAR ADC is limited by the rate at which the DAC can settle inside the feedback loop. In fact, the DAC must settle n times for every n bits of resolution desired in the ADC. In order to achieve faster rates, the SAR architecture can be used as the basis for a different ADC, the multistage.

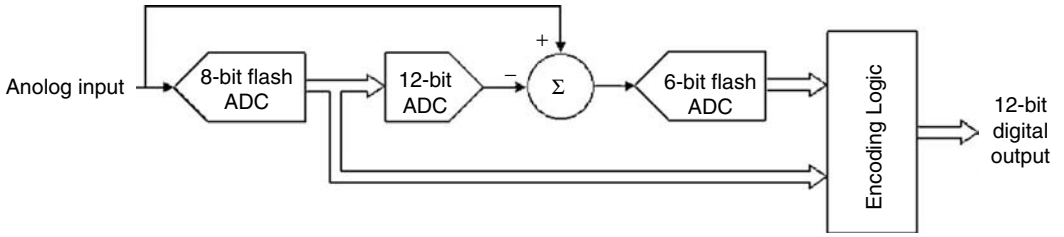


FIGURE 31.3 Multistage ADC—a multistage converter is a combination of the SAR and flash converters to provide faster sampling than the SARs and at a higher resolution than the flash converters could provide.

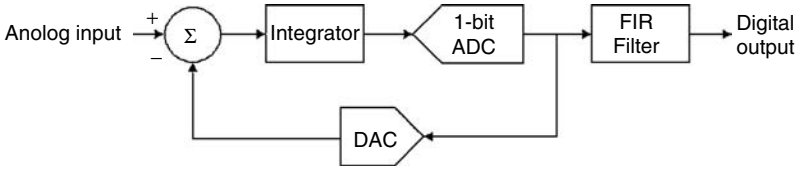


FIGURE 31.4 SDADC—asigma-delta converter uses a 1-bit comparator to determine the signal value. SD converters have great linearity by design, because the 1-bit ADC is perfectly linear, theoretically, since it can assume only one of two values.

31.3.12 Multistage

In order to achieve higher rates than the SAR, multistage ADCs use the iterative approach of the SAR but reduce the number of comparisons needed to complete the conversion. In addition to the comparator, the multistage ADC uses low-resolution flash converters, see Figure 31.3. In the figure, the 6-bit flash is used to convert the residual errors from the 8-bit flash. These two outputs fromthe ADCs are then combined using digital logic to produce a 12-bit output.

Most multistage ADCs are actually pipelined ADC. Pipelined ADCs have the same architecture as a multistage ADC, but each flash converter contains a T/H at the input. This allows each stage to convert the residual error while the previous stage has moved on to the next sample. This way the whole converter can operate at the speed of the slowest stage, as opposed to the multistage that operates at a speed equal to the sum of all the stages.

31.3.13 Integrating

Integrating ADCs are designed to return very high-resolution readings. As a trade-off, they operate at slower speeds. It is a very simple design; the integrating ADCs consist of an integrating amplifier, a comparator, a digital counter, and an extremely stable capacitor. The most common integrating ADC is the dual slope. In this architecture, the capacitor is initially discharged to have no potential across it. At a set time, the input is applied across the capacitor and it begins to charge for a set period of time T1. Because of the properties of a capacitor, the rate of charge is proportional to the input voltage. After T1, the capacitor is switched to a negative reference voltage and begins to discharge at a rate proportional to the reference. The digital counter simply measures how much time it took for the capacitor to completely discharge T2. Since T1 and the rate at which the capacitor discharges are both known values, the voltage of the input can be obtained by a simple ratio.

It is important to note that the converter is not actually measuring the input voltage itself. Instead, the ADC obtains the voltage by measuring time and using digital logic to calculate the input voltage. This method has the advantage of rejecting noise, such as periodic noise, to which other ADCs are susceptible. In addition, most integrating ADCs operate on a multiple of an AC line period (1/60 or 1/50 s) so that stray electromagnetic fields caused by power systems are cancelled.

31.3.14 Sigma Delta

The sigma-delta (SD) ADC is one of the most popular types of ADCs because of its fit on the speed versus resolution curve. SD ADCs can provide 16–24 bits of resolution at sample rates of up to hundreds of thousands of samples per second. This speed and resolution makes them ideal for certain applications such as vibration and audio analysis; however, the process of integration causes the SD ADC to have poor DC accuracy. Figure 31.4 shows the design of an SD ADC. The heart of an SD ADC is actually a 1-bit ADC that samples at incredibly high rates. Typically, these 1-bit ADCs sample at 64 or 128 times the eventual sample rate, which is a process known as oversampling. In addition to the high-speed ADC, an SD architecture consists of an analog low-pass filter and a DAC all together in a feedback loop. The result forces otherwise unavoidable quantization noise into higher frequency bands. This resulting spectrum of the noise is part of a process called noise shaping. The output of this feedback loop, which is actually just a streamof 1-bit conversions, is then fed to a digital filter. The digital filter then increases the resolution, reduces the data rate, and applies a low-pass digital filter to the data coming out of the feedback loop. After this process, the SD ADC has an output with high resolution and signals only in the frequency band of interest, eliminating most of the inherent electronic noise.

31.3.15 Digital-to-Analog Converters

The opposite of an ADC, which takes an analog value and produces a digital value, would be a device that takes digital values and creates analog values. A DAC is a device that, given a digital representation of a signal, can create an analog signal at a specific voltage level. Although much of the theory behind ADCs discussed previously applies to DACs, a unique set of terms and phenomena do exist.

31.3.16 Updating

Updating can be thought of as the DAC equivalent to sampling. If a DAC is to generate a sine wave from a group of digital values, we need some way to specify how this waveform is to be generated. Simply put, the update rate is how many points per second that a DAC can output an analog value, generally given in samples per second, kilosamples per second, or million samples per second.

31.4 Digital-to-Analog Converter Specifications

31.4.1 Range

The range of a DAC is identical to the de.nition of the range of an ADC. This refers to the voltage range of values that the DAC can output.

31.4.2 Resolution

The resolution of a DAC is specified identical to the ADC; however, the perspective is reversed. In an ADC, resolution defines how many digital bits would represent an analog value, thus giving us a level of granularity we could acquire. With a DAC, the resolution indicates how many digital bits need to be supplied to the DAC to operate and what analog signal granularity we can produce.

31.4.3 Monotonicity

One of the most useful specifications of a DAC is the monotonicity. If a DAC is monotonic, this implies that as the digital value increases, the analog output value will also increase or at least stay the same. Conversely, a device is said to be nonmonotonic if one or more values of the analog output may actually be less than the values corresponding to codes having smaller weight. Many applications are sensitive to fine changes in output value; therefore, any DAC used needs to be monotonic on all bits.

31.4.4 Settling Time and Slew Rate

Settling time and slew rate together determine how rapidly a DAC can change the analog value it is outputting. Settling time refers to the amount of time it takes the output of the DAC to reach a specified accuracy level. Most DACs specify settling time as a full-scale change in voltage, from the smallest output value to the largest. Slew rate, specified in volt per second, is the maximum rate of change of the output of the DAC. Therefore, a DAC with a fast slew rate and a small settling time can generate high-frequency signals because an accurate voltage level can be obtained in a very small amount of time.

31.4.5 Offset Error and Gain Error

Offset error refers to the transfer characteristic of the DAC not outputting an analog value of 0 when the digital value of 0 is applied. The range fromz ero to full would be offset fromthe specified value because the offset would carry throughout the transfer function. The offset error can be thought of as a translation in the transfer line either up or down fromthe ideal. Gain error indicates a linear deviation fromthe ideal transfer line of a DAC. This can be caused by a variety of factors, which results in a change of slope from the ideal.

31.4.6 Architecture of Digital-to-Analog Converters

Unlike ADCs, DACs do not implement a wide range of approaches to convert a digital input code to an analog value. Instead, almost all DACs use some combination of a switch network, resistive network, and summing amplifier. This is not to say that all DACs have the same design, but they are all based on the principle of switching.

31.4.7 Switching Network

The switching network of a DAC can be thought of as the heart of the conversion. Since digital bits are either on or off, these bits can be used to control single poll switches. These switches are then used to direct some formof analog circuitry to develop an analog value. For example, a 3-bit DAC would comprise three switches, one for each bit of input data. Depending on the code given, these switches would close in such a way to develop an analog value froma reference source that is equal to the digital representation, see Figure 31.5. Depending on the design of the analog circuitry in the DAC, the switches may be connecting current or voltage references to a resistive network.

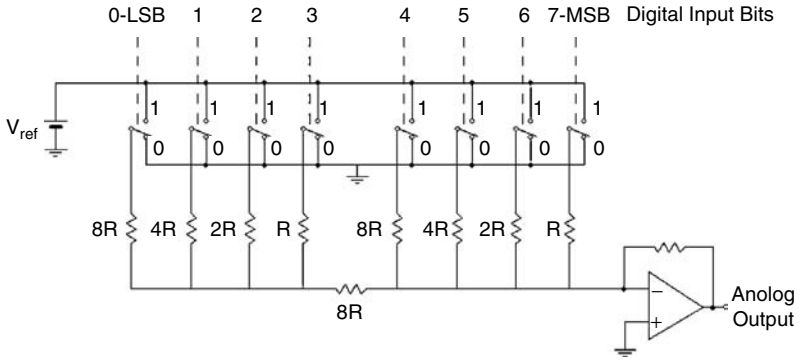


FIGURE 31.5 DAC architecture—most digital-to-analog converters (DACs) follow a standard architecture of a switch network, a resistive network, and an amplifier.

31.4.8 Resistive Networks

Resistive networks in a DAC provide the varying levels of analog output voltage, which will form the output of the DAC. Although many different resistive schemes are used in DAC design, the basic principle is common in all of them. The one shown in Figure 31.5 uses a dual resistor quad approach. In the figure, bits 0–3 and 4–7 are separated by a single resistor. These two independent groups are each a resistor quad with resistor values of 1R-2R-4R-8R where R is equal to 10 k. If any of the switches is in the 1 position, a current will develop across the resistor proportional to the resistors' value. Therefore, if bit 0 is on, a current proportional to 1/1R is generated; whereas, if switch 2 is on, a current proportional to 1/4R would be generated. The resistor between the two quads has the effect of a 16:1 current attenuator, so that even though bit 4 would generate a current proportional to 1/1R, once it gets to the amplifier it would appear to have a current proportional to 1/16R. In this case, bit 0 would be the most signi.cant bit (MSB), and bit 7 would be the least significant bit (LSB).

31.4.9 Summing Amplifier

The operational amplifier (op-amp) used in the DAC circuit of Figure 31.5 is acting as a summing amplifier. As the different bits generate a particular current, the op-amp is designed to collect the total current and would generate an output voltage. This output voltage of the op-amp is now an analog representation of the digital code that was fed to the DAC.

32

Signal Conditioning

Stephen A. Dyer
Kansas State University

Kelvin's first rule of instrumentation states, in essence, that the measuring instrument must not alter the event being measured. For the present purposes, we can consider the instrument to consist of an input transducer followed by a signal-conditioning section, which in turn drives the data-processing and display section (the remainder of the instrument). We are using the term *instrument* in the broad sense, with the understanding that it may actually be a measurement subsystem within virtually any type of system.

Certain requirements are imposed upon the transducer if it is to reproduce an event faithfully: It must exhibit amplitude linearity, phase linearity, and adequate frequency response. But it is the task of the signal conditioner to accept the output signal from the transducer and from it produce a signal in the form appropriate for introduction to the remainder of the instrument.

Analog signal conditioning can involve strictly *linear* operations, strictly *nonlinear* operations, or some combination of the two. In addition, the signal conditioner may be called upon to provide auxiliary services, such as introducing electrical isolation, providing a reference of some sort for the transducer, or producing an excitation signal for the transducer.

Important examples of linear operations include *amplitude scaling, impedance transformation, linear filtering*, and *modulation*.

A few examples of nonlinear operations include obtaining the *root-mean-square* (*rms*) *value, square root, absolute value*, or *logarithm* of the input signal.

There is a wide variety of building blocks available in either modular or integrated-circuit (IC) form for accomplishing analog signal conditioning. Such building blocks include operational amplifiers, instrumentation amplifiers, isolation amplifiers, and a plethora of nonlinear processing circuits such as comparators, analog multiplier/dividers, log/antilog amplifiers, rms-to-DC converters, and trigonometric function generators.

Also available are complete signal-conditioning subsystems consisting of various plug-in input and output modules that can be interconnected via universal backplanes that can be either chassis- or rack-mounted.

32.1 Linear Operations

Three categories of linear operations important to signal conditioning are amplitude scaling, impedance transformation, and linear filtering.

32.1.1 Amplitude Scaling

The amplitude of the signal output from a transducer must typically be scaled—either amplified or attenuated—before the signal can be processed.

32.1.1.1 Amplification

Amplification is generally accomplished by an *operational amplifier*, an *instrumentation amplifier*, or an *isolation amplifier*.

Operational Amplifiers

A conventional operational amplifier (op amp) has a differential input and a single-ended output. An *ideal* op amp, used often as a first approximation to model a real op amp, has infinite gain, infinite bandwidth, infinite differential input impedance, infinite slew rate, and infinite **common-mode rejection ratio** (**CMRR**). It also has zero output impedance, zero noise, zero bias currents, and zero input offset voltage. Real op amps, of course, fall short of the ideal in all regards.

Important parameters to consider when selecting an op amp include:

1. DC voltage gain K_0.
2. Small-signal **gain-bandwidth product** (**GBWP**) f_T, which for most op amps is $f_T \approx K_0 f_1$, where f_1 is the lower break frequency in the op amp's transfer function. The GBWP characterizes the closed-loop, high-frequency response of an op-amp circuit.
3. **Slew rate**, which governs the large-signal behavior of an op amp. Slew rates range from less than 1 V/μs to several thousand volts per microsecond.

Other parameters, such as input and output impedances, DC offset voltage, DC bias current, drift voltages and currents, noise characteristics, and so forth, must be considered when selecting an op amp for a particular application.

There are several categories of operational amplifiers. In addition to "garden-variety" op amps there are many op amps whose characteristics are optimized for one or more classes of use. Some categories of op amps include:

1. *Low-noise* op amps, which are useful in the portions of signal conditioners required to amplify very-low-level signals.
2. *Chopper-stabilized* op amps, which are useful in applications requiring extreme DC stability.
3. *Fast* op amps, which are useful when large slew rates and large GBWPs are required.
4. *Power* op amps, which are useful when currents of greater than a few mA must be provided to the op amp's load.
5. *Electrometer* op amps, which are used when very high ($>10^{13}$ **Ω**) input resistances and very low (<1 pA) input bias currents are required.

An introduction to op amps and basic circuit configurations occurs in essentially any modern text on circuit theory or electronics, and the reader can find detailed theoretical developments and many useful configurations and applications in Roberge (1975), Graeme et al. (1971), Graeme (1973, 1977), Horowitz and Hill (1989), and Stout and Kaufman (1976).

Instrumentation Amplifiers

Instrumentation amplifiers (IAs) are gain blocks optimized to provide high input impedance, low output impedance, stable gain, relatively high **common-mode rejection** (**CMR**), and relatively low offset and drift. They are well suited for amplification of outputs from various types of transducers such as strain gages, for amplification of low-level signals occurring in the presence of high-level common-mode voltages, and for situations in which some degree of isolation is needed between the transducer and the remainder of the instrument.

Although instrumentation amplifiers can be constructed from conventional op amps [a three-op-amp configuration is typically discussed; see, for example, Stout and Kaufman (1976)], they are readily available and relatively inexpensive in IC form. Some IAs have digitally programmable gains, whereas others are programmable by interconnecting resistors internal to the IA via external pins. More-basic IAs have their gains set by connecting external resistors.

Isolation Amplifiers

Isolation amplifiers are useful in applications in which a voltage or current occurring in the presence of a high common-mode voltage must be measured safely, accurately, and with a high CMR. They are also useful when safety from DC and line-frequency leakage currents must be ensured, such as in biomedical instrumentation.

The isolation amplifier can be thought of as consisting of three sections: an input stage, an output stage, and a power circuit. All isolation amplifiers have their input stages galvanically isolated from their output stages. Communication between the input and output stages is accomplished by modulation/demodulation.

An isolation amplifier is said to provide two-port isolation if there is a DC connection between its power circuit and its output stage. If its power circuit is isolated from its output stage as well as its input stage, then the amplifier is said to provide three-port isolation. Isolation impedances on the order of $10^{10}\ \Omega$ are not atypical.

Isolation amplifiers are available in modular form with either two-port or three-port isolation. Both single-channel and multichannel modules are offered.

32.1.1.2 Attenuation

Although the majority of transducers are low-level devices such as thermocouples, thermistors, resistance temperature detectors (RTDs), strain gages, and so forth, whose outputs require amplification, there are many measurement situations in which the input signal must be attenuated before introducing it to the remainder of the system.

Voltage Scaling

Most typically, the signals to be attenuated take the form of voltages. Broadly, the attenuation is accomplished by either a *voltage divider* or a *voltage transformer.*

Voltage Dividers

In many cases a simple chain divider proves adequate. The transfer function of a two-element chain of impedances $Z_1(s)$ and $Z_2(s)$ is

$$\frac{V_o(s)}{V_{\text{in}}(s)} = \frac{Z_1(s)}{Z_1(s) + Z_2(s)}$$

where the output voltage $V_o(s)$ is the voltage across $Z_1(s)$ and the input voltage V_{in} is the voltage across the two-element combination.

Of course, the impedances of the source (transducer) and the load (the remainder of the system) must be taken into account when designing the divider network.

Resistive Dividers

If the elements in the chain are resistors, then the divider is useful from DC up through the frequencies for which the impedances of the resistors have no significant reactive components. For $Z_1(s) = R_1$ and $Z_2(s) = R_2$,

$$\frac{V_o(s)}{V_{\text{in}}(s)} = \frac{R_1}{R_1 + R_2}$$

Other configurations are available for resistive dividers. One example is the Kelvin–Varley divider, which has several advantages that make it useful in situations requiring high accuracy. For a detailed description, see Gregory (1973).

Capacitive Dividers

If the elements in the chain divider are capacitors, then the divider has as its transfer function

$$\frac{V_o(s)}{V_{\text{in}}(s)} = \frac{C_2}{C_1 + C_2}$$

This form of divider is useful from low frequencies up through frequencies of several megahertz. A common application is in the scaling of large voltages.

Inductive Dividers
If the elements in the chain divider are inductors, then an autotransformer results. Inductive dividers are useful over frequencies from a few hertz to several hundred kilohertz. Errors in the parts-per-billion range are achievable.

Voltage Transformers
Voltage transformers constitute one of the most common means of accomplishing voltage scaling at line frequencies. Standard double-wound configurations are useful unless voltages above about 200 kV are to be monitored. For very high voltages, alternative configurations such as the *capacitor voltage transformer* and the *cascade voltage transformer* are employed (Gregory, 1973).

Current Scaling
Current scaling is typically accomplished via either a current shunt or a current transformer.

A *current shunt* is essentially an accurately known resistance through which the current to be measured is passed. The voltage developed across the shunt as a result of the current is the quantity measured. Shunts are useful at DC and frequencies through the audio range. Two disadvantages are (1) the shunt consumes power, and (2) the measurement circuitry must be operated at the same potential as the shunt.

The *current transformer* overcomes the mentioned disadvantages of the current shunt. Typically, the current transformer consists of a specially constructed toroidal core upon which the secondary (sense) winding is wrapped and through which the primary winding is passed. A single-turn primary is commonly used, although mutiturn primaries are available.

Other Attenuators
In addition to the aforementioned means of voltage and current scaling are attenuator pads, which provide, in addition to voltage or power reduction, the ability to be matched in impedance to the source and load circuits between which it is connected. The common pads include the T, L, and Π types, either balanced or unbalanced. Resistive attenuator pads are discussed in most textbooks on circuit design (e.g., Cuthbert, 1983). They are useful from DC through several hundred megahertz.

32.1.2 Impedance Transformation

Oftentimes the impedance of the transducer must be transformed to a value more acceptable to the remainder of the measurement system. In many cases maximum power must be transferred from the transducer's output signal to the remaining circuitry. In other cases it is sufficient to provide buffering that presents a very high impedance to the transducer, a very low impedance to the rest of the system, and a voltage gain of unity.

Matching transformers, passive matching networks such as attenuator pads, and unity-gain buffers are standard means of accomplishing impedance transformation. Unity-gain buffers are available in IC form.

32.1.3 Linear Filtering

Although, in general, digital signal processing offers many advantages over analog techniques for filtering signals, there are many relatively simple applications for which *frequency-selective analog filtering* is well suited.

Filters are used within signal conditioners (1) to reduce the effects of noise that corrupts the input signal, (2) as part of a demodulator, (3) to limit signal bandwidth, or (4) if the signal is to be sampled, to limit its bandwidth in order to prevent aliasing. These filters can be built either entirely of passive components or based on active devices such as op amps.

There are many good references that discuss methods of characterizing, specifying, and implementing frequency-selective analog filters. See Van Valkenburg (1960) for design of passive filters; for the design of active-RC filters, see Sedra and Brackett (1978) and Stephenson (1985).

32.2 Nonlinear Operations

There is a wide variety of nonlinear operations useful to signal-conditioning tasks. Listed below are some typical nonlinear blocks along with brief descriptions. Most of the blocks are available as ICs.

1. *Comparator.* A comparator is a two-input device whose output voltage, V_o, takes on one of two stable values, V_{o0} and V_{o1}, as follows:

$$V_o = \begin{cases} V_{o0}, & \text{if } V_2 < V_1 \\ V_{o1}, & \text{otherwise} \end{cases}$$

   where V_1 and V_2 are the voltages at the two inputs.
2. *Schmitt trigger.* A Schmitt trigger is a comparator with hysteresis. It can be constructed from a comparator by applying positive feedback.
3. *Multiplier.* A two-input multiplier supplies an output voltage that is proportional to the product of its input voltages.
4. *Divider.* A two-input divider has as its output a voltage proportional to the ratio of its input voltages. The functions of multiplication and division are usually combined within a single device.
5. *Squarer.* A squarer has as its output a voltage proportional to the square of its input. Squarers can be constructed by a number of means: from multipliers, based on diode-resistor networks, based on FETs, and so forth.
6. *Square-rooter.* A square-rooter has as its output a voltage proportional to the square root of its input. A square-rooter can be built most easily from either a divider or a log/antilog amplifier.
7. *Logarithmic/antilogarithmic amplifier.* A log/antilog amplifier produces an output voltage proportional to the logarithm or the antilogarithm of its input voltage.
8. *True RMS-to-DC converter.* A true RMS-to-DC converter computes the square root of the average, over some interval of time, of the instantaneous square of the input signal. The averaging operation is generally accomplished via a simple low-pass filter whose capacitor is selected to give the desired interval.
9. *Trigonometric function generator.* Generators are available in IC form that produce as their outputs any of the standard trigonometric functions or their inverses, taken as functions of the differential voltage at the generator's inputs.
10. *Sample-and-hold and track-and-hold amplifiers.* A sample-and-hold amplifier (SHA) is a device that samples the signal at its input and holds the instantaneous value whenever commanded by a logic control signal. A track-and-hold amplifier is identical to an SHA but is used in applications where it spends most of its time tracking the input signal (i.e., in "sample" or "track" mode), in contrast to the SHA, which spends most of its time in "hold" mode.
11. *Precision diode-based circuits.* Circuits such as precision half-wave rectifiers, absolute-value circuits, precision peak detectors, and precision limiters are relatively easy to design and implement based on diodes and op amps. See Horowitz and Hill (1989), Stout and Kaufman (1976), and Graeme (1977).

A detailed description of these and other nonlinear circuit blocks can be found in Sheingold (1976).

Example

We provide briefly an example of a device that has embedded within it several signal-conditioning circuits. Figure 32.1 shows the basic block diagram of a therapeutic ultrasound unit, which finds widespread use in physical medicine.

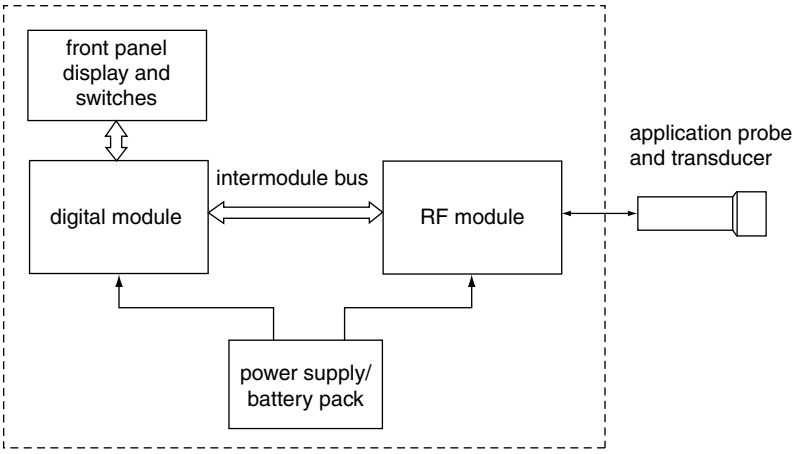


FIGURE 32.1 Basic block diagram of the therapeutic ultrasound unit discussed as an example.

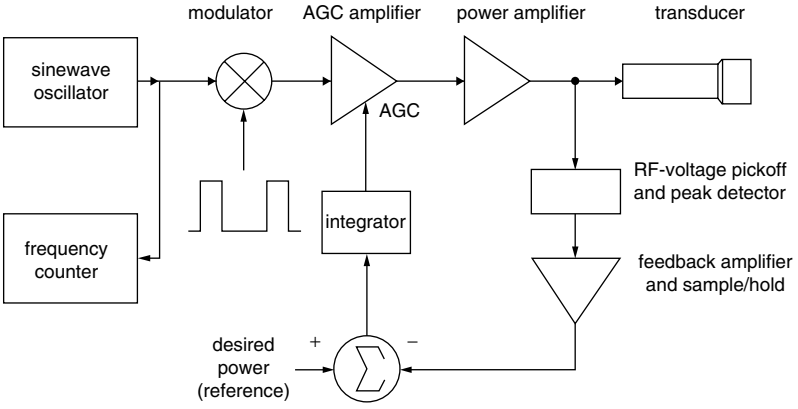


FIGURE 32.2 Simplified block diagram of the RF module used in the ultrasound unit of Figure 32.1.

The particular unit being discussed consists of five principal subsystems:

1. An application probe and ultrasound transducer, which imparts ultrasonic energy to the tissue being treated. *Note that this transducer is NOT an input transducer such as has been discussed in relation to signal conditioners.*
2. A radio-frequency (RF) module, which provides electrical excitation to the ultrasound transducer.
3. Front-panel display and switches, which allow communication between the unit and its operator.
4. A microprocessor-based digital module, which orchestrates the overall control of the ultrasound unit.
5. A power supply/battery pack, which provides operating power to the unit.

We focus now on the RF module, whose basic block diagram is shown in Figure 32.2. The module consists of a sine-wave oscillator that produces a signal at the resonant frequency of the transducer, a modulator that allows that signal to be pulse-modulated, and an amplifier with RF-voltage feedback. Incorporated in the amplifier are a power amplifier capable of driving the transducer and automatic-gain-control (AGC) circuitry required to adjust the output power to coincide with that selected by the operator. The AGC uses

a standard feedback-control loop to maintain a constant-voltage envelope on the RF signal output from the power amplifier.

Some of the signal conditioners employed within the RF module include the following:

1. The RF-voltage pickoff at the output of the power amplifier. The pickoff employs a half-wave rectifier, followed by a simple capacitive chain divider for voltage scaling.
2. A precision peak detector, which obtains the peak value of the output from the voltage divider during a modulation cycle and presents that value to the feedback loop.
3. An amplifier, having digitally selectable gain, which amplifies the output of the peak detector.
4. A sample-and-hold amplifier, used to hold the amplified output from the peak detector during the "off-time" of the modulator. The SHA is needed since the time constant of the peak detector is not sufficient to prevent significant "droop" during the off-time of the modulator.
5. An integrator (an example of frequency-selective filtering), which develops the control voltage for the AGC loop from the output of the differencer.
6. A current shunt, not shown in Figure 32.2, which is used to monitor the DC current supplied to the power amplifier.

As can be seen from this simple example, several signal-conditioning functions may be employed within a single system, and the system itself might not even be an instrument!

Defining Terms

Common-mode rejection (CMR): CMRR given in decibels. CMR = 20 log|CMRR|. CMR is a nonlinear function of common-mode voltage and depends on other factors such as temperature.

Common-mode rejection ratio (CMRR): The ratio of the differential gain to the common-mode gain of an amplifier.

Gain-bandwidth product (GBWP): The product of an amplifier's highest gain and its corresponding bandwidth. Used as a rough figure of merit for bandwidth.

Slew rate: The maximum attainable time rate of change of an amplifier's output voltage in response to a large step change in input voltage.

References

Cuthbert, T. R. 1983. *Circuit Design Using Personal Computers.* John Wiley & Sons, New York.

Graeme, J. G. 1973. *Applications of Operational Amplifiers.* McGraw-Hill, New York.

Graeme, J. G. 1977. *Designing with Operational Amplifiers.* McGraw-Hill, New York.

Graeme, J. G., Tobey, G. E., and Huelsman, L. P. (Ed.) 1971. *Operational Amplifiers.* McGraw-Hill, New York.

Gregory, B. A. 1973. *An Introduction to Electrical Instrumentation.* Macmillan, London.

Horowitz, P. and Hill, W. 1989. *The Art of Electronics,* 2nd ed. Cambridge University Press, New York.

Roberge, J. K. 1975. *Operational Amplifiers.* John Wiley & Sons, New York.

Sedra, A. S. and Brackett, P. O. 1978. *Filter Theory and Design: Active and Passive.* Matrix, Beaverton, OR.

Sheingold, D. H. (Ed.) 1976. *Nonlinear Circuits Handbook.* Analog Devices, Norwood, MA.

Stephenson, F. W. 1985. *RC Active Filter Design Handbook.* John Wiley & Sons, New York.

Stout, D. F. and Kaufman, M. (Ed.) 1976. *Handbook of Operational Amplifier Circuit Design.* McGraw-Hill, New York.

Van Valkenburg, M. E. 1960. *Introduction to Modern Network Synthesis.* John Wiley & Sons, New York.

Further Information

IEEE Transactions on Instrumentation and Measurement. Published bimonthly by the Institute of Electrical and Electronics Engineers.

IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing. Published monthly by the Institute of Electrical and Electronics Engineers.

The Best of Analog Dialogue, *1967–1991*. 1991. Analog Devices, Norwood, MA. A collection of practical articles covering circuits, systems, and software for signal processing.

Analog Devices Special Linear Reference Manual and *Analog Devices Amplifier Reference Manual.* Presents an extensive selection of ICs, modules, and subsystems for signal conditioning.

Pallás-Areny, R. and Webster, J. G. 1991. *Sensors and Signal Conditioning.* John Wiley & Sons, New York. Provides an excellent introduction to sensors and signal-conditioning circuits required by them.

Sheingold, D. H. (Ed.) 1980. *Transducer Interfacing Handbook*. Analog Devices, Norwood, MA. Covers signal-conditioning techniques applicable to temperature, pressure, force, level, and flow transducers.

33

Virtual Instrumentation Systems

Darcy Dement
National Instruments, Inc.

33.1 Introduction to Instrumentation

Like instrumentation systems of the past, today's instrumentation systems build on widely accepted contemporary technologies. In the early days of instrumentation, the jeweled movement of clocks in the nineteenth century was first adapted to build analog meters. In the 1930s, when engineers began to accept the variable capacitor, variable resistor, and vacuum tubes as pieces of the radio, the first electronic instruments were introduced using the same components. As display technologies improved for use on the first televisions, oscilloscopes and analyzers began using the same technology to display the user's measurements. These first steps of integrating relevant technologies into instrumentation set the stage for today's virtual instrumentation.

33.2 Approaches to Instrumentation: Virtual and Traditional

Fundamentally, there are two types of instrumentation today: virtual and traditional. Figure 33.1 illustrates the architectures of these approaches.

The diagrams show the similarities in these two approaches. Both have measurement hardware, a chassis, a power supply, a bus, a processor, an OS, and a user interface. Because the approaches use the same basic components, the most obvious difference from a purely hardware standpoint is how the components are packaged. A traditional, or stand-alone, instrument puts all the components in the same

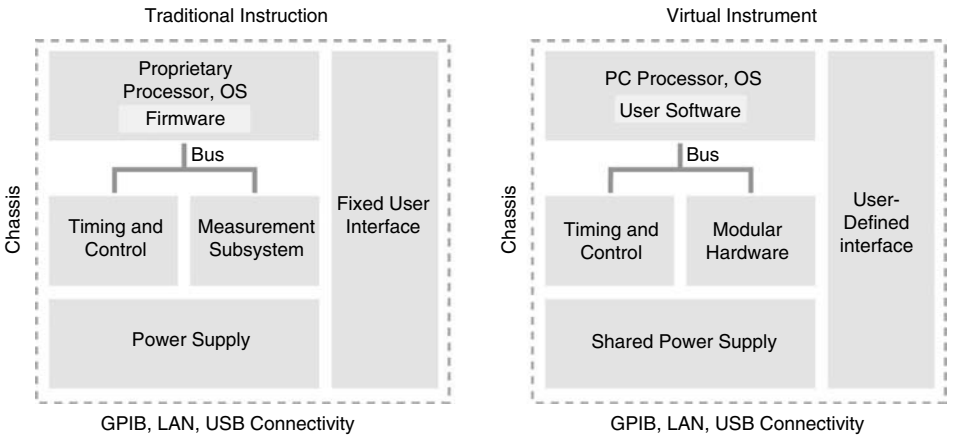


FIGURE 33.1 Traditional versus virtual instrumentation architectures. Both share similar hardware components; the primary difference between the architectures is where the software resides and whether it is user accessible.

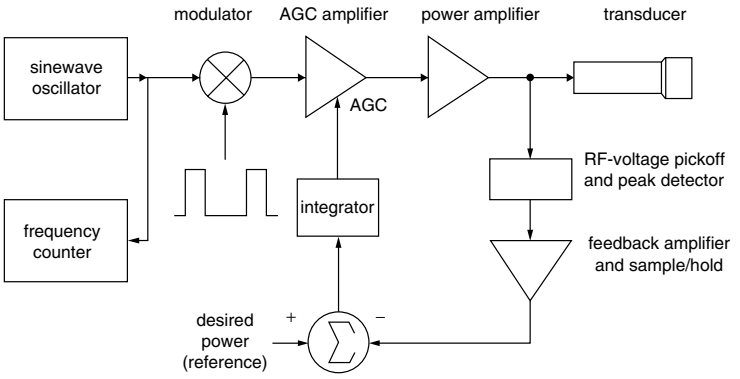


FIGURE 33.2 Example of a virtual instrumentation system using PXI hardware and National Instruments LabVIEW graphical development software.

box for every discrete instrument. When automated, a traditional instrument is often controlled with GPIB, USB, or LAN/Ethernet. While there are a large number of traditional instruments, the software and user interface are fixed and can only be updated when and how the vendor chooses (e.g., through a firmware update). This makes it impossible for the user to perform measurements not included in the traditional instrument's function list, and it makes it challenging to perform measurements for new standards or to modify the system if needs change.

By contrast, a virtual instrument makes the raw data from the hardware available to the user to define their own measurements and user interface. This software-defined approach lets users make custom measurements, perform measurements for emerging standards, or modify the system if requirements change (e.g., to add instruments, channels, or measurements).

A virtual instrument can take several forms. Referring again to the architecture of a virtual instrumentation system in Figure 33.1, many of the components are shared across instrument modules—such as the chassis and power supply—instead of duplicating these components for every instrument function. These instrument modules can also include different types of hardware, including oscilloscopes, function generators, digital, and RF. In some cases, such as PXI (PCI eXtensions for Instrumentation)—a rugged platform for test, measurement, and control supported by over 70 member companies—the measurement hardware is housed in an industrial chassis (see Figure 33.2).

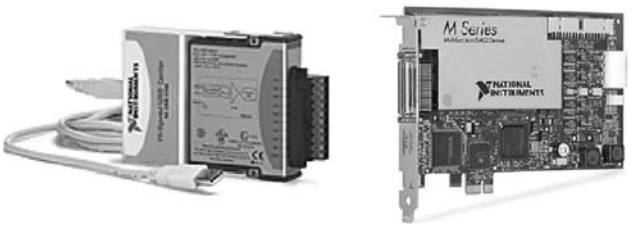


FIGURE 33.3 Examples of other measurement hardware choices for virtual instrumentation. On the left is a USB peripheral module and on the right, a PCI Express plug-in module.

FIGURE 33.4 Virtual instrumentation implementation using a traditional instrument as the measurement hardware.

In those cases, the host computer can be embedded in the chassis (as shown in Figure 33.2) or it can be a separate desktop PC or server-class machine that controls the measurement hardware through a cable. In other cases, as shown in Figure 33.3, the measurement hardware is simply a peripheral that is installed in one of the host computer's peripheral ports or peripheral slots.

A virtual instrument can even use a traditional instrument as the measurement hardware (see Figure 33.4). While modular hardware provides other benefits described in the next section, fundamentally, a virtual instrument system can use any type of instrument as the measurement hardware, provided the user has access to the raw data to define custom measurements. A virtual instrumentation system can also use different measurement hardware—whether PXI, USB, plug-in peripheral, or traditional instrument—in the same system. (This is termed a "hybrid" system.) Strictly speaking, a virtual instrumentation system is defined as follows:

- Virtual instrument system (*n.*): Software-defined system, where software based on user requirements defines the functionality of generic measurement hardware
- Virtual instrument (*n.*): Diminutive form of a virtual instrument system

By any definition, the measurement software that resides on the host computer is fundamental to a virtual instrument. This user-defined software is what converts the raw data into the measurement of interest. It could be a common task such as a simple voltage measurement or a fast Fourier transform, or it could be a custom application such as a flyback measurement, octave analysis, or brain activity mapping.

Note that while the term "computer based" is often applied to virtual instruments, ultimately many modern traditional instruments are also computer based. Modern traditional instruments have all the same components of a PC, including a processor, memory, and a bus to move data internally, as shown in Figure 33.1. Thus, the distinction between virtual and traditional instruments is determined by whether the software is exposed to the user for custom measurements and tasks.

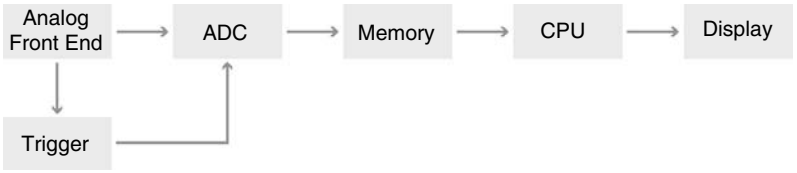


FIGURE 33.5 Typical digitizer architecture. In a virtual instrument implementation, most of the memory, the CPU, and the display are provided by the host PC.

33.3 Modular Hardware for System Scalability

Modular hardware uses commercial semiconductor, processor, and bus technologies to create virtual instruments with high performance at a low cost. The explosion of widely used commercial technologies such as analog-to-digital converters (ADCs), digital-to-analog converters (DACs), field-programmable gate arrays (FPGAs), digital signal processors (DSPs) has resulted in rapid growth of functionality and performance in modular I/O.

The most common instruments in use today digitize an analog signal into bits that can be interpreted by a processor, for example, digitizing a high-frequency signal (as in the case of an oscilloscope) or digitizing a slower signal with higher resolution (as with a digital multimeter). A greatly simplified digitizer architecture is shown in Figure 33.5.

The analog input path attenuates, amplifies, filters, and/or couples the signal to optimize the digitization by the ADC. (For some applications—e.g., with many physical measurements via transducers—additional signal conditioning is housed externally and connected to the digitizer via cable or direct mount.) The ADC samples the conditioned waveform and converts the analog input signal to digital values that represent the conditioned input signal. The data is then stored in onboard memory, where it is transferred across a bus (e.g., PCI Express) to the processor. (In some applications, there can be very little onboard memory if the bus throughput is high enough to stream the data continuously from the measurement hardware to the host PC.) Then, the software (in the CPU for a virtual instrument) makes the proper measurement and presents the data to the user interface, sends it to a database across the network, publishes it to the Internet, or whatever the instrument user chooses.

Analog input is presented here as an example, but many other measurement types can be inferred from this case. For example, engineers can build signal generation (analog output), digital I/O, counter/timers, RF, switching, and other instrument functions using the same virtual approach with different front-end architectures. Because the system is software defined, engineers can combine any of these elements into an integrated system with very little additional work—a unique property of virtual instrumentation. This property also means that the system developer can add new instruments, measurements, or channels over time as requirements change—a concept referred to as "scalability." Although not discussed in detail here, these modular I/O components also typically include timing and synchronization resources for sharing clocks, triggers, and events between hardware I/O modules. Engineers can use these shared resources to create a 1000-channel digitizer, a mixed-signal system with 10 ps skew between modules, or a fast, hardware-handshaked digital multimeter/switch system.

33.4 Software for Flexible, Custom Measurements

The role of software in virtual instrumentation cannot be overstated. Software converts the raw bitstream from hardware into a useful measurement. A well-architected virtual instrumentation system considers multiple layers of software, including I/O drivers, application development, and test management, as shown in Figure 33.6.

The bottom layer, measurement and control services, is one of the most crucial elements of a virtual instrumentation system, though often overlooked. This layer represents the driver I/O software and

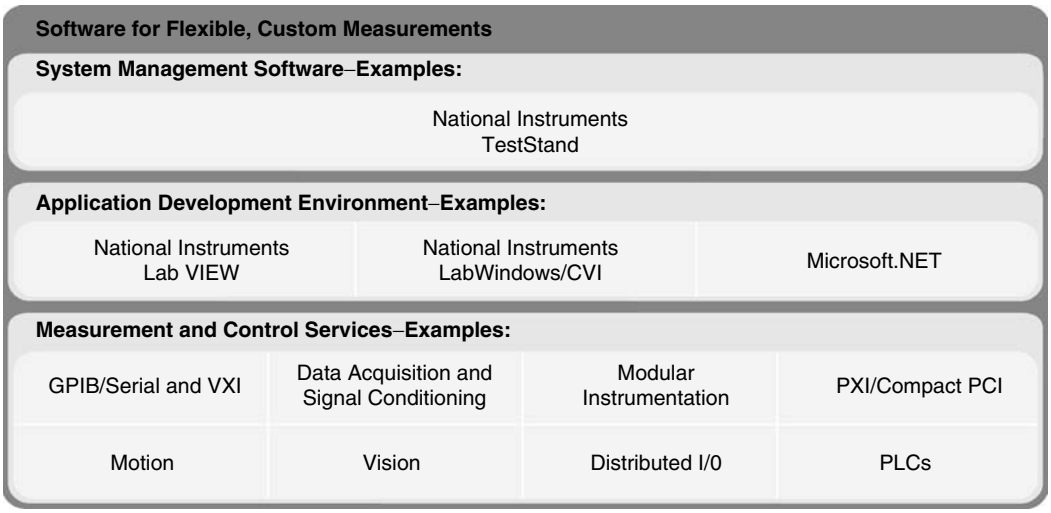


FIGURE 33.6 Software layers in a virtual instrumentation system.

hardware configuration tools. This driver software is critical, because it provides the connectivity between the test development software and the hardware for measurement and control.

Instrument drivers provide a set of high-level, human-readable functions for interfacing with instruments. Each instrument driver is specifically tailored to a particular model of instrument to provide an interface to its unique capabilities. Of particular importance in an instrument driver is its integration with the development environment so that the instrument commands are a seamless part of the application development. System developers need instrument driver interfaces optimized for their development environment of choice, for example, NI LabVIEW, C, C++, or Microsoft .NET.

Also represented in measurement and control services are configuration tools. These configuration tools include resources for configuring and testing I/O, as well as storing scaling, calibration, and channel aliasing information. These tools are important for quickly building, troubleshooting, and maintaining an instrumentation system.

The software in the application development environment layer provides the tools to develop the code or procedure for the application. Although graphical programming is not a requirement of a virtual instrument system, these systems often use graphical tools for their ease of use and rapid development time. Graphical programming uses "icons" or symbolic functions that pictorially represent the action to be performed, as shown in Figure 33.7. These symbols are connected together through "wires" that pass data and determine order of execution. NI LabVIEW provides the industry's most used and most complete graphical development environment.

Some applications also require an additional layer of software management, either for test execution or visibility into test data. These are represented in the system management software layer. For highly automated test systems, test management software provides a framework for sequencing, branching/looping, report generation, and database integration. The test management toolmust also provide tight integration into the development environments where the application-specific code is created. National Instruments TestStand, for example, provides this framework for sequencing, branching, report generation, and database integration and includes connectivity to all common development environments. In other applications that need visibility into large quantities of test data, other tools may be useful. These needs include quick access to large volumes of scattered data, consistent reporting, and data visualization. These software tools aid in managing, analyzing, and reporting data collected during data acquisition and/or generated during simulations.

Every layer of this software architecture should be considered for a virtual instrumentation system.

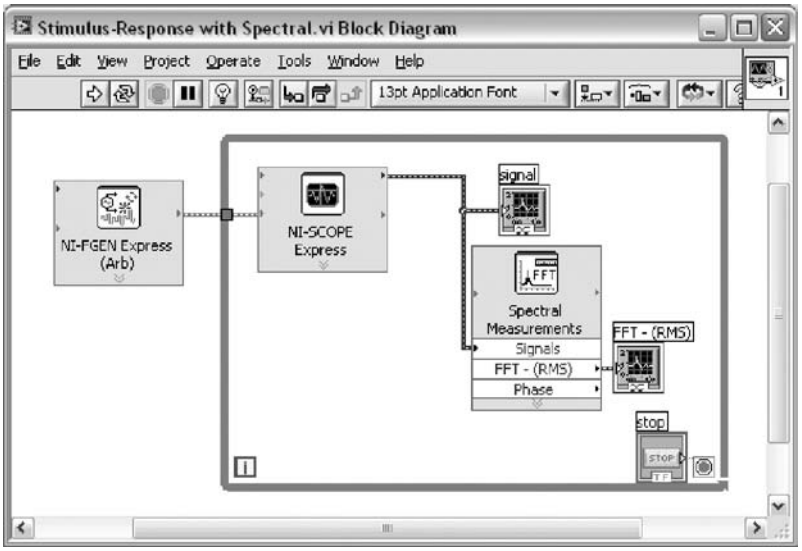


FIGURE 33.7 Code for a typical stimulus/response application using virtual instrumentation. This code is written in National Instruments LabVIEW for (1) generating a signal from an arbitrary waveform generator; (2) acquiring the signal with a digitizer/oscilloscope; (3) performing a fast Fourier transform; and (4) graphing the result of the FFT on the user interface (front panel).

33.5 Extensions beyond Test and Measurement

While virtual instrumentation has roots in test and measurement, these concepts can extend to industrial control and design prototyping. In fact, engineers and scientists who adopt a virtual instrumentation approach see large efficiency gains from using this approach across different types of applications.

In control and industrial applications, PCs and PLCs both play important roles. PCs bring great software flexibility and capability, while PLCs deliver outstanding ruggedness and reliability. A programmable automation controller (PAC) is an example of how virtual instrument techniques can be applied to industrial control; a PAC combines elements of both PCs and PLCs to accommodate more complex, dynamic, adaptive, and algorithm-based control while retaining its ruggedness and reliability. Other industrial-focused hardware might include logic, motion, machine vision, and other functions. In all these cases, the software environment can be the same one used for a virtual instrumentation system for test and measurement, bringing great efficiency gains to applications combining test, measurement, and industrial control.

For design prototyping, virtual instrumentation technology offers an architecture for designing custom measurement hardware circuitry using reconfigurable FPGA chips and NI LabVIEW graphical development tools. This architecture facilitates the customization of data acquisition, communication, and control hardware using the same software tools for test and measurement or industrial control. These users can quickly create custom hardware circuitry with high-performance I/O and unprecedented flexibility in system timing control without actually building custom circuitry.

33.6 Benefits of Virtual Instrumentation Systems

The software-defined nature of virtual instrumentation brings several benefits to engineers and scientists, which are discussed in the following sections.

33.6.1 Adaptability and Longevity

Only with virtual instrumentation can engineers and scientists create the user-defined instruments that nimbly adapt to a changing design or a multifunctional device under test. As the device under test gains more features and new measurements or tasks are added, only a software-defined architecture can adapt at the speed of the device under test itself. Because virtual instrumentation uses highly productive software, modular I/O, and commercial platforms, it is uniquely positioned to keep pace with the product development rate. Engineers can modify the software to add new functionality or support new devices or tests, and the modular I/O hardware can be rapidly combined in any order or quantity to performany I/O task required. These same properties can accommodate a wide variety of test, measurement, or control tasks required for a single device or process, or disparate tasks across multiple devices.

33.6.2 Greater Speed

Virtual instrumentation applications that use modular I/O are capable of high-speed measurement and high throughput to PCmemory because of the use of bus and processor technologies. The PCI Express bus, for example, is commonly available today with a throughput of 1 GB/s—1000 times faster than the GPIB bus used to connect most traditional instruments to a PC, 16 times faster than USB 2.0, 80 times faster than 100-Mbps Ethernet, and even 8 times as fast as emerging Gbps Ethernet. For high-streaming applications, PCI Express slots are available today with 4 GB/s bandwidth, and plans exist for higher throughputs in the future. In virtual instrumentation systems, GHz PC processors analyze the data and make measurements using software. The result is measurements at 10–100 times the throughput of a test system built solely on traditional instruments that use built-in vendor-defined firmware and application-specific processors.

33.6.3 Lower Cost and Size

A virtual instrumentation system using modular I/O shares resources across the instruments, such as the processor, the bus, and the memory (see Figure 33.1). By sharing these resources across the instruments in the system, the integrated solution is much smaller and much less expensive than the traditional solution. The benefits of greater throughput, smaller size, and the ability to only change or upgrade specific components of the system when necessary also lead to lower total cost of ownership.

It is worth noting that these benefits do not sacrifice measurement performance. Today, the instruments designed with a virtual instrumentation approach include the industry's highest-resolution digitizer, the highest-bandwidth arbitrary waveform generator, and the most accurate 71/2-digit digital multimeter.

Further Reading

Allan Hambley, *Electrical Engineering: Principles and Applications*, 3rd Edn., Upper Saddle River, NJ: Prentice Hall (copyright 2005).

Gary W. Johnson and Richard Jennings, *LabVIEW Graphical Programming*, 4th Edn., New York:McGraw-Hill (copyright 2007).

David G. Alciatore and Michael B. Histand, *Introduction to Mechatronics and Measurement Systems*, 3rd Edn., New York: McGraw-Hill (copyright 2007).

34

Software Design and Development*

Margaret H. Hamilton
Hamilton Technologies, Inc.

A software-based system can be neatly compared with a biological entity called a superorganism. Comprising software, hardware, peopleware and their interconnectivity (such as the Internet), and requiring all to survive, the silicon superorganism is itself a part of a larger superorganism—for example, a medical system including patients, drugs, drug companies, doctors, hospitals, and health care centers; a space mission including the spacecraft, the laws of the universe, mission control, and the astronauts; a system for researching genes including funding organizations, funds, researchers, research subjects, and genes; a financial system including investors, money, politics, financial institutions, stock markets, and the health of the world economy; or it could be just the business itself.

Whether that business be government, academic, or commercial, the software-based system, like its biological counterpart, must grow and adapt to meet rapidly changing requirements. And, like other organisms, the business has both physical infrastructure and operational policies, which guide and occasionally constrain its direction and the rate of evolution, which it can tolerate without becoming dysfunctional.

Compared to a biological superorganism, which may take many generations to effect even a minor hereditary modification, software can be modified immediately. This makes it far superior in this respect to the biological entity in terms of its evolutionary adaptability. Continuity of business rules and/or the physical infrastructure provides a natural tension between "how fast the software can change" and "how rapidly the overall system can accept change." Software, the brain of the silicon superorganism, controls the action of the entire entity. Keep in mind, however, it was a human being that created the software.

In this chapter we will discuss the tenets of software, what it is and how it is developed, as well as the precepts of software engineering, which are the methodologies by which ideas are turned into software.[†]

* Parts of this chapter were taken from *Object Thinking: Development Before the Fact*, M. H. Hamilton and W. R. Hackler, in press.

[†]001, 001 Tool Suite, DBTF, Development Before the Fact, SOO, and System Oriented Objects are all trademarks of Hamilton Technologies, Inc.

34.1 The Notion of Software

Software is the embodiment of logical processes, whether in support of business functions or in control of physical devices. The nature of software as an instantiation of process can apply very broadly, when modeling complex organizations, or very narrowly as when implementing a discrete numerical algorithm. In the former case, there can be significant linkages between reengineering businesses to accelerate the rate of evolution—even to the point of building operational models, which then transition into application suites and thence into narrowly focused implementations of algorithms, as above. Software thus has a potentially wide range of application, and that well designed has a potentially long period of utilization.

While some would define software as solely the code that a programming language generates from the compilation process, a broader and more precise definition includes requirements, specifications, designs, program listings, documentation, procedures, rules, measurements, and data as well as the tools used to create, test, optimize, and implement the software.

That there is more than one definition of software is a direct result of the confusion about the very process of software development itself. A 1991 study by the Software Engineering Institute (SEI) [1] amplifies this rather startling problem. SEI developed a methodology for classifying an organization's "software process maturity" into one of five levels which range from Level 1, the initial level (where there is no formalization of the software process), to Level 5, the optimizing level where methods, procedures, and metrics are in place with a focus toward continuous improvement in software reliability. The result of this study showed that fully 86% of organizations surveyed in the United States were at Level 1 where the terms "ad-hoc," "dependent on heroes," and "chaotic" are commonly applied. And, given the complexity of today's Internet-based applications, it would not be surprising to see the percentage of Level 1 organizations increase.

Creating order from this chaos requires an insightful understanding into the component parts of software as well as the development process. Borrowing again from the world of natural science, an entelechy is something complex that emerges when a large number of simple objects are put together. For example, one molecule of water is rather boring in its utter lack of activity. But pour a bunch of these molecules into a glass and there is a ring of ripples on the water's surface. If many of these molecules are combined, the result is an ocean. So too software. By itself, a line of code is a rather simple item. But combine many lines and the result is a complex program. Add additional programs and the result could be a system that can put a person on the moon.

Although the whole is indeed bigger than the sum of its parts, one must still understand those parts if the whole is to work in an orderly and controlled fashion. Like a physical entity, software can "wear" as a result of maintenance, changes in the underlying system, and updates made to accommodate the requirements of the ongoing user community. Entropy is a significant phenomenon in software, especially for Level 1 organizations.

Software at the lowest programming level is termed source code. This differs from executable code (i.e., which can be executed by the hardware to perform one or more specified functions) in that software is written in one or more programming languages and cannot, by itself, be executed by the hardware. A programming language is a set of words, letters, numerals, and abbreviated mnemonics, regulated by a specific syntax, used to describe a program to a computer. There are a wide variety of programming languages, many of them tailored for a specific type of application. C, one of today's more popular programming languages, is used in engineering as well as business environments while object-oriented languages such as C++ [2] and Smalltalk have been gaining acceptance in both of these environments. More recently, Java [3] has been gaining acceptance. In fact, it has become a language of choice for Internet-based applications. In the recent past, engineering applications have often used programming languages such as FORTRAN, HAL (or HAL/s) for NASA space applications, and Ada for government applications while commercial business applications have favored COBOL (COmmon Business Oriented Language). For the most part one finds that in any given organization there are no prescribed rules, other than those related to what is most popular at the moment, which dictate which languages are to be used. And, as one might expect, a wide diversity of languages is being deployed.

The programming language, whether it be C++, Java, Visual BASIC, C, FORTRAN, HAL/s, COBOL, or something else, provides the capability to code such logical constructs as that having to do with:

- *User Interface.* Provides a mechanism whereby the ultimate end-user can input, view, manipulate, and query information contained in an organization's computer systems. Studies have shown that productivity increases dramatically when visual user interfaces are provided. Known as GUIs (graphical user interfaces), each operating system provides its own variation. Some common graphical standards are Motif for UNIX systems and Microsoft Windows for PC-based systems.
- *Model Calculations.* Perform the calculations or algorithms (step-by-step procedures for solving a problem) intended by a program, e.g., process control, payroll calculations, or a Kalman filter.
- *Program Control.* Exerts control in the form of comparisons, branching, calling other programs, and iteration to carry out the logic of the program.
- *Message Processing.* There are several varieties of message processing. Help-message processing is the construct by which the program responds to requests for help from the end-user. Error-message processing is the automatic capability of the program to notify and then recover from an error during input, output, calculations, reporting, communications, etc. And, in object-oriented development environments, message processing implies the ability of program objects to pass information to other program objects.
- *Moving Data.* Programs store data in a data structure. Data can be moved between data structures within a program, moved from an external database or file to an internal data structure or from user input to a program's internal data structure. Alternatively, data can be moved from an internal data structure to a database or even to the user interface of an end-user. Sorting and formatting are data moving operations used to prepare the data for further operations.
- *Database.* A collection of data (objects[1]) or information about a subject or related subjects, or a system (for example, an engine in a truck or a personnel department in an organization). A database can include objects, such as forms and reports or a set of facts about the system (for example, the information in the personnel department needed about the employees in the company). A database is organized in such a way so as to be easily accessible to computer users. Its data is a representation of facts, concepts, or instructions in a manner suitable for processing by computers. It can be displayed, updated, queried, printed, and reports can be produced from it. A database can organize data in several ways including in a relational, hierarchical, network, or object-oriented format.
- *Data Declaration.* It describes data and data structures to a program. An example would be associating a particular data structure with its type (for example, data about a particular employee might be of type person).
- *Object.* A person, place, or thing, which could be physical or abstract. An object contains other more primitive objects (or data) and a set of operations to manipulate objects (or data). When brought to life, it knows things (called attributes) and can do things (to change itself or interact with other objects). For example, in a robotics system a robot object may contain the functions to move its own armature to the right, while it is coordinating with another robot to transfer yet another object. Objects can communicate with each other through a communications medium (e.g., message passing, radio waves, Internet).
- *Real-Time.* A software system that satisfies critical timing requirements. The correctness of the software depends on the results of computation, as well as on the time at which the results are produced. Real-time systems can have varied requirements such as performing a task within a specific deadline and processing data in connection with another process outside of the computer.

*Data and object are used interchangeably throughout this chapter to define information in a software program.

Applications such as transaction processing, avionics, interactive office management, automobile systems, and video games are examples of real-time systems.

- *Distributed.* Any system in which a number of independent, interconnected processes can cooperate. The client/server model is one of the most popular forms of distribution in use today. In this model, a client initiates a distributed activity and a server carries out that activity.
- *Simulation.* The representation of selected characteristics of the behavior of one physical or abstract system by another system. For example, a software program can simulate an airplane or an organization or another software program.
- *Documentation.* It includes the description of requirements, specification, and design as well as written or generated documentation, which describe how each program within the larger system operates and can be used; and comments which describe the operation of the program that are stored internally in the program.
- *Tools.* The software programs used to design, develop, test, analyze, or maintain system designs or another software program and its documentation. They include code generators, compilers, editors, database management systems (DBMS), GUI builders, debuggers, operating systems, and software development and systems engineering tools referred to in the 1990s as computer-aided software engineering (CASE) tools and now often referred to as life-cycle design or life-cycle development environments, which combine a set of tools, including some of those listed above.

Although the reader should by now understand the dynamics of a line of source code, where that line of source code fits into the superorganism of software is dependent upon many variables. This includes the industry the reader hails from as well as the software development paradigm used by the organization.

As a base unit, a line of code can be joined with other lines of code to form many things. In a traditional software environment many lines of code form a program, sometimes referred to as an application program or just plain application. But lines of source code by themselves cannot be executed. First, source code must be run through what is called a compiler to create an object code. Next, the object code is run through a linker which is used to construct an executable code. Compilers are programs themselves. Their function is twofold. The compiler first checks the source code for obvious syntax errors and then, if it finds none, creates object code for a specific operating system. UNIX, Linux (a spinoff of UNIX), and NT are all examples of operating systems. An operating system can be thought of as a supervising program that controls the application programs that run under its control. Since operating systems (as well as computer architectures) can be different from each other, the object code resulting from the source code compiled for one operating system cannot be executed under a different kind of operating system—without a recompilation.

Solving a complex business or engineering problem often requires more than one program. One or more programs that run in tandem to solve a common problem is known collectively as a system. The more modern technique of object-oriented development dispenses with the notion of the program altogether and replaces it with a classification-oriented concept of an object.

Where a program can be considered a critical mass of code, which performs many functions in the attempt to solve a problem with little consideration for object boundaries, an object is associated with the code to solve a particular set of functions having to do with just that type of object. By combining objects, like molecules, it is possible to create more organized systems than those created by traditional means. Software development becomes a speedier and less error-prone process as well. Since objects can be reused, once tested and implemented, they can be placed in a library for other developers to reuse. The more objects in the library, the easier and quicker it is to develop new systems. And since the objects being reused have, in theory, already been warranted (i.e., they've been tested and made error-free), there is less possibility that object-oriented systems will have major defects.

The process of writing programs and/or objects is known as software development, or software engineering. It is composed of a series of steps or phases, collectively referred to as a development life cycle. The phases include (at a bare minimum) the following: an analysis or requirements phase, where the business problem is dissected and understood; a specification phase, where decisions are made as to how

the requirements will be fulfilled (e.g., deciding what functions are allocated to software and what functions are allocated to hardware); a design phase, where everything from the GUI to the database to the output is designed or selected as part of a design; an implementation or programming phase, where one or more tools are used to write and/or generate code; a testing (debugging) phase, where the code is tested against a business test case and errors in the program are found and corrected; an installation phase, where the systems are placed in production; and a maintenance phase, where modifications are made to the system. But different people develop systems in different ways. These different paradigms make up the opposing viewpoints of software engineering.

34.2 The Nature of Software Engineering

Engineers often use the term "systems engineering" to refer to the tasks of specifying, designing, and simulating a non-software system such as a bridge or electronic component. Although software may be used for simulation purposes, it is but one part of the systems engineering process. Software engineering, on the other hand, is concerned with the production of nothing but software.

In the 1970s industry pundits began to notice that the cost of producing large-scale systems was growing at a high rate and that many projects were failing or, at the very least, resulting in unreliable products. Dubbed the software crisis, its manifestations were legion and the most important include the following:

- *Programmer Productivity.* In government in the 1980s, an average developer using C was expected to produce 10 lines of code per day (an average developer within a commercial organization was expected to produce 30 lines a month); today the benchmark in the government is more like 2 to 5 lines a day while at the same time the need is dramatically higher than that, perhaps by several orders of magnitude, ending up with a huge backlog. Programmer productivity is dependent upon a plethora of vagaries—from expertise to complexity of the problem to be coded to the size of the program that is generated. The science of measuring the productivity of the software engineering process is called metrics. Just as there are many diverse paradigms in software engineering itself, there are many paradigms of software measurement. Today's metric formulas are complex and often take into consideration the following: cost, time to market, productivity on prior projects, data communications, distributed functions, performance, heavily used configuration, transaction rate, online data entry, end-user efficiency, online update, complex processing, reusability, installation ease, operational ease, and multiplicity of operational sites.
- *Defect Removal Costs.* The same variables that affect programmer productivity affect the cost of "debugging" the programs and/or objects generated by those programmers. It has been observed that the testing and correcting of programs consumes a large share of the overall effort.
- *Development Environment.* Development tools and development practices greatly affect the quantity and quality of software. Most of today's design and programming environments contain only a fragment of what is really needed to develop a complete system. Life-cycle development environments provide a good example of this phenomena. Most of these tools can be described either as addressing the upper part of the life cycle (i.e., they handle the analysis and design) or the lower part of the life cycle (i.e., they handle code generation). There are few integrated tools on the market (i.e., that seamlessly handle both upper and lower functionalities). There are even fewer tools that add simulation, testing, and cross-platform generation to the mix. Rare are the tools that seamlessly integrate system design to software development.
- *GUI Development.* Developing GUIs is a difficult and expensive process unless the proper tools are used. The movement of systems from a host-based environment to the workstation and/or PC saw the entry of countless GUI development programs onto the marketplace. But the vast majority of these GUI-based tools do not have the capability of developing the entire system (i.e., the processing component as opposed to merely the front-end). This leads to fragmented and error-prone systems. To be efficient, the GUI builder must be well integrated into the software development environment.

The result of these problems is that most of today's systems require more resources allocated to maintenance than to the original development of that system. Lientz and Swanson [4] demonstrate that the problem is, in fact, larger than the one originally discerned during the 1970s. Software development is indeed complex, and the limitations on what can be produced by teams of software engineers given finite amounts of time, budgeted dollars, and talent have been amply documented by Jones [5].

Essentially the many paradigms of software engineering attempt to rectify the causes of declining productivity and quality. Unfortunately, this fails because current paradigms treat symptoms rather than the root problem. In fact, software engineering is itself extremely dependent upon both the software and hardware as well as the business environments upon which they sit [6].

SEI's process maturity grid very accurately pinpoints the root of most of our software development problems. The fact that a full 86% of organizations studied remain at the ad hoc or chaotic level indicate that only a few organizations (the remaining 14%) have adopted any formal process for software engineering. Simply put, 86% of all organizations react to a business problem by just writing codes. If they do employ a software engineering discipline, in all likelihood it is one that no longer fits the requirements of the ever-evolving business environment.

In the 1970s, the "structured methodology" was popularized. Although there were variations on the theme (i.e., different versions of the structured technique included the popular Gane–Sarson method and Yourdon method), for the most part, it provided a methodology to develop usable systems in an era of batch computing. In those days, online systems with even the dumbest of terminals were a radical concept and GUIs were as unthinkable as the fall of the Berlin Wall.

Although times have changed and today's hardware is one thousand times more powerful than when structured techniques were introduced, this technique still survives. And it survives in spite of the fact that the authors of these techniques have moved on to more adaptable paradigms, and more modern software development and systems engineering environments have entered the market.

In 1981, Finkelstein and Martin popularized "information engineering" [7] for the more commercially oriented users (i.e., those whose problems to be solved tended to be more database centered) which, to this day, is quite popular among mainframe developers with an investment in CASE strategies of the 1990s. Information engineering is essentially a refinement of the structured approach. However, instead of focusing on the data so preeminent in the structured approach, information engineering focuses on the information needs of the entire organization. Here business experts define high-level information models, as well as detailed data models. Ultimately, the system is designed from these models.

Both structured and information engineering methodologies have their roots in mainframe-oriented commercial applications. Today's migration to client/server technologies (where the organization's data can be spread across one or more geographically distributed servers while the end-user uses his or her GUI of choice to perform local processing), disables most of the utility of these methodologies. In fact, many issues now surfacing in more commercial applications are not unlike those that needed to be addressed earlier in the more engineering-oriented environments such as telecommunications and avionics.

Client/server environments are characterized by their diversity. One organization may store its data on multiple databases, program in several programming languages, and use more than one operating system, and hence, different GUIs. Since software development complexity is increased 100-fold in this new environment, a better methodology is required. Today's object-oriented techniques solve some of the problems. Given the complexity of the client/server environment, code trapped in programs is not flexible enough to meet the needs of this type of environment. We have already discussed how coding via objects rather than large programs engenders flexibility as well as productivity and quality through reusability. But object-oriented development is a double-edged sword.

While it is true that to master this technique is to provide dramatic increases in productivity, the sad fact of the matter is that object-oriented development, if done inappropriately, can cause problems far greater than problems generated from structured techniques. The reason for this is simple. The stakes are higher. Object-oriented environments are more complex than any other, the business problems chosen to be solved by object-oriented techniques are far more complex than other types of problems, and there are few if any conventional object-oriented methodologies and corollary tools to help the development

team develop good systems. There are many flavors of object orientation. But with this diversity comes some very real risks. As a result, the following developmental issues must be considered before the computer is even turned on.

- Integration is a challenge and needs to be considered at the onset. With traditional systems, developers rely on mismatched modeling methods to capture aspects of even a single definition. Whether it be integration of object to object, module to module, phase to phase, or type of application to type of application, the process can be an arduous one. The mismatch of products used in design and development compounds the issue. Integration is usually left to the devices of myriad developers well into development. The resulting system is sometimes hard to understand and objects are difficult to trace. The biggest danger is there is little correspondence to the real world. Interfaces are often incompatible and errors usually propagate throughout development. As a result, systems defined in this manner can be ambiguous and just plain incorrect.
- Errors need to be minimized. Traditional methods including those that are object oriented can actually encourage the propagation of errors, such as propagating errors through the reuse of objects with embedded and inherited errors throughout the development process. Errors must be eliminated from the very onset of the development process before they take on a life of their own.
- Languages need to be more formal.*[2] Although some languages are formal and others are friendly, it is hard to find languages both *formal* and *friendly*. Within environments where more informal approaches are used, lack of traceability and an overabundance of interface errors are a common occurrence. Recently, more modern software requirements languages have been introduced (for example, the Unified Modeling Language, UML [8]), most of which are informal (or semi-formal); some of these languages were created by "integrating" several languages into one. Unfortunately, the bad comes with the good—often, more of what is not needed and less of what is needed; and since the formal part is missing, common semantics need to exist to reconcile differences and eliminate redundancies.
- The syndrome of locked-in design needs to be eliminated. Often, developers are forced to develop in terms of an implementation technology that does not have an open architecture, such as a specific database schema or a GUI. Bad enough is to attempt an evolution of such a system; worse yet is to use parts of it as reusables for a system that does not rely on those technologies. Well thought-out and formal business practices and their implementation will help minimize this problem within an organization.
- Flexibility for change and handling the unpredictable must be dealt with up front. Too often it is forgotten that the building of an application must take into account its evolution. Users change their minds, software development environments change, and technologies change. Definitions of requirements in traditional development scenarios concentrate on the application needs of the user, but without consideration of the potential for the user's needs or environment to change. Porting to a new environment becomes a new development for each new architecture, operating system, database, graphics environment, or language. Because of this, critical functionality is often avoided for fear of the unknown, and maintenance, the most risky and expensive part of a system's life cycle, is left unaccounted for during development. To address these issues, tools and techniques must be used to allow cross technology and changing technology, as well as provide for changing and evolving architectures.
- Developers must prepare ahead of time for parallelism and distributed environments. Often, when it is known that a system is targeted for a distributed environment, it is first defined and developed for a single processor environment and then redeveloped for a distributed environment—an unproductive use of resources. Parallelism and distribution must be dealt with at the very start of the project.

*See Defining Terms for definition of formal.

- Resource allocation should be transparent to the user. Whether or not a system is allocated to distributed, asynchronous, or synchronous processors and whether or not two or ten processors are selected, with traditional methods, it is still up to the designer and developer to be concerned with incorporating such detail into the application. There is no separation between the specification of what the system is to do vs. how the system does it. This results in far too much implementation detail to be included at the level of design. Once such a resource architecture becomes obsolete, it is necessary to redesign and redevelop those applications which have old designs embedded within them.
- Automation that minimizes manual work needs to replace "make work" automated solutions. In fact, automation itself is an inherently reusable process. If a system does not exist for reuse, it certainly does not exist for automation. But most of today's development process is needlessly manual. Today's systems are defined with insufficient intelligence for automated tools to use them as input. In fact, automated tools concentrate on supporting the manual process instead of doing the real work. Typically, developers receive definitions, which they manually turn into code. A process that could have been mechanized once for reuse is performed manually again and again. Under this scenario, even when automation attempts to do the real work, it is often incomplete across application domains or even within a domain, resulting in incomplete code such as shell code. The generated code is often inefficient or hardwired to a particular kind of algorithm, an architecture, a language, or even a version of a language. Often partial automations need to be integrated with incompatible partial automations or manual processes. Manual processes are needed to complete unfinished automations.
- Run-time performance analysis (decisions between algorithms or architectures) should be based on formal definitions. Conventional system definitions contain insufficient information about a system's run-time performance, including that concerning the decisions between algorithms or architectures. System definitions must consider how to separate the system from its target environment. Design decisions, where this separation is not taken into account, thus depend on analysis of results from ad hoc "trial and error" implementations and associated testing scenarios.
- The creation of reliable reusable definitions must be promoted, especially those that are inherently provided. Conventional requirements definitions lack the facilities to help find, create, use, and ensure commonality in systems. Modelers are forced to use informal and manual methods to find ways to divide a system into components natural for reuse. These components do not lend themselves to integration and, as a result, they tend to be error-prone. Because these systems are not portable or adaptable, there is little incentive for reuse. In conventional methodologies, redundancy becomes a way of doing business. Even when methods are object oriented, developers are often left to their own devices to explicitly make their applications object oriented. This is because these methods do not support all that is inherent to the process of object orientation.
- Design integrity is the first step to usable systems. Using traditional methods, it is not known if a design is a good one until its implementation has failed or succeeded. Usually, a system design is based on short-term considerations because knowledge is not reused from previous lessons learned. Development, ultimately, is driven towards failure. The solution is to have an inherent means to build reliable, reusable definitions.

Once these issues are addressed, software will cost less and take less time to develop. But time is of the essence. These issues are becoming compounded and even more critical as developers prepare for the distributed environments that go hand in hand with the increasing predominance of Internet applications.

With respect to the challenges described above, an organization has several options, ranging from one extreme to the other. The options include: (1) keep things the same; (2) add tools and techniques that support business as usual, but provide relief in selected areas; (3) bring in more modern but traditional tools and techniques to replace existing ones; (4) use a new paradigm with the most advanced tools and techniques that formalizes the process of software development, while at the same time capitalizing on

software already developed; or (5) completely start over with a new paradigm that formalizes the process of software development and uses the most-advanced tools and techniques.

34.3 Development before the Fact

Thus far, this chapter has explained the derivation of software and attempted to show how it has evolved over time to become the true "brains" of any automated system. But, like a human brain, this software brain must be carefully architected to promote productivity, foster quality, and enforce control and reusability.

Traditional software engineering paradigms fail to see the software development process from the larger perspective of the superorganism described at the beginning of this chapter. It is only when we see the software development process as made of discrete, but well-integrated, components can we begin to develop a methodology that can produce the very benefits that have been promised by the advent of software decades ago.

Software engineering, from this perspective, consists of a methodology as well as a series of tools with which to implement the solution to the business problem at hand. But even before the first tool can be applied, the software engineering methodology must be deployed to assist in specifying the requirements of the problem. How can this be accomplished successfully in the face of the issues needed to be addressed outlined in the last section? How can this be accomplished in situations where organizations must develop systems that run across diverse and distributed hardware platforms, databases, programming languages, and GUIs when traditional methodologies make no provision for such diversity? And how can software be developed without having to fix or "cure" those myriad of problems, which result "after the fact" of that software's development?

What is required is a radical revision of the way we build software, an approach that understands how to build systems using the right techniques at the right time. First and foremost, it is a preventative approach. This means it provides a framework for doing things right the first time. Problems associated with traditional methods of design and development are prevented "before the fact" just by the way a system is defined. Such an approach would concentrate on preventing problems of development from even happening rather than letting them happen "after the fact," and fixing them after they have surfaced at the most inopportune and expensive point in time.

Consider such an approach in its application to a human system. To fill a tooth before it reaches the stage of a root canal is curative with respect to the cavity, but preventive with respect to the root canal. Preventing the cavity by proper diet prevents not only the root canal, but the cavity as well. To follow a cavity with a root canal is the most expensive alternative, to fill a cavity on time is the next most expensive, and to prevent these cavities in the first place is the least expensive option.

Preventiveness is a relative concept. For any given system, be it human or software, one goal is to prevent, to the greatest extent and as early as possible, anything that could go wrong in the life cycle process.

With a preventative philosophy, systems would be carefully constructed to minimize development problems from the very outset. A system could be developed with properties that controlled its very own design and development. One result would be reusable systems that promote automation. Each system definition would model both its application and its life cycle with built-in constraints—constraints that protect the developer, but yet do not take away his flexibility.

The philosophy behind preventative systems is that reliable systems are defined in terms of reliable systems. Only reliable systems are used as building blocks, and only reliable systems are used as mechanisms to integrate these building blocks to form a new system. The new system becomes reusable for building other systems.

Effective reuse is a preventative concept. That is, reusing something (e.g., requirements or code) that contains no errors to obtain a desired functionality avoids both the errors and the cost of developing a new system. It allows one to solve a given problem as early as possible, not at the last moment. But to make a system truly reusable, one must start not from the customary end of a life cycle, during the implementation or maintenance phase, but from the very beginning.

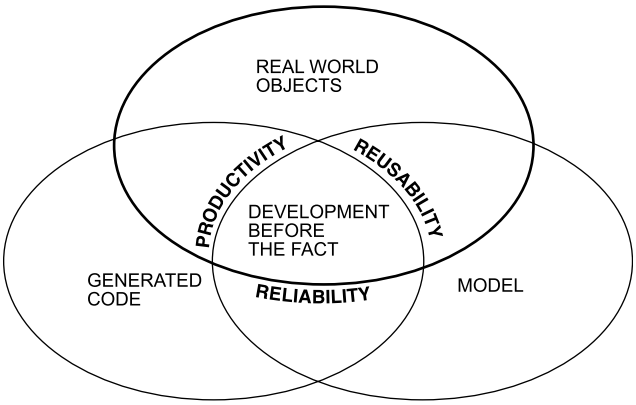


FIGURE 34.1 The development before the fact paradigm.

Preventative systems are the true realization of the entelechy construct where molecules of software naturally combine to form a whole much greater than the sum of its parts. Or one can think of constructing systems from the tinker toys of our youth. One recalls that the child never errs in building magnificent structures from these tinker toys. Indeed, tinker toys are built from blocks that are architected to be perpetually reusable, perfectly integratable, and infinitely user-friendly.

One approach that follows this preventative philosophy is development before the fact (DBTF), as shown in Figure 34.1. Not yet in the mainstream, it has been used successfully by research and "trail blazer" organizations and is now being adopted for more commercial use. This technology is described in order to illustrate, by example, the potential that preventative approaches have.

Where traditional approaches begin the process of developing software after the fact, the DBTF paradigm is very much about beginnings. It was derived from the combination of steps taken to solve the problems of traditional systems engineering and software development. DBTF includes a technology, a language, and a process (or methodology) based on a formal theory.

34.3.1 Language

Once understood, the characteristics of good design can be reused by incorporating them into a language for defining any system (i.e., not just a software system). One language based on DBTF is a formalism for representing the mathematics of systems. A system defined with this language has properties that come along "for the ride" that in essence control its own destiny. Based on a theory (DBTF) that extends traditional mathematics of systems with a unique concept of control, this formal, but friendly language has embodied within it a natural representation of the physics of time and space. With this language, every object is a system-oriented object (SOO), an integration that includes aspects of being function oriented (including dynamics) and object oriented. Instead of systems being object oriented, objects are systems oriented. All systems are objects and all objects are systems.

Because of this, many things heretofore not believed possible with traditional methods are possible. A DBTF system inherently integrates all of its own objects (and all aspects, relationships, and viewpoints of these objects) and the combinations of functionality, including timing, using these objects; maximizes its own reliability and flexibility to change (including the change of target requirements, static and dynamic architectures, and processes and as well reconfiguration in real time); capitalizes on its own parallelism and traceability; supports its own run-time performance analysis; and maximizes the potential for its own reuse (providing inherent resource allocation and reuse without need for the designer's intervention); and it provides the ability to automate design and development wherever and whenever possible. Each DBTF system is defined with built-in quality, built-in productivity, and built-in control.

The language—meta-language, really—is the key to DBTF. Its main attribute is to help the designer reduce the complexity and bring clarity into his thinking process, turning it into the ultimate reusable, which is wisdom itself. It can be used to define any aspect of any system and integrate it with any other aspect. The crucial point is that these aspects are directly related to the real world and, therefore, the same language can be used to define system requirements, specifications, design, and detailed design for functional, resource, and resource allocation architectures throughout all levels and layers of seamless definition, including hardware, software, and peopleware.

This language based on DBTF can be used to define organizations of people, missile or banking systems, cognitive systems, as well as real-time or database environments and is, therefore, appropriate across industries, academia, or government.

34.3.2 Technology

Real-world experience sets the stage for the DBTF technology. Having evolved over three decades, the theory has roots in the worlds of systems theory, formal methods, and object technology. The DBTF technology embodies the theory, the language supports its representation, and its automation supports its application and use. Each is evolutionary (in fact, recursively so), with experience feeding the theory and the theory feeding the language, which in turn feeds the automation. All are used, in concert, to design systems and build software.

The DBTF approach had its beginnings in 1968 with an empirical analysis of the Apollo space missions. A better way was needed to define and develop systems than the ones being used and available because the existing ones (just like the traditional ones today) did not solve the pressing problems. Research for developing software for man-rated missions led to the finding that interface errors accounted for approximately 75% of all errors found in the flight software during final testing (in traditional development, the figure is as high as 90%). Such errors include data flow, priority, and timing errors from the highest levels of a system to the lowest level of detail. Each error was categorized according to how it could be prevented just by the way a system is defined. This work led to a theory and methodology for defining a system that would eliminate all interface errors.

The first technology derived from this theory concentrated on defining and building reliable systems. Having realized the benefits of addressing one major issue, such as reliability, research continued to evolve by addressing other major issues the same way, that is, just by the way a system is defined [9–11].

DBTF is a function- and object-oriented approach based on a unique concept of control, which is lacking in any other software engineering paradigm. The foundations are based on a set of axioms and on the assumption of a universal set of objects. Each axiom defines a relation of immediate domination. The union of the relations defined by the axioms is control. Among other things, the axioms establish the relationships of an object for invocation, input and output, input and output access rights, error detection and recovery, and ordering during its developmental and operational states. Table 34.1 summarizes some of the properties of objects within DBTF systems.

34.3.3 Process

Where software engineering fails is in its inability to grasp that not only the right paradigm (out of many paradigms) must be selected, but that the paradigm must be part of an environment that provides an integrated automated means to solve the problem at hand. What this means is that the paradigm must be coupled with an integrated system of tools with which to implement the results of utilizing that paradigm to develop the model of the system.

Essentially, the paradigm generates the model and a toolset must be provided to generate the system. DBTF provides this next-generation capability.

This DBTF approach is used throughout a life cycle, starting with requirements and continuing with functional analysis, simulation, specification, analysis, design, system architecture design, algorithm development, implementation, configuration management, testing, maintenance, and reverse engineering. Its users include end users, managers, system engineers, software engineers, and test engineers.

TABLE 34.1 System Oriented Object Properties of Development before the Fact

Quality (better, faster, cheaper)
- Reliable
- Affordable

Reliable (better)
- In control and under control
- Based on a set of axioms
  - –domain identification (intended, unintended)
  - –ordering (priority and timing)
  - –access rights: Incoming object (or relation), outgoing object (or relation)
  - –replacement
- Formal
  - –consistent, logically complete
  - –necessary and sufficient
  - –common semantic base
  - –unique state identification
- Error free (based on formal definition of "error")
  - –always gets the right answer at the right time and in the right place
  - –satisfies users and developers intent
- Handles the unpredictable
- Predictable

Affordable (faster, cheaper)
- Reusable
- Optimizes resources in operation and development
  - –in minimum time and space
  - –with best fit of objects to resources

Reusable
- *Understandable*, integratable and maintainable
- Flexible
- Follows standards
- Automation
- Common definitions
  - –natural modularity
    - -natural separation (e.g., functional architecture from its resource architectures);
    - -dumb modules
    - -an object is integrated with respect to structure, behavior and properties of control
  - –integration in terms of structure and behavior
  - –type of mechanisms
    - -function maps (relate an object's function to other functions)
    - -object type maps (relate objects to objects)
    - -structures of functions and types
  - –category
    - -relativity
      - instantiation
      - polymorphism
      - parent/child
      - being/doing
      - having/not having
    - -abstraction
      - encapsulation
      - replacement

Handles the unpredictable
- throughout development and operation
- Without affecting unintended areas
- Error detect and recover from the unexpected
- Interface with, change and reconfigure in asynchronous, distributed, real-time environment

Flexible
- Changeable without side effects
- Evolvable
- Durable
- Reliable
- Extensible
- Ability to break up and put together
  - –one object to many: modularity, decomposition, instantiation
  - –many objects to one: composition, applicative operators, integration, abstraction
- Portable
  - –secure
  - –diverse and changing layered developments
  - –open architecture (implementation, resource allocation, and execution independence)
  - –plug-in (or be plugged into) or reconfiguration of different modules
  - –adaptable for different organizations, applications, functionality, people, products

Automation
- the ultimate form of reusable
- formalize, mechanize, then automate
  - –it
  - –its development
  - –that which automates its development

Understandable, integratable and maintainable
- Reliable
- A measurable history
- Natural correspondence to real world
  - –persistence, create and delete
  - –appear and disappear
  - –accessibility
  - –reference
  - –assumes existence of objects
  - –real time and space constraints
  - –representation
  - –relativity, abstraction, derivation
- Provides user friendly definitions
  - –recognizes that one user's friendliness is another user's nightmare
  - –hides unnecessary detail (abstraction)
  - –variable, user selected syntax
  - –self teaching
  - –derived from a common semantic base
  - –common definition mechanisms
- Communicates with common semantics to all entities
- Defined to be simple as possible but not simpler

(*continued*)

TABLE 34.1 (Continued)

relation including function typing including classification form including both structure and behavior (for object types and functions) -derivation deduction inference inheritance	• Defined with integration of all of its objects (and all aspects of these objects) • Traceability of behavior and structure and their changes (maintenance) throughout its birth, life and death • Knows and able to reach the state of completion –definition –development of itself and that which develops it -analysis -design -implementation -instantiation -testing -maintenance

Note: all underlined words point to a reusable.

Source: Hamilton, M., "Software Design and Development," *The Electronics Handbook,* CRC Press, Boca Raton, FL, 1996. With permission.

The DBTF process combines mathematical perfection with engineering precision. Its purpose is to facilitate the "doing things right in the first place" development style, avoiding the "fixing wrong things up" traditional approach. Its automation is developed with the following considerations: error prevention from the early stage of system definition, life cycle control of the system under development, and inherent reuse of highly reliable systems. The development life cycle is divided into a sequence of stages, including requirements and design modeling by formal specification and analysis, automatic code generation based on consistent and logically complete models, test and execution, and simulation.

The first step in building a DBTF system is to define a model with the language. This process could be in any phase of the developmental life cycle, including problem analysis, operational scenarios, and design. The model is automatically analyzed to ensure it was defined properly. This includes static analysis for preventive properties and dynamic analysis for user-intent properties.

In the next stage, the generic source code generator automatically generates a fully production-ready and fully integrated software implementation for any kind of application, consistent with the model, for a selected target environment in the language and architecture of choice. If the selected environment has already been configured, the generator selects that environment directly; otherwise, the generator is first configured for a new language and architecture.

Because of its open architecture, the generator can be configured to reside on any new architecture (or interface to any outside environment), e.g., to a language, communications package, an Internet interface, a database package, or an operating system of choice; or it can be configured to interface to the users own legacy code. Once configured for a new environment, an existing system can be automatically regenerated to reside on that new environment. This open architecture approach, which lends itself to true component-based development, provides more flexibility to the user when changing requirements or architectures, or when moving from an older technology to a newer one.

It then becomes possible to execute the resulting system. If it is software, the system can undergo testing for further user-intent errors. It becomes operational after testing. Application changes are always made to the requirements/specification definition—not to the code (the developer does not even need to change the code). Target architecture changes are made to the configuration of the generator environment (which generates one of a possible set of implementations from the model)—not to the code. If the real system is hardware or peopleware, the software system serves as a simulation upon which the real system can be based. Once a system has been developed, the system and the process used to develop it are analyzed to understand how to improve the next round of system development.

Seamless integration is provided throughout from systems to software, requirements to design to code to tests to other requirements and back again; level to level and layer to layer. The developer is able to trace from requirements to code and back again.

TABLE 34.2 A Comparison

Traditional (After the Fact)	DBTF (Before the Fact)
Interface errors (*over 75% of all errors*)	*No interface errors*
Most found after implementation	All found before implementation
Some found manually	All found by automatic and static analysis
Some found by dynamic runs analysis	
Some never found	Always found
Ambiguous requirements	*Unambiguous requirements*
Informal or semiformal language	formal, but friendly language
Different phases, languages, and tools	All phases, same language and tools
Different language for other systems than for software	Same language for software, hardware and any other system
Automation supports manual process	*Automation does real work*
Mostly manual documentation, programming, test generation, traceability, etc.	Automatic documentation, programming, test generation, traceability, etc.
	100% code automatically generated for any kind of software
No guarantee of function integrity after implementation	*Guarantee of function integrity after implementation*
Systems not traceable or evolvable	*Systems traceable and evolvable*
Locked in products, architectures, etc.	Open architecture
Painful transition from legacy	Smooth transition from legacy
Maintenance performed at code level	Maintenance performed at spec level
Reuse not inherent	*Inherent reuse*
Reuse is adhoc	Every object a candidate for reuse
Customization and reuse are mutually exclusive	Customization increases reuse pool
Mismatched objects, phases, products, architectures and environment	*Integrated & seamless objects, phases, products, architectures, and environment*
System not integrated with software	System integrated with software
Function oriented or object oriented	System oriented objects: integration of function, timing, and object oriented
GUI not integrated with application	GUI integrated with application
Simulation not integrated with software code	Simulation integrated with software code
Automation not defined and developed with itself	*Automation defined with and generated by itself*
	#1 in all evaluations
Dollars wasted, error prone systems	*Better, faster, cheaper systems*
Not cost-effective	10 to 1, 20 to 1, 50 to 1…dollars saved
Difficult to meet schedules	Minimum time to complete
Less of what you need and more of what you don't need	No more, no less of what you need

Given an automation that has these capabilities, it should be of no surprise that an automation of DBTF has been defined with itself and that it continues to automatically generate itself as it evolves with changing architectures and changing technologies. Table 34.2 contains a summary of some of the differences between the more modern preventative paradigm and the traditional approach.

A relatively small set of things is needed to master the concepts behind DBTF. Everything else can be derived, leading to powerful reuse capabilities for building systems. It quickly becomes clear why it is no longer necessary to add features to the language or changes to a developed application in an ad hoc fashion, since each new aspect is ultimately and inherently derived from its mathematical foundations.

34.4 Experience with DBTF

That preventative development is a superior alternative has been proven rather dramatically in several experiments. DBTF has been through many evaluations and competitions conducted and sponsored by leading academic institutions, government agencies, and commercial organizations. In every evaluation and competition this alternative came out on top. What set this alternative apart from the others was

that it provided a totally integrated system design and development environment, whereas the traditional methods resulted in an informal, difficult to integrate (including application modules as well as the products used to implement them), fragmented, more manual, and "after the fact" life-cycle process.

The National Test Bed of the U.S. Department of Defense sponsored an experiment in which it provided a development problem to each of three contractor/vendor teams chosen from a large pool of vendors and development environments, based upon a well-defined set of requirements. The application was a real-time, distributed, multiuser, client server system, which needed to be defined and developed under the government 2167A guidelines.

All teams were able to complete the first part, the definition of preliminary requirements. Two teams completed the detailed design. But only one team was able to generate complete, integrated, and fully production-ready code automatically; a major portion of this code was running in both C and Ada at the end of the experiment [12]. The team that was able to generate the production-ready code was using the 001 Tool Suite, a development environment based on the DBTF methodology.

34.5 Conclusion

Businesses that expected a big productivity payoff from investing in technology are, in many cases, still waiting to collect. A substantial part of the problem stems from the manner in which organizations are building their automated systems. While hardware capabilities have increased dramatically, organizations are still mired in the same old methodologies that saw the rise of the behemoth mainframes. Old methodologies simply cannot build the new systems.

There are other changes as well. Users demand much more functionality and flexibility in their systems. And given the nature of many of the problems to be solved by this new technology, these systems must also be error-free as well.

Where the biological superorganism has built-in control mechanisms fostering quality and productivity, until now the silicon superorganism has had none. Hence, the productivity paradox.

Often, the only way to solve major issues or to survive tough times is through nontraditional paths or innovation. One must create new methods or new environments for using new methods.

Innovation for success often starts with a look at mistakes from traditional systems. The first step is to recognize the true root problems, then categorize them according to how they might be prevented. Derivation of practical solutions is a logical next step. Iterations of the process entail looking for new problem areas in terms of the new solution environment and repeating the scenario. That is how DBTF came into being.

With DBTF all aspects of system design and development are integrated with one systems language and its associated automation. Reuse naturally takes place throughout the life cycle. Objects, no matter how complex, can be reused and integrated. Environment configurations for different kinds of architectures can be reused. A newly developed system can be safely reused to increase even further the productivity of the systems developed with it.

The paradigm shift occurs once a designer realizes that many of the old tools are no longer needed to design and develop a system. For example, with one formal semantic language to define and integrate all aspects of a system, diverse modeling languages (and methodologies for using them), each of which defines only part of a system, are no longer necessary. There is no longer a need to reconcile multiple techniques with semantics that interfere with each other.

DBTF can support a user in addressing many of the challenges presented in today's software development environments. There will, however, always be more to do to capitalize on this technology. That is part of what makes a technology like this so interesting to work with. Because it is based on a different premise or set of assumptions (set of axioms), a significant number of things can and will change because of it. There is the continuing opportunity for new research projects and new products. Some problems can be solved, because of the language, that could not be solved before. Software development as we know it will never be the same. Many things will no longer need to exist—they, in fact, will be rendered extinct, just as that phenomenon occurs with the process of natural selection in the biological system. Techniques for

bridging the gap from one phase of the life cycle to another become obsolete. Testing procedures and tools for finding most errors are no longer needed because those errors no longer exist. Tools to support programming as a manual process are no longer needed.

Compared to the development using traditional techniques, the productivity of DBTF developed systems has been shown to be significantly greater. Upon further analysis, it was discovered that the larger and more complex the system, the greater the productivity—the opposite of what one finds with traditional systems development. This is, in part, because of the high degree of DBTF's support of reuse. The larger a system, the more it has the opportunity to capitalize on reuse. As more reuse is employed, productivity continues to increase. Measuring productivity becomes a process of relativity—that is, relative to the last system developed.

Capitalizing on reusables within a DBTF environment is an ongoing area of research interest. An example is understanding the relationship between types of reusables and metrics. This takes into consideration that a reusable can be categorized in many ways. One is according to the manner in which its use saves time (which translates to how it impacts cost and schedules). More intelligent tradeoffs can then be made. The more we know about how some kinds of reusables are used, the more information we have to estimate costs for an overall system. Keep in mind also that the traditional methods for estimating time and costs for developing software are no longer valid for estimating systems developed with preventative techniques.

There are other reasons for this higher productivity as well, such as the savings realized and time saved due to tasks and processes that are no longer necessary with the use of this preventative approach. There is less to learn and less to do—less analysis, little or no implementation, less testing, less to manage, less to document, less to maintain, and less to integrate. This is because a major part of these areas has been automated or because of what inherently take place because of the nature of DBTF's formal systems language.

In the end, it is the combination of the technology and that which executes it that forms the foundation of successful software. Software is so ingrained in our society that its success or failure will dramatically influence both the operation and the success of an organization. For that reason, today's decisions about systems engineering and software development have far-reaching effects.

Software is a relatively young technological field that is still in a constant state of change. Changing from a traditional software environment to a preventative one is like going from the typewriter to the word processor. Whenever there is any major change, there is always the initial overhead needed for learning the new way of doing things. But, as with the word processor, progress begets progress.

Collective experience strongly confirms that quality and productivity increase with the increased use of properties of preventative systems. In contrast to the "better late than never" after the fact philosophy, the preventive philosophy behind DBTF is to solve—or if possible, prevent—a given problem as early as possible. Finding a problem statically is better than finding it dynamically. Preventing it by the way a system is defined is even better. Better yet is not having to define (and build) it at all.

Reusing a reliable system is better than reusing one that is not reliable. Automated reuse is better than manual reuse. Inherent reuse is better than automated reuse. Reuse that can evolve is better than one that cannot evolve. Best of all is reuse that ultimately approaches wisdom itself. Then, have the wisdom to use it.

The answer continues to be in the results just as in the biological system; and the goal is that the systems of tomorrow will inherit the best of the systems of today.

Defining Terms

Data Base Management System (DBMS): The computer program that is used to control and provide rapid access to a database. A language is used with the DBMS to control the functions that a DBMS provides. For example, SQL is the language that is used to control all of the functions that a relational architecture-based DBMS provides for its users, including data definition, data retrieval, data manipulation, access control, data sharing, and data integrity.

Graphical User Interface (GUI): The ultimate user interface, by which the deployed system interfaces with the computer most productively, using visual means. Graphical user interfaces provide a series of intuitive, colorful, and graphical mechanisms that enable the end-user to view, update, and manipulate information.

Interface: A point of access in a boundary between objects or programs or systems. It is at this juncture that many errors surface. Software can interface with hardware, humans, and other software.

Methodology: A set of procedures, precepts, and constructs for the construction of software.

Metrics: A series of formulas that measure such things as quality and productivity.

Software Architecture: The structure and relationships among the components of software.

Formal: A system defined in terms of a known set of axioms (or assumptions); it is, therefore, mathematically based (e.g., a DBTF system is based on a set of axioms of control). Some of its properties are that it is consistent and logically complete. A system is consistent if it can be shown that no assumption of the system contradicts any other assumption of that system. A system is logically complete if the assumptions of the method completely define a given set of properties. This assures that a model of the method has that set of properties. Other properties of the models defined with the method may not be provable from the method's assumptions. A logically complete system has a semantic basis (i.e., a way of expressing the meaning of that system's objects). In terms of the semantics of a DBTF system, this means it has no interface errors and is unambiguous, contains what is necessary and sufficient, and has a unique state identification.

Further Information

Hamilton, M. and Hackler, W. R., Object Thinking: Development Before the Fact, In Press.

Krut, Jr., B. "Integrating 001 Tool Support in the Feature-Oriented Domain Analysis Methodology" (CMU/SEI-93-TR-11, ESC-TR-93-188). Pittsburgh, PA: Software Engineering Institute, Carnegie-Mellon University, 1993.

Ouyang, M. and Golay, M. W., "An Integrated Formal Approach for Developing High Quality Software of Safety-Critical Systems," Massachusetts Institute of Technology, Cambridge, MA, Report No. MIT-ANP-TR-035., September, 1995.

McCauley, B. "Software Development Tools in the 1990s," AIS Security Technology for Space Operations Conference, July 1993, Houston, TX.

Hamilton, M. and Hackler, W. R., Towards Cost Effective and Timely End-to-End Testing, HTI, prepared for Army Research Laboratory, Contract No. DAKF11-99-P-1236, July 17, 2000.

Keyes, J., *Internet Management,* Chapters 30–33, on 001-developed systems for the Internet, Auerbach, Boca Raton, FL, 2000.

References

1. Software Engineering Institute. *Capability Maturity Model,* Pittsburgh, PA: Carnegie, Mellon University, 1991.
2. Stroustrup, B., *The C++ Programming Language,* Reading, MA: Addison-Wesley, 1997.
3. Gosling, J., Joy, B., and Steele, G., *The Java Language Specification,* Reading, MA: Addison-Wesley, 1996.
4. Lientz, B.P., and Swanson, E.B., *Software Maintenance Management,* Reading, MA: Addison-Wesley, 1980.
5. Jones, T.C., *Program Quality and Programmer Productivity,* IBM Tech. Report TR02.764 January: 80, San Jose, CA: Santa Teresa Labs, 1977.
6. Keyes, J., *Handbook of E-Business,* Chapter F5, Hamilton, M., Defining e…com for e-Profits, RIA, 2000.
7. Martin, J., and Finkelstein, C.B., *Information Engineering,* Carnforth, Lancs, U.K.: Savant Institute, 1981.
8. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide,* Addison-Wesley, 1999.
9. Hamilton, M., "*Inside Development Before the Fact,*" Electronic Design, April 4, 1994, ES.
10. Hamilton, M., "*Development Before the Fact in Action,*" Electronic Design, June 13, 1994, ES.
11. Keyes, J., *The Ultimate Internet Developers Sourcebook,* AMACOM, to be published Fall 2001.
12. Software Engineering Tools Experiment-Final Report, Vol. 1, Experiment Summary, Table 1, Page 9, Department of Defense, Strategic Defense Initiative, Washington, D.C., 20301-7100, October 1992.

35

Data Recording and Logging

Craig Anderson
National Instruments, Inc.

35.1 Overview

Data logging and recording is a very common measurement application. In its most basic form, data logging is the measurement and recording of physical or electrical parameters over a period of time. These parameters can be temperature, strain, displacement, .ow, pressure, voltage, current, resistance, power, or any of a wide range of other measurement types. Real-world data-logging applications are typically more involved than just acquiring and recording signals, as illustrated in Figure 35.1. They usually involve some combination of online analysis, offline analysis, display, report generation, and data sharing. In addition, many data-logging applications are beginning to require the acquisition and storage of other types of data. One example would be recording sound and video in conjunction with the other parameters measured during an automobile crash test.

Data logging is used in a broad spectrum of applications. Chemists record data such as temperature, pH, and pressure when performing experiments in a lab. Design engineers log performance parameters such as vibration, temperature, and battery level to evaluate product designs. Civil engineers record strain and load on bridges over time to evaluate safety. Geologists use data logging to determine mineral formations when drilling for oil. Breweries log the conditions of their storage and brewing facilities to maintain quality.

The list of applications for data logging goes on and on, but all these applications have similar requirements. The purpose of this chapter is to provide some general background on data logging, discuss the various functional requirements that are common to most data-logging applications, and examine some

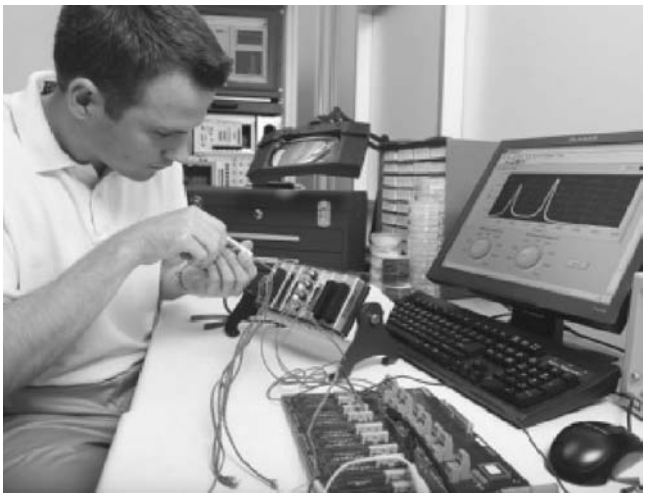


FIGURE 35.1 PC-based data-logging application.

of the modern hardware and software options available that allow scientists and engineers to implement powerful PC-based data-logging systems.

35.2 Historical Background

The earliest form of data logging involved taking manual measurements from analog instruments, such as thermometers and manometers. These measurements were recorded in a written log, along with the time of observation. To view trends over time, people manually plotted their measurements on paper. In the late-nineteenth century, it became possible to begin automating this process with machines, and strip chart recorders evolved. Strip chart recorders are analog instruments that translate electrical impulses from sensors into mechanical movement of an arm. A pen is attached to the arm, and long rolls of paper aremoved at a constant rate under the pen. The result is a paper chart displaying the parameters measured over the course of time. Strip chart recorders were a great improvement over manual data logging, but they still had drawbacks. For example, translating the traces on the paper into meaningful engineering measurements was tedious at best, and the data that was recorded took up reams and reams of paper.

With the development of the personal computer in the 1970s and 1980s, people began to leverage computers for analysis of data, data storage, and report generation. The need to bring data into the PC brought about a new type of equipment—the data logger. Data loggers are stand-alone box instruments that measure signals, convert them to digital data, and store the data internally. This data must be transferred to the PC for analysis, permanent storage, and report generation. Data is typically transferred either by manually moving a storage device, such as a .oppy disk, from the data logger to the computer, or by connecting the data logger to the PC through some communications link, such as serial or Ethernet.

In the 1990s, a further evolution in data logging took place as people began to create PC-based data-logging systems. These systems combined the acquisition and storage capabilities of stand-alone data loggers with the archiving, analyzing, reporting, and displaying capabilities of modern PCs. PC-based logging systems finally enabled full automation of the data-logging process. The move to PC-based data-logging systems was made possible by the following three technological enhancements:

1. Increasing PC reliability
2. Steadily decreasing cost of PC hard drive space
3. PC-based measurement hardware that could meet or exceed stand-alone data logger measurement capabilities

Today, PC-based logging systems provide the widest range of measurement types, analysis capabilities, and reporting tools. The remainder of this chapter focuses on the functionality necessary to implement a PC-based data-logging system.

35.3 Data Logging Functional Requirements

Every data-logging application—from .fteenth-century monks manually recording weather patterns to twenty-first-century physicists logging the experimental parameters of a fusion reactor test—can be broken down into a set of .ve commonfunctional requirements, illustrated in Figure 35.2. Acquiring is the process of actually measuring the physical parameters and bringing them into your logging system. Online analysis consists of any processing done to the data while it is being acquired, including alarms, data scaling, and sometimes control. Logging or storing the data is an obvious requirement of every data-logging system.

Offline analysis is everything that is done with the data after it has been acquired in order to extract useful information from it. The final functional block is made up of display, reporting, and data sharing. These are all the "miscellaneous" requirements that fill out the functionality of a data-logging system. Let us examine how each of these functional blocks is addressed with modern PC-based data-logging systems.

35.3.1 Acquisition

The acquisition function is one of the most critical components of every data-logging system. In a PCbased system, the acquisition is accomplished by the measurement hardware, which can be broken down further into sensors, signal connectivity, signal conditioning, and analog-to-digital (A/D) conversion, as shown in Figure 35.3. Each of these topics is covered in more detail in other sections of this chapter, so only a high-level overview is given here.

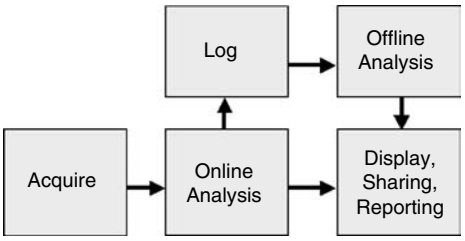


FIGURE 35.2 Basic elements of a data-logging system.

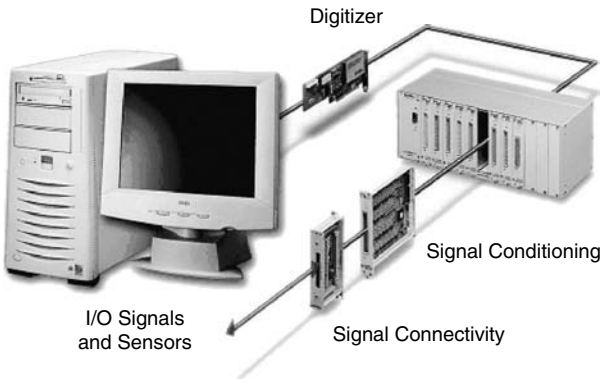


FIGURE 35.3 Measurement hardware components of a PC-based data-logging system.

35.3.2 Sensors

A wide variety of sensors are used to convert physical parameters into electrical signals. Temperature sensors such as thermocouples, RTDs, or thermistors are some of the most common sensors used in data-logging applications. Other widespread sensors are .owmeters, pressure transducers, strain gauges, accelerometers, and microphones, to name a few. Proper sensor selection and installation is beyond the scope of this chapter.

35.3.3 Signal Connectivity

After sensors are installed, they must be connected to the data-logging system. Signal connectivity is the component of the measurement hardware that allows you to connect your sensors to your logging system. Screw terminals, which allow you to connect bare wires from sensors directly to your logging system, are the most basic form of connectivity. Screw terminals are a good choice for general-purpose use, particularly when you need to connect a large number of signals into a small amount of space. The disadvantage of screw terminals is that they are time-consuming to connect and dif.cult to recon.gure. Figure 35.4 shows some other standard connectivity options that are designed to make connecting and disconnecting sensors less labor intensive. Minithermocouple connectors are a widely used connectivity option for thermocouples. BNC and SMB connectors are commonly used when electrical shielding is required for noise immunity. Banana jacks are often used when measuring current, resistance, or higher voltages. The sensor provider typically defines the connectivity options available for a sensor, and it is up toy ou toc hoose measurement hardware that can accept that connectivity.

35.3.4 Signal Conditioning

Signal conditioning is one of the most important and most overlooked components of a PC-based datalogging system. Many sensors require special signal conditioning technology. For example, signals from a thermocouple have very low voltage and require amplification, filtering, and linearization. Other sensors, such as strain gauges and accelerometers, require power in addition to amplification and filtering, while other signals may require isolation to protect the system from high voltages. No single stand-alone data logger can provide the flexibility required to make all these measurements. However, with front-end signal conditioning, you can combine the necessary technologies to bring these various types of signals into a single PC-based data-logging system.

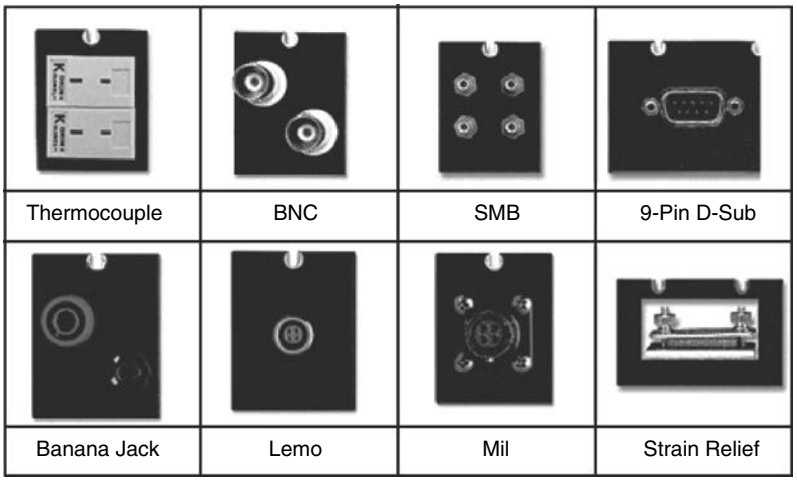


FIGURE 35.4 Examples of signal connectivity options.

Because of the vast array of signal-conditioning technologies, the role and need for each technology can quickly become confusing. A list of common types of signal conditioning, their functionality, and examples of when they are needed is given below.

35.3.4.1 Amplification

When the voltage levels being measured are very small, amplification is used to maximize the effectiveness of the digitizer. By amplifying the input signal, the conditioned signal uses more of the effective range of the A/D converter. This allows better accuracy and resolution of the measurement. Typical sensors that require amplification are thermocouples and strain gauges.

35.3.4.2 Attenuation

Attenuation is the opposite of amplification. It is necessary when the voltages to be digitized are outside the input range of the digitizer. This form of signal conditioning divides the input signal so that the conditioned signal is within the range of the A/D converter. Attenuation is necessary for measuring high voltages.

35.3.4.3 Isolation

Voltage signals outside the range of the digitizer can damage the measurement systemand harm the operator. For that reason, isolation is usually required in conjunction with attenuation to protect the system and the user from dangerous voltages or voltage spikes. Isolation may also be required when the sensor is on a different electrical ground plane from the measurement sensor (such as a thermocouple mounted on an engine).

35.3.4.4 Multiplexing

Typically, the digitizer is the most expensive part of a data acquisition system. Multiplexing allows you to automatically routemultiple signals into a single digitizer, providing a cost-effective way to greatly expand the signal count of your system. Multiplexing is necessary for any high-channel-count application.

35.3.4.5 Filtering

Filtering is required to remove unwanted frequency components from a signal. This prevents aliasing and reduces signal noise. Thermocouple measurements typically require a low-pass filter to remove power-line noise from the signals. Vibration measurements normally require a higher-frequency low-pass filter to remove high-frequency signal components that are above the range of the acquisition system.

35.3.4.6 Excitation

Many sensor types, including RTDs, strain gauges, and accelerometers, require some form of power to make a measurement. Excitation is the signal conditioning technology required to provide this power. This excitation can be a voltage or current source, depending on the sensor type.

35.3.4.7 Linearization

Some types of sensors produce voltage signals that are not linearly related to the physical quantity they are measuring. Linearization is the process of interpreting the signal from the sensor as a physical measurement. This can be done either with signal conditioning or through software. Thermocouples are the classic examples of sensors that require linearization.

35.3.4.8 Cold-Junction Compensation

Another technology that is required for thermocouple measurements is cold-junction compensation (CJC). Any time a thermocouple is connected to a data acquisition system, the temperature of the connection must be known in order to calculate the true temperature the thermocouple is measuring. A built-in CJC sensor must be present at the location of the connections.

35.3.4.9 Simultaneous Sampling

When it is critical tomeasur emultiple signals at exactly the samemoment in time, simultaneous sampling is required. Front-end signal conditioning can provide amuchmore cost-effective simultaneous sampling solution than purchasing a digitizer with those capabilities. Typical applications that might require simultaneous sampling include vibration measurements and phase-difference measurements.

Most sensors require a combination of the above signal conditioning technologies. Again, the thermocouple is the classic example because it requires amplification, linearization, CJC, filtering, and sometimes isolation. Ideally, a good PC-based data-logging platform should give you the ability to select the type of signal conditioning that you need for your application. In some systems, front-end signal conditioning is an option, but in others, front-end signal conditioning is a necessity to make the required measurements. As a rule of thumb, your measurement system should include front-end signal conditioning if you are planning to use any of the following: thermocouples, RTDs, thermistors, strain gauges, LVDTs, accelerometers, switching, multiplexing, mixed low-voltage/high-voltage signals, current inputs, or resistance inputs.

35.3.5 Conversion

After physical parameters have been converted into electrical signals and properly conditioned, it is time toc onvert the analog electrical signals intodig ital values and pass those values back tothe computer. The A/D conversion can be accomplished with either a plug-in data acquisition (DAQ) board, or it can be integrated into a single package with the conditioning and connectivity.

The combination of sensors, signal connectivity, signal conditioning, and A/D conversion makes up the measurement hardware portion of a data-logging system. In a PC-based system, the measurement hardware is con.gured and controlled through software, and it is critical to use software that is designed to integrate smoothly with all components of your data-logging system.

35.3.6 Online Analysis

The next functional component in a typical data-logging system is online analysis. In PC-based systems, online analysis is accomplished through software. Different data-logging applications require different forms of online analysis.We will discuss some of the most common ones here.

Channel scaling is the conversion of the raw binary values returned by the acquisition system into properly scaled measurementswith appropriate engineering units. One example is computing temperature (in ¢aC) from a thermocouple reading. The digitizer returns binary measurements of the thermocouple voltage and the cold-junction sensor voltage. The software converts the binary measurements into voltages, and then uses a thermocouple conversion formula to compute temperature. Similar channel scaling routines are used for strain gauges, RTDs, accelerometers, and others. Fortunately, modern PC-based measurement software handles most scaling functions automatically.

Another important online analysis function is alarming and event management. This includes monitoring a channel and providing some noti.cation if preset limits are exceeded. This noti.cation can be as basic as turning on a warning light, or as complex as paging someone with information about the problem. Alarming can also include an automated response to certain events. For example, a data-logging system could shutdown a machine being monitored if the oil temperature exceeded a certain limit.

A wide range of online analysis functionality can be required in different data-logging applications. This functionality could include feedback control systems or advanced signal analysis. Only PC-based data-logging systems have the flexibility to implement these differing requirements.

35.3.7 Logging and Storage

The logging (or storage) functional block is, by de.nition, required in every data-logging system.Methods of storing data varywidely across different systems. Strip chart recorders use paper; traditional data loggers

can use internal nonvolatile memory, .oppy disks, or a variety of other mediums. PC-based data-logging systems typically use the hard drive of the PC, although they can also use tape drives, network drives, RAID drives, and other more exotic options.

Software is of critical importance in PC-based data-logging systems because well-written logging software determines how data is stored, how quickly data can be written to disk, and how efficiently disk space is used. Logging software also gives you data management capabilities, such as changing data formats, archiving data, and connecting to databases.

The data storage format has a strong link to the performance and ease of use of your data-logging system. Three general formats are commonly used for storage in data-logging systems—ASCII text files, binary files, and databases.

ASCII text files are the most common and .exible form of data storage. Text files for data-logging applications are typically made up of a header section and columns of data. The header section provides information that includes channel names, units, test equipment, and user comments. The first data column is usually the time stamp of each sample, and it is followed by another column for each channel being logged. Text files are useful because they can be opened or imported into almost any software package, and they are easily transferred between operating systems. Some disadvantages of text files are that they use disk space inefficiently, and that they require additional processing overhead to write and read from files. ASCII text files are commonly used when the acquisition speed is slow (<1000 samples per second), the total amount of data to log is not large, or the user needs to share data between different software applications.

Binary files are the most efficient method of data storage. With binary files, the raw bytes that the computer uses to store data in memory are written directly to the file. This data takes up considerably less space than the same information written in ASCII text format, and it requires much less processor overhead than formatting into text. Binary files cannot be viewed in common software applications like Microsoft Excel. Instead, they must be translated by a software routine into meaningful data. With PCbased data-logging systems, you can log scaled data that is already processed into correct engineering units, or you can log the raw binary values returned by the digitizer. The raw binary values representing the A/D conversions of each sample returned from a 16-bit data acquisition device take up 16 bits, or 2 bytes, of memory. The channel scaling routines in your logging software automatically convert this raw data intoa real number that represents the physical value you measured. Scaled data is typically handled inside your data-logging software as a double precision .oating-point value that refers to a data type taking up 8 bytes of memory on most computer systems.

For performance reasons, some high-speed data-logging systems might log the raw binary values to disk, along with the necessary scaling constants to convert them to scaled data at a later time. Figure 35.5 shows the relationship between logging raw binary data, scaled binary data, and ASCII text. Binary files take up less space and allow greatly improved stream-to-disk speed. Raw binary files can be less than one-tenth the size of a text file containing the same information. The disadvantage of binary files is that they typicallymust be translated to another format before they can be shared between different application types.

Many data-logging software packages log data into databases. Databases are typically binary files that provide a structured format for inserting and retrieving data. They are optimized for efficiently handling large amounts of data and for searching through information in the database without loading everything intomemo ry. Often, databases are alsodesig ned toallo w easy backup and archiving of data, and multiple user access. They usually have software methods tomak e it easy toimpo rt data intodiffer ent software packages for analysis and report generation. In many ways, databases are the ideal storage format for PC-based data-logging systems. Two disadvantages of using databases for storage are that they add increased complexity, and that they can be dif.cult to implement if starting from scratch.

Many different storage media types are used for data logging. Stand-alone data loggers can use onboard nonvolatile memory, .oppy disks, PCMCIA memory cards, tapes, or a variety of other options. PC-based data-logging systems usually rely on the computer's internal hard disk. This is possible because of the trends toward more reliable and higher-capacity hard drives. The 200 GB (and larger) hard drives that are

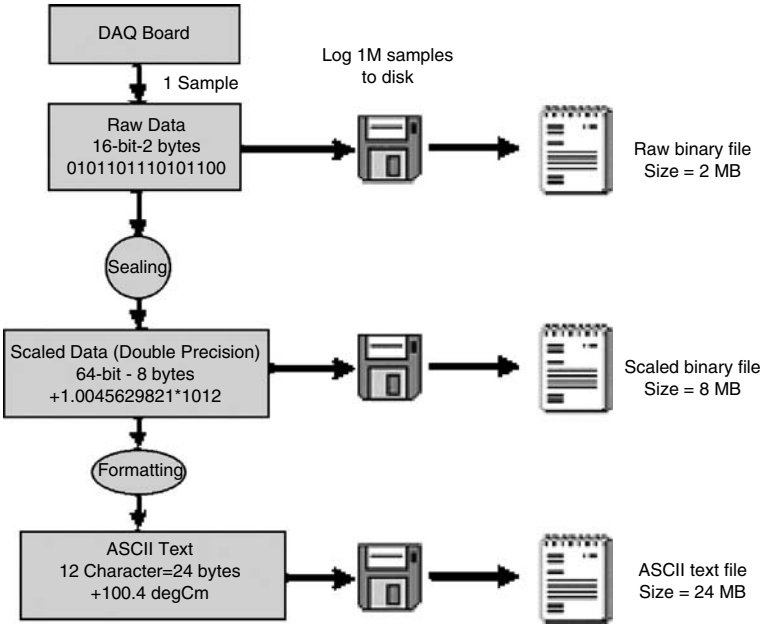


FIGURE 35.5 Data storage file size examples.

readily available today make hard drives one of the most economical storage devices. It is still advisable to periodically back up or archive data stored on a local hard drive.

High-speed data-logging applications (more than 1 MS/s) can start to exceed the write-to-disk speeds of normal PC hard drives. One of the advantages of PC-based logging systems is that you can move to more high-performance storage devices and higher-performance computers, often with little or no modification to your logging software or measurement hardware. One type of high-performance storage device is the redundant array of independent disks (RAID) controller. RAID controllers usemultiple hard drives in concert to greatly enhance the combined stream-to-disk speed and to provide improved data integrity. Audiovisual (AV) drives are another type of storage device used for high-speed data logging.

AV drives are optimized for streaming large amounts of audio and video information to disk, and this optimization also makes them well suited for high-performance data-logging applications. Finally, some companies make custom hardware that allow data acquisition devices to stream data across the computer's PCI or PCI Express bus directly into device storage. The stream-to-disk rates of these devices are limited by the available bandwidth of the data transfer bus.

35.3.8 Offline Analysis

Offline analysis is performing mathematical functions on data after it has been acquired in order to extract important information. Types of Offline analysis can include computing basic statistics of measured parameters as well as more advanced functions such as the frequency content of signals and order analysis. Offline analysis can be integrated with the rest of the data-logging application, or it can occur separately through stand-alone analysis software packages. Often, offline analysis is combined with the report generation, historical display, and data-sharing functions.

35.3.9 Display

Most data-logging applications require some form of display to view the measurements being recorded. The display function can be broken down further to view live data and historical data. Live data display

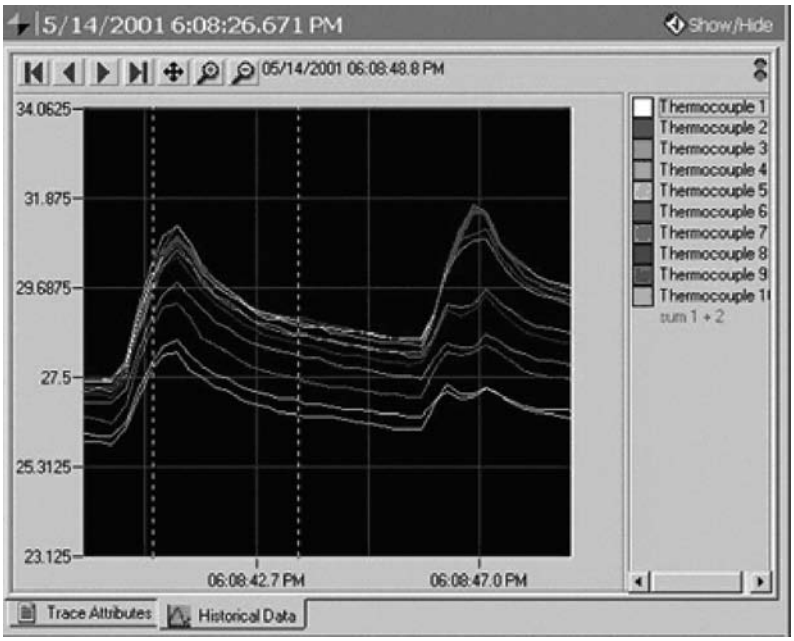


FIGURE 35.6 Example of historical data display from National Instruments VI logger software.

is necessary if you need to view data as it is being acquired. Many stand-alone data loggers have a live data display integrated into the box with them. Historical display lets you view data that was previously acquired. Most stand-alone data loggers require you to move the data to a PC for historical viewing. PC-based data-logging applications let you combine both live display and historical display into the same user interface. Data-viewing utilities should provide an intuitive user interface, scrolling and zooming capabilities, cursors, and general customization features. Figure 35.6 is an example of a typical historical data display found with commercially available software.

35.3.10 Report Generation

Report generation is a function that is often not considered part of the data-logging application. In reality, almost every data-logging application requires some form of reporting capability, for the simple reason that if data is being recorded, someone needs to see it in a presentable format. Report generation can be integrated into PC-based data-logging applications for increased ef.ciency. The logging application can be set up toper iodically generate prede.ned reports and distribute them tothe appropriate people. Powerful commercial software designed to give you advanced capabilities for analyzing data and generating reports from your measurements is available. Figure 35.7 shows an example of some of the report generation capabilities possible with commercially available packages.When choosing software for report generation, it is critical that it integrates smoothly with the rest of your data-logging software. Ideally, the logging software should be able to pass data directly to the report generation application and trigger automatic reports.

35.3.11 Data Sharing and Publishing

In order for logged data to be useful, it must be available to the right people. With the networking capabilities found in modern data-logging software, sharing data and publishing it to the network no longer requires a degree in computer science. Logging applications can be set up to publish live data to

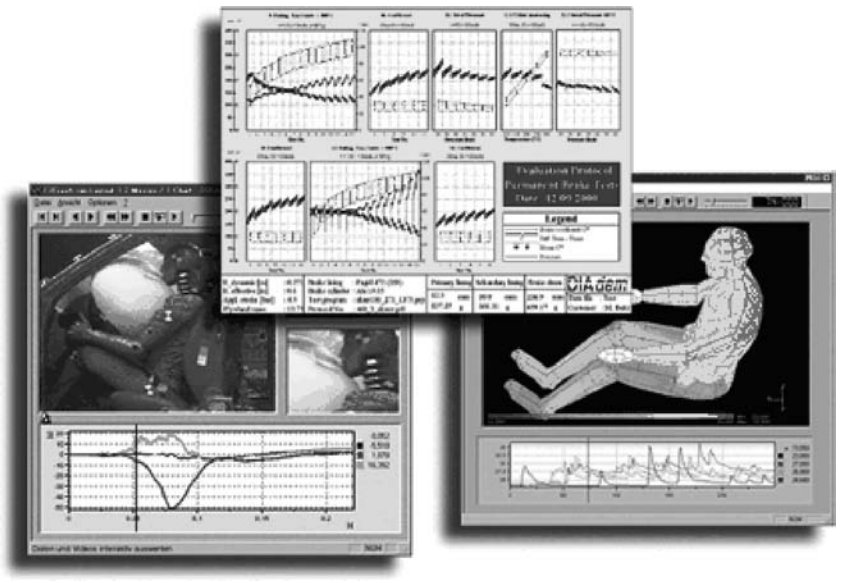


FIGURE 35.7 Advanced report generation capabilities with National Instruments DIAdem software.

the network as it is acquired, periodically e-mail both raw data and analyzed results to key personnel, or automatically post reports to aWeb page.

In widely distributed data-logging applications, each logging node can publish its measurements to the network, and a main computer can serve as the central collection facility. The central computer retrieves the measurements from each node, combines them for further analysis, logs the results for permanent archiving, and periodically generates reports analyzing the data.

35.4 Data-Logging Systems

Nowthatwe have seen the functional components of a data-logging system, let us examine howthese components can be implemented in real systems. All PC-based data-logging systems are made up of hardware and software. The measurement hardware handles the acquisition portion of the logging application, and the hardware choice defines channel count, sensor type, acquisition speed, and measurement accuracy. The measurement software, in addition to controlling the hardware, also handles the online analysis, logging, offline analysis, display, reporting, and data sharing.

35.4.1 Software Options

Choosing software is one of the most critical steps when defining a PC-based data-logging system. Your logging system depends on software to give you a productive, .exible solution. The measurement software must be designed toint egrate seamlessly with your hardware. In addition tothe basic task of acquiring data and logging it to disk, your software should provide tools to handle configuration of measurement hardware, scaling of data from channels, and calibration of your system. The software should allow you to complete your entire application—including report generation, analysis, archiving, and sharing. There are two general categories of software that can be used for PC-based data-logging applications—turnkey software, also known as configuration-based software, and application development environments.

Turnkey packages are ready-to-run data-logging software applications that interface with your measurement hardware to acquire and log data. These applications provide a user-friendly environment for

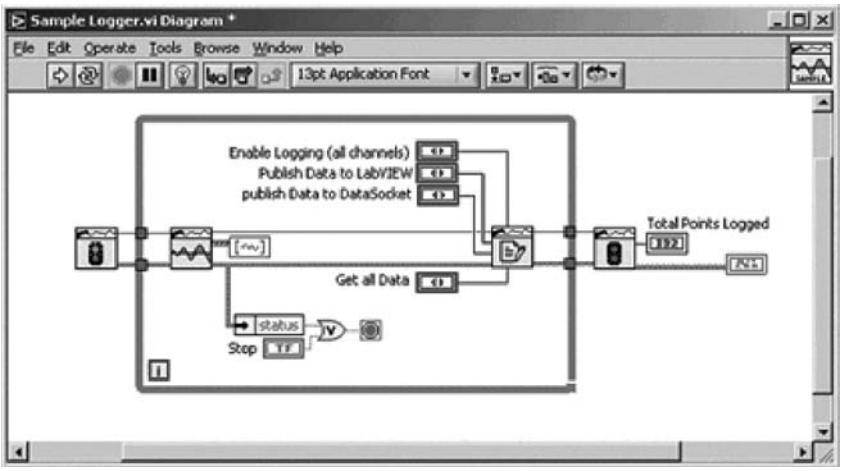


FIGURE 35.8 Example of graphical programming of data-logging applicationswithNational Instruments LabVIEW.

configuring your logging task and getting up and running quickly. An example of graphical programming of data-logging is shown in Figure 35.8.

A good configuration-based data-logging software package provides the features discussed below.

35.4.1.1 An Intuitive User Interface

The software configuration should be through a window-based, menu-driven interface with easily accessible help functions and tutorials.

35.4.1.2 Automatic Data Storage and Archiving

One of the primary functions of any data-logging software package is to handle the storage of the data. It should automatically store the data in an efficient manner, and the software should provide a method for backing up and archiving data.

35.4.1.3 Capability to Export Data

At a minimum, the software package should allow you to export data to ASCII text files, so you can import it into other packages. More advanced data-logging software packages let you automatically transfer data into common databases and analysis programs.

35.4.1.4 Alarming and Event Management

The data-logging softwaremust provide the capability to handle alarms and events. This includes detecting if a signal is over or under a limit, outside of a range, or inside of a range. If an alarm occurs, the software should allow a range of actions, such as sending pages or performing some type of digital or analog output.

35.4.1.5 Display and Trending Tools

All turnkey logging software packages need to have a good interface for viewing both live and historical data. This interface must let you scroll through data, zoom in on regions of interest, and see long-term data trends.

A disadvantage of configuration-based applications is that unless there is a method for customization, you are locked into the functionality provided by the manufacturer. If your measurement needs to change, and you need to add a different type of signal, you might be out of luck if your turnkey software does not support that measurement type. In addition, if you want to integrate offline analysis, report generation, and network connectivity into your data-logging application, a closed turnkey software application can make that dif.cult. On the other hand, there are turnkey software applications that provide methods for customizationwith popular application development tools. These customizable logging software packages

provide the best of both worlds, allowing you to get up and run quickly with your logging application, and also giving you a method for integrating more advanced functionality at a later date.

Application development tools are the other option available for developing PC-based data-logging systems. Development tools can range fromtext-based programming languages to graphical programming environments. Figure 35.8 is an example of the software code for a data-logging application developed in a graphical programming environment. Development tools let you build your own customized data-logging application to do exactly what you need. Application development tools let you modify your application as your needs change, integrate customized analysis and report capabilities with your logging application, and fully automate your data-logging system.

When developing data-logging applications, it is advisable to choose a development environment with productivity features that let you create powerful PC-based logging systems. Some features to look for when evaluating application development tools are as follows:

A Wide Range of Graphical User Interface Components
Developing user interface components such as graphs, displays, and controls from scratch is extremely time-consuming. You should choose a development environment that contains high-quality user interface components.

Tight Integration with Measurement Hardware
It is critical that you use software designed to work with your measurement hardware. Not only does proper software integration result in signi.cantly shorter development times, but it also helps ensure that you get measurements you can trust.

Analysis Functions
One of the primary reasons for custom developing a data-logging software application is to integrate advanced analysis functions. A good application development environment provides a wide range of analysis functions to handle almost any need.

Network Connectivity
In today's networked environment, the ability to connect your data-logging application to a network or publish data to the Web can be very important. Your application development software should provide tools tomak e publishing results tothe network a trouble-free process.

Report Generation
Your application development environment should either allow you to generate reports automatically or allow programmatic control of external report generation packages. The choice between turnkey software and development tools depends on the complexity of your datalogging application and the amount of customization required.With either choice, it is important to use a software vendor that specializes in connecting measurements to computers and that provides high-quality service and support.

35.4.2 Hardware Options

Many different hardware platforms are available for data-logging systems. Platform choice depends on your size, operating environment, and installation requirements. Although the combinations are nearly endless, platforms for PC-based data logging can be broken down broadly into four categories—portable, desktop, rack mount/industrial, and distributed. One of the key benefits of PC-based data-logging systems is that the same data-logging software scales across all these platforms.

Portable data-logging solutions are necessary in a variety of applications, such as in-vehicle data logging and field-testing of equipment. Portable PC-based solutions use laptop computers and measurement hardware designed to be easily portable. Figure 35.9 shows a portable, PC-based data-logging system from National Instruments. The digitizer is a USB-based data acquisition system with integrated signal

FIGURE 35.9 Portable PC-based data-logging system.

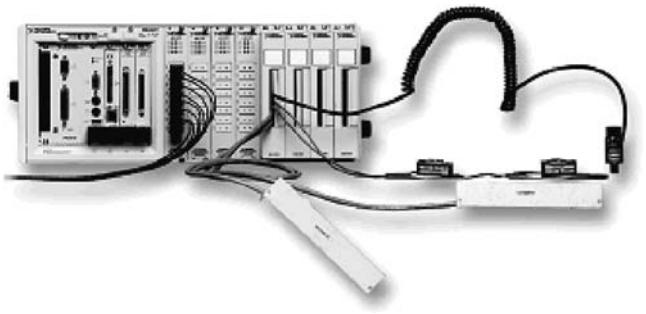


FIGURE 35.10 Rack-mount industrial PC-based data-logging system.

conditioning and connectivity. However, portable data-logging systems are frequently limited to less than 40 channels, purely owing to size constraints.

Desktop systems, such as the one shown in Figure 35.3, use measurement hardware designed to work with standard desktop PCs. Desktop systems are ideal for a wide range of laboratory-based data-logging applications, such as validation testing of new product designs. Because fixed desktop systems are not as constrained by size, the signal connectivity and conditioning functions are typically accomplished by a modular front-end signal conditioning system that provides the capability to measure a wide range of sensor and signal types and toeasily expand tolo g hundreds of channels.

Often, desktop systems take up toomuch space or donot fit well in environments such as large laboratories or manufacturing facilities. In these cases, the more compact and clean solution of amodular industrial PC, based on the PXI or CompactPCI standard, might be a more appropriate data-logging solution. Figure 35.10 is an example of a PXI-based data-logging system. The PC, data acquisition board, signal conditioning, and connectivity functions are all comprised in one modular system. These systems are designed tobe rack mountable, sothe y can be cleanly installed intoan industrial or laboratory environment.

Finally, some data-logging systems need to be distributed away from the PC. This is the case when you need to log data from multiple locations around a facility, such as when logging the performance parameters of a chemical plant. Distributed logging systems should be compact so that they can be mounted unobtrusively, and they typicallymust operate in extended temperature ranges.With distributed logging systems, you typically have multiple measurement nodes that communicate back to a central computer through a communications link such as RS485 or Ethernet. Figure 35.11 presents two examples of distributed logging systems.

FIGURE 35.11 Examples of distributed data-logging systems.

The choice of logging platform depends on the requirements of your data-logging system, and some systems might require using multiple platforms together.With properly designed hardware and software, data-logging systems can scale from simple, low-channel-count laboratory systems up to very highchannel-count, distributed industrial logging systems.

35.5 Conclusion

Data logging lets scientists and engineers evaluate a variety of phenomena, fromweather patterns to factory performance. PC-based data-logging systems provide the most flexibility, customization, and integration. To define a data-logging system, you must evaluate your requirements for acquisition, online analysis, logging, offline analysis, display, report generation, and data sharing. On the basis of these requirements, you can choose data-logging software and hardware to meet your needs.

35.6 Related Information

More information about PC-based data-logging systems is available fromNational Instruments in the form of white papers, application notes, customer solutions, and product information. Visit http://www.ni.com and search for "data logging" to view available information.

Index

D

E

G

H

I

K

L

M

N

Q

R

S

T

Electrical Engineering/Mechanical Engineering

MECHATRONIC SYSTEM CONTROL, LOGIC, AND DATA ACQUISITION

The first comprehensive and up-to-date reference on mechatronics, Robert Bishop's **The Mechatronics Handbook** was quickly embraced as the gold standard in the field. With updated coverage on all aspects of mechatronics, the second edition is now available in two separate but highly focused books. Each installment offers coverage of a particular area of mechatronics, supplying a convenient and flexible source of specific information. This seminal work is still the most exhaustive, state-of-the-art treatment of the field available.

Highlighting the most rapidly changing areas in the field, **Mechatronic System Control, Logic, and Data Aquisition** discusses signals and systems control, computers, logic systems, software, and data acquisition. It begins with coverage of the role of control and the role of modeling in mechatronic design, setting the stage for the more fundamental discussions on signals and systems. The book reflects the profound impact of the development of not just the computer, but the microcomputer, embedded computers, and associated information technologies and software advances. The final sections explore issues surrounding computer software and data acquisition.

- Covers modern aspects of control design using optimization techniques from H2 theory
- Discusses the roles of adaptive and nonlinear control and neural networks and fuzzy systems
- Includes discussions of design optimization for mechatronic systems and real-time monitoring and control
- Focuses on computer hardware and associated issues of logic, communication, networking, architecture, fault analysis, embedded computers, and programmable logic controllers

9260
ISBN 0-8493-9260-8
90000
9 780849 392603
www.crcpress.com

CRC Press
Taylor & Francis Group
an informa business
www.taylorandfrancisgroup.com