Programmable Automation Technologies

An Introduction to
CNC, Robotics and PLCs

Daniel E. Kandray, P.E.

Industrial Press Inc.

New York

Preface

For many years I taught an engineering technology course on robotics and flexible automation. I found that books that covered the fairly familiar concept of robotics were available, as were books that did an excellent job with computer numerical control (CNC) and programmable logic controllers (PLCs). However, books that truly addressed flexible automation were not so easy to find. In fact, it was very difficult to find a single text that incisively and usefully addressed all these engineering technology topics. So, throughout the years I collected and organized necessary and important information concerning flexible automation, from various sources, and disseminated it to my students. Armed with these notes, students would not need to purchase several books that would cover the course topics. Eventually, I decided to write the present book; with it I hope to fill a significant void in the literature.

Flexible automation is the use of a conglomeration of manufacturing equipment organized or connected into a single entity called a *manufacturing cell*. Manufacturing cells contain an assortment of material handling equipment, including robots and CNC processing equipment. Most often the cell's activities are orchestrated and directed by a PLC. The robot, CNC equipment, and PLC make the cell "flexible," as they can be programmed and reprogrammed to perform a wide variety of tasks and produce different products. This single text addresses all three technologies of robotics, CNC, and PLCs.

Yet, "flexible automation" is, in fact, a misnomer. While it is true that the term is appropriate for a specific manufacturing cell in which the technologies are employed, when grouping robotics, CNC, and PLCs under a collective banner, one should highlight what these technologies have in common—namely, "programmability." Therefore, these technologies are collectively named "programmable automation technologies," and this book is so titled: *Programmable Automation Technologies: An Introduction to CNC, Robotics and PLCs*.

While I was writing this text, the nation's—indeed the world's—economy plunged into a severe recession. To rise from the current economic turmoil the manufacturing industry must become more productive, a goal that is readily achievable through automation. Programmable automation technologies are the building blocks from which all automation is developed. Hence, the urgent need to improve productivity and become more competitive in the global economy should motivate a significantly greater interest in programmable automation.

The present text is organized into a four sections, which follow a logical sequence of inquiry. The first section is introductory: Chapter 1 provides some background on manufacturing and defines programmable automation. Chapter 2 explains calculation methods used to justify automation expenditures, as motivated by productivity concepts. The second section treats computer numerical control: Chapter 3 introduces CNC

technology, Chapter 4 discusses CNC programming, and Chapter 5 addresses CNC simulation. Robotics is covered in the third section in much the same way that CNC was covered in the second section: Chapter 6 introduces robotics technology and Chapter 7 goes over both robotic programming *and* simulation. (Note that robotic simulation does not have a dedicated chapter.) The last section of the text addresses PLCs: Chapter 8 introduces PLCs and Chapter 9 covers programming and simulation of PLCs. Finally, Chapter 10 concludes the text with a discussion of how all three technologies are brought together to create a programmable automation cell.

Engineering technology students at two- and four-year colleges comprise the book's primary audience. However, anyone with a technical background and a general understanding of manufacturing and manufacturing processes will find this text useful, as well as to those who wish, simply, to study and understand the use of these technologies.

Engineering technology is an applied science, so its students need to learn much more than theory: They need also practical knowledge, skills, and abilities that will allow them to readily apply automation technology. For this reason, the text offers plentiful examples and identifies and discusses readily available simulation software with which the reader can experiment.

I welcome and look forward to feedback from students, instructors and the general reader. Please write to me at info@industrialpress.com and the publisher will forward your messages to me.

Dan Kandray
April, 2010

TABLE OF CONTENTS   vii

Chapter 1

Introduction to Programmable Automation

Contents

Objective

The objective of this chapter is to introduce the reader to programmable automation, define automation in general, and to introduce the ideas of when and where automation is applied.

1.1 Introduction to Programmable Automation

Programmable automation technology is a remarkably useful tool for manufacturing engineers/technologists who seek to improve *manufacturing systems* of their respective industries. It combines mechanical, electrical, and computer technologies that have been developed for very specific automation capabilities. The term "programmable automation technology" actually refers to three individually distinct technologies that have a common thread: programmability. These technologies are *computer numerical control (CNC) technology*, *robotics technology*, and *programmable logic control* (*PLC*). Each, in some form, either directly or indirectly, is used in almost all modern automation systems—one is unlikely to walk into a modern manufacturing facility without observing one of these technologies in action. This, however, has not always been the case.

Initial migration to programmable automation was gradual, hampered by the complexity, expense, and, in some cases, poor reliability of early systems. Additionally, utilization of the technology required that companies retain technical experts specifically devoted to the implementation, programming, and maintenance of the systems. However, over the last 25 years programmable automation technology has greatly matured: Modern systems are standardized, substantially less complicated and expensive, and extremely reliable. Whereas use of this technology was initially the domain of specialists—particularly engineers—now virtually every member of an engineering or maintenance staff is expected to use the technology at some level. In fact, it is now imperative that mechanical engineers and technologists—who used to avoid "electrical stuff"—have a solid foundation in it. Hence, the goal of this text: to instruct any member of an engineering team so he or she may comfortably delve into the automation arena.

Before we properly define "programmable automation" and develop a full description of its capabilities, we first must present an understandable picture of manufacturing in general. In the following section we explore manufacturing and define some of its key terms. Subsequent sections define automation in general and programmable automation in particular. The concept of *productivity* will be introduced, and the last few sections will address reasons for automation, corresponding benefits of it, and ways automation can be implemented.

1.2 The Manufacturing Process

1.2.1 Manufacturing overview

Manufacturing, regardless of the industry under consideration, is a *conversion process*. Some form of raw material is brought into a manufacturing facility and converted into a more useful finished product. The conversion is accomplished by applying a series of manufacturing steps, or *manufacturing processes*, to the raw material. Manufacturing processes alter the raw material's shape, appearance, physical and mechanical properties, and/or assemble it with other components into a desired finished product. This is achieved

through the use of equipment, tools, and supplies combined with the application of labor, time, and energy.

The way that manufacturing operations are organized within the facility defines *plant layout*. The term "manufacturing system" of the facility refers to the plant layout and worker execution of the operations. The manufacturing system used is determined by the product(s) characteristics. Figure 1-0 shows the plant layout of an imaginary facility, the XYZ Company. XYZ Company manufactures widgets. As the layout indicates, bar stock is processed into a widget through a series of manufacturing processes, which include sawing, turning, milling, and painting. Each manufacturing process executes a systematic sequence of operations called a *program of instructions*. When a program of instructions is complete for one manufacturing process, the product is routed to the next process.

Figure 1-0 XYZ Company plant layout

Note that the factory must perform operations in addition to manufacturing processes to create the product. The product must be moved between the manufacturing processes. It must also be inspected at some point to ensure that it satisfies the customer's requirement. Additionally, someone must optimize the processes, schedule the operations, monitor labor usage, schedule maintenance, coordinate material handling, control inventory, and make sure the product is shipped on time. These activities do not contribute to the conversion process of the product per se. However, they are critically important to the manufacture of the product. The manufacturing processes and other activities combined are called *manufacturing operations*. Typical manufacturing operations found in factories include:

- Manufacturing processes
- Material handling
- Quality control
- Manufacturing support.

Each *manufacturing process* is designed to accomplish a very specific raw material conversion step. Thus, the number of manufacturing processes and the way they are organized within the facility are determined by the product(s) made. Manufacturing processes might include shaping processes such as molding or machining, property-enhancing processes such as the heat treating of steel, surface processes such as cleaning, coating, or painting, and various types of assembly processes. Assembly processes can be permanent, as in the case of welding, soldering, brazing, or adhesive bonding. However, some assembly processes are considered *semi-permanent*. Semi-permanent assembly processes typically include various types of mechanical joining, such as what is accomplished with the use of threaded fasteners, rivets, and expansion fits. All manufacturing processes are said to *add value* to the product; but, the other three operations—*material handling, quality control, and manufacturing support*—do not add value and are often the first targets for automation.

The way manufacturing operations are organized within a facility defines its *plant layout*. The term "manufacturing system" refers to both plant layout and worker execution of operations.

1.2.2 Defining the product

A given manufacturing facility may turn out only one product, a variety of models of one product, or many different products. Products may be as simple as a paper clip or as complex as a photocopier. Additionally, the facility might make only one product per year, or it might turn out millions of products. A product is either *continuous*, such as a liquid like gasoline, or it is *discrete*, like an automobile. (This text focuses on programmable automation of manufacturing processes and systems for discrete products only; continuous product manufacturing processes and systems will not be addressed.) Taken together, these distinctions make up the *product definition* and are naturally related to the manufacturing system(s) used within the facility.

The choice of manufacturing system employed for a discrete product is a function of the *manufacturing product definition*, which encompasses three main factors:

- Product complexity
- Product variety
- Product quantity.

These factors provide the most complete picture of the type of manufacturing system(s) needed to make a product economically.

The level of a product's *complexity* is tied to the level of difficulty in the manufacture of that product. In general, product complexity is an indication of whether the product is a small, simple, single component, as is the case with the widget made by the XYZ Company, or a large and complex product, like a nuclear submarine, which has numerous complex individual components. Obviously, the manufacturing systems that would

produce these two products would be vastly different. This product complexity is further illustrated in Figure 1-1.

Paperclip

Laptop Computer

Low Product Complexity   High Product Complexity

Figure 1-1 Product complexity example

Product *variety* refers to the number of different product designs, versions, or models to be produced within a facility. If a facility made just a single product, such as a toothbrush, it would use a manufacturing system conducive to efficient production of that one product. Conversely, if the facility were to manufacture, say, hairbrushes in addition to toothbrushes, it might have multiple manufacturing systems or a single system designed to accommodate this product variety.

In addition to this factor, there are different *levels* of product variety. *Soft product variety* indicates that product difference within the operation is small. *Hard product variety* means the products are vastly different. The best way to differentiate soft from hard product variety for assembled products is to consider the number of common parts each product uses. A high percentage of common parts indicates soft product variety, whereas a low percentage points to hard product variety. The case of the toothbrushes and hairbrushes represents hard product variety. Soft product variety might exist in the production of different models or styles of toothbrushes (color, bristle density, and so on). An illustration of this case of hard versus soft product variety is shown in Figure 1-2.

Soft Product Variety   Hard Product Variety

Figure 1-2 Hard versus soft product variety

Product *quantity*, naturally, specifies the number of products that are to be turned out over a given time period. This is more often referred to as *product volume*. As will be seen in subsequent sections, high volume product requirements dictate the use of automated high speed manufacturing systems. Low volume manufacturing systems are typically less automated with more worker involvement.

Observe that product definition and manufacturing system are interdependent. High volume manufacturing systems are typically less capable of accommodating product variety. Conversely, manufacturing systems capable of handling a great deal of product variety cannot produce as high a volume. Complex products dictate a complex manufacturing system; simple products will typically be turned out by a relatively simple manufacturing system. Thus, once a manufacturing system is established within the facility it will dictate the type of future products that can be produced, unless, of course, capital is allocated to add other manufacturing systems to the facility.

1.2.3 Manufacturing system

As mentioned already, the manufacturing system is the combination of manufacturing processes and the organization of workers, designed so as to efficiently and effectively create the desired product. There are essentially four standard systems, with numerous variations, that have evolved over time. Each system is geared to produce a fairly specific product definition. The four standard manufacturing systems are:
1. Fixed-position
2. Process
3. Quantity manufacturing system
4. Flow-line.

(Refer to Figure 1-3.) Because of the narrowness of product definition relating to each system, an individual facility may employ one or more of these systems.

Fixed-position manufacturing systems (Figure 1-3(a)) are used for making large, complex products. Because such products are not easily moved, the manufacturing processes are taken to the product; the product remains in a fixed position throughout manufacture. This system produces items that are highly complex, relatively large and immobile, with low volume production requirements, and of soft product variety. Submarines, ships, and large aircraft are examples of products that use this system.

The *process* manufacturing system is used when product complexity is relatively low and there is hard product variety. Accordingly, production quantities range from low to moderate. This system is also called a *job shop system* because it can accommodate a wide variety of products or jobs. The manufacturing processes are grouped together according to function or process. Products are routed though the facility to the required manufacturing processes in groups called *lots* or *batches*. The size of the lot, or *lot size,* is the number of products in the group. The use of lots is necessary because of the variety of products the system must accommodate. In a process manufacturing system each

a) Fixed Position - Submarine

b) Process

c) Quantity

d) Flow - line

Figure 1-3 Standard manufacturing systems

manufacturing process to which a product is routed will have to be *set up*, or prepared, to process that particular product. Consequently, it is economically desirable to run a specific number of products through the process each time it is set up. Some factors that influence lot size include variety of the products made in the facility, setup times, order sizes, and manufacturing lead-time. This system is shown in Figures 1-0 and 1-3(b).

The *quantity* and the *flow-line* manufacturing systems are often combined into one category, called the *product manufacturing system*. These systems produce mass quantities of products and are thus mass production systems. However, both product complexity and product variety of mass-produced items dictate the division of the product manufacturing system into the quantity and flow-line manufacturing systems.

The quantity manufacturing system is used with low product complexity and hard product variety. These products are produced on a single standard machine, like a plastic injection molding machine with exchangeable tooling. (Refer to Figure 1-3(c).) Other examples of products manufactured with this system include metal stampings and blow molded plastic products, like water bottles.

Flow-line manufacturing is for products with high product complexity and soft product variety. For assembled products, flow-line is more commonly called *assembly line* manufacturing system. Raw material flows down a line of manufacturing processes, in the end to be converted into a finished product. Henry Ford is often credited with perfecting this system. He produced the Model T using the flow-line manufacturing system to reduce cost so that the average American could afford his automobile. Because of the nature of the flow-line manufacturing system, product variety was essentially nonexistent. All Model T cars had the same body shape and were made in one color. The early flow-line system evolved to a modern form that accommodates soft product variety. Figure 1-3(d) illustrates the flow-line manufacturing system.

The manufacturing systems discussed above are summarized in the table of Figure 1-4 as shown on page 9 according to product complexity, variety, and volume. Also, it cannot be emphasized enough that many manufacturing facilities will have more than one of these systems and/or a variation of some system as dictated by the product definition. Consider Figure 1-6. This figure demonstrates how some manufacturing systems feed into other manufacturing systems to produce the finished product. In this example a flow-line manufacturing system in the form of an assembly line produces the finished product. The components of the finished products are produced on various other manufacturing systems as shown. This is typical of most, if not all, assembled products.

Manufacturing System	Product Complexity	Product Variety	Product Volume
Fixed Position	High	Soft	Low
Process	Low	Hard	Moderate
Quantity	Low	Hard	High
Flow-line	High	Soft	HIgh

Figure 1-4 Manufacturing systems versus product complexity, variety, and volume

Figure 1-5 Manufacturing support systems

Figure 1-6 Multiple manufacturing processes used to make a product

1.2.4 Manufacturing Support Systems

Another key ingredient to the conversion process is the *manufacturing support system*. Manufacturing support systems provide the management of the business operations of the facility and the manufacturing system. Thus, the success of a facility, in terms of productivity and thus profitability, is dictated by how well its manufacturing support systems manage its manufacturing system.

A manufacturing support system utilizes *people* and *procedures* to manage the manufacturing system and the overall facility. Whereas the manufacturing system processes the *raw material*, the manufacturing support system processes *information* necessary to accomplish the conversion of the raw material into the finished product. Accounting, customer service, marketing, human resources, product design, manufacturing engineering, materials engineering, quality control, production planning, and shop floor control are all good examples of manufacturing support systems. Figure 1-5 as shown on page 9 demonstrates how manufacturing support systems interact with the manufacturing system.

Through understanding the different manufacturing systems and manufacturing support systems, as well as how the systems interact, the seeds for automation are sown. The next section will define automation in more precise terms and identify specific types of automation.

1.3 Automation

Webster's Online Dictionary (http://www.websters-online-dictionary.org/ definition/ automation) defines *automation* as "a highly technical implementation; usually involving electronic hardware; automation replaces human workers by machines." In his first book *Automation, Production Systems and Computer-Integrated Manufacturing,* 2nd ed. (2001), M.P. Groover defined automation, when directed strictly toward the manufacturing environment, as "…technology concerned with the application of mechanical, electronic and computer-based systems to operate and control production" (p. 9).

Each of these definitions provides some key terms and phrases. However, Webster's definition implies that automation replaces or eliminates workers and that automation is accomplished only with machines—statements that are misleading. Automation does not always replace the worker; it more often *displaces* the worker to other tasks. Additionally, automation can be implemented in many forms. Often it is with a machine, but it can also be a device or software added to an existing process.

For example, consider the automation of the manual drafting process. The implementation of computer-aided drafting (CAD) is a great example of the automation of a manufacturing support system. CAD automates the creation of engineering drawings, a key component in the manufacturing material conversion process. Prior to the implementation of CAD, engineering drawings were created by hand with paper and

pencil at drafting tables. The combination of computer hardware (the machine) and CAD software automated this process. Early CAD systems were two-dimensional and essentially duplicated the manual drawing process, but were much more accurate and faster than manual drafting. Human intervention was still required to operate the software, but to a lesser degree. Hence, manual drafters (the workers) were not replaced. The author observed in the plants with which he was associated that most of the drafters were trained to operate the CAD system. Since the CAD system is more productive than manual drafting, most excess drafters (workers) were displaced to other activities or other companies, and eventually an entire new generation of drafters trained primarily in CAD.

When companies first switched to CAD from manual drafting, a machine, the computer, was required. Computers have since become standard pieces of equipment in the engineering department. Hence, subsequent automation of the CAD process was accomplished only with software, an example of how automation does not always entail insertion of a machine into the process. Consider another example, solid modeling software. Solid modeling software automates the product development process, including the two-dimensional CAD drafting process. It models products as three-dimensional solid objects in the virtual world of a computer. The model enables better visualization of the product. Additionally, rapid prototypes, computer numerical control programs, and engineering drawings can all be created directly from this model.

The definitions above imply that automation only occurs in a production environment. However, automation can occur anywhere people are performing tasks. Consider a grocery store. Modern grocery stores have electronic barcode scanners to determine the price of goods purchased during checkout. The barcode scanners automated the task of a cashier manually entering the price of each item into the cash register. Now, many stores have added automated checkout, which allows the consumer to swipe the purchased goods and pay a machine directly, further reducing the level of cashier involvement. Often this allows more checkout lines than the store would normally be able to payroll. However, a worker is still needed to supervise the automated checkouts and assist shoppers as needed.

Machines and or systems that perform tasks automatically inevitably consist of some combination of mechanical technology (gears, cams, bearings…), electrical technology, and/or computer technology. Early automated machines were mostly mechanical. More modern automated equipment utilizes electrical and computer technology to a greater extent. Additionally, many automated machines or systems are combinations of many smaller automated machines. Thus, automated machines can be further automated with the application of even more technology.

The preceding discussion highlights the fact that a more encompassing definition of automation is needed than is typically found in the literature. So, the author defines automation here as follows:

Automation is the application of mechanical, electrical, and/or computer technology to reduce the level of human participation in task performance.

Note that in this definition "task" is an intentionally vague term. Tasks are not limited to work-related activities. They can be related to any activity requiring human participation. Consider the television remote control, for example. It automated the task of manually changing the television channel, a purely entertainment-related activity. The definition also makes clear that humans are not necessarily replaced. Their level of participation may be greatly reduced, typically displacing them to other activities. This point is important because often great fear exists in the workforce when system automation is considered. Workers inevitably assume that such automation will eliminate jobs, which, as discussed, does not always occur, particularly in a company that values its workforce. Thus, the distinction between "displacing" workers and "replacing" them needs to be emphasized. The other notable aspect of the definition is that it emphasizes that automation can be implemented with various forms and combinations of technology.

Now that a clear definition of automation has been established, the different automation types that have evolved in manufacturing are discussed.

1.3.1 Types of Automation

Automation in a manufacturing facility can occur in the manufacturing support systems and/or in manufacturing systems. Automation in manufacturing support systems is primarily accomplished through the use of computer technology to automate the business operations of the facility. Computer-aided design (CAD) software and computer-aided manufacturing (CAM) software have dramatically impacted the way products are designed and engineered. Shop floor control systems combined with material resource planning systems provide management with a very fast and accurate picture of a facility's current status. Computer-integrated manufacturing (CIM) takes the automation of the manufacturing support systems a step further. CIM is intended to integrate and thereby automate the entire manufacturing enterprise. In other words, CIM links the automation in the manufacturing support systems directly with automation in the manufacturing systems, resulting in a completely integrated manufacturing facility. However, the focus of this text is not on the automation of manufacturing support systems. Rather, it is on a specific type of automation of manufacturing systems.

Automation in manufacturing systems is centered on reducing the level of human participation in manufacturing processes. Three standard types of automation can be defined. Each type has very specific capabilities relating to the sequence of the processing steps and the definition of the product being processed. The three types are:

- Fixed automation
- Programmable automation
- Flexible automation.

Fixed automation equipment typically consists of processing stations linked together with some form of material handling, which progressively moves the workpiece through the processing steps. Fixed automation can be considered special purpose automation because it is designed to automate a specific process or series of processes. Therefore, the processing sequence is fixed by the organization of the processing stations. In general, it is relatively inflexible in accommodating any type of product variety. However, if it is capable of handling soft product variety, conversion of the machine allowing it to run the variation may be time-consuming. The time to make this change is often termed "changeover time" or "setup time." Fixed automation can handle a wide variety of product complexity from simple to very complex products; but, the cost of creating such a specialized machine is often quite high. Consequently, fixed automation is typically only used with extremely high volume products. To accommodate the volume, these systems operate at very high rates. Hence, fixed automation equipment is most often associated with flow-line manufacturing systems, often as assembly lines. Early fixed automation was created using mostly mechanical technology combined with electrical technology to drive its mechanical components. Current systems make extensive use of computer technology and often integrate programmable automation into the machine, as well.

Whereas fixed automation is "fixed" to a specific operation progression, *programmable automation* has the capability to alter both the type of operation to be performed and the order in which it is to be executed. Thus, it can adjust the process to accommodate the product, which makes it capable of handling hard product variety. Programmable automation equipment is multipurpose equipment that can be programmed, and repeatedly reprogrammed, to perform a wide variety of processing operations. However, each programmable automation machine is limited to a specific type of manufacturing process such as machining material removal, metal forming, or material handling. Reprogramming and changing tooling is necessary to accommodate the different products, creating long setup times between products. This combination of versatility, hard product variety, and long setup times would appear to limit programmable automation to use in the process manufacturing system. It is interesting to note that this is in fact the manufacturing system for which programmable automation was developed and where it is still primarily used. However, programmable automation equipment has proven to be so very capable and reliable that it is also used in the quantity and flow line manufacturing systems, either outside of or integrated into fixed and flexible automated equipment.

Flexible automation possesses some of the features of both fixed and programmable automation, with an added characteristic of no time lost for changeover between products. It utilizes, essentially, a fixed automation machine that can process soft product variety with no setup. The elimination of the setup is achieved with the versatility of programmable automation integrated directly into the machine. The machine recognizes or identifies different product configurations and automatically adjusts the operation sequence. The exorbitant cost of such a system limits its use to high volume applications

typical of the flow-line manufacturing system. However, this added versatility means reduced production rates when compared with what is possible by a fixed automation machine.

Figure 1-7 summarizes the capabilities of each of the automation types.

Automation Type	Product Complexity	Product Variety	Product Volume	Production Rate	Manufacturing System
Fixed	Low to High	None	High	High	Flow-line
Programmable	High	Hard	Moderate	Moderate	All
Flexible	Low	Hard	High	High	Flow-line

Figure 1-7 Automation types

It should be noted that the lines of distinction between these three types of automation and the manufacturing systems in which they appear are often blurred. What should become clear is that programmable automation is at the core of both fixed and flexible automation. In fact, it has become a primary building block of almost all automated machines. Additionally, it is unlikely that a person would walk into any modern manufacturing facility and not encounter programmable automation, as it is found in practically every manufacturing system.

1.3.2 Programmable Automation

Improved productivity is the primary focus of global competition in the world economy. As will be shown in subsequent sections and chapters, automation is a key ingredient to improving productivity. Programmable automation, in some form, is found in almost all automation systems. It is used individually in process manufacturing systems, or it can be fully integrated into fixed automation machines in flow-line manufacturing systems. Flexible automation machines were not even possible prior to the development and maturation of programmable automation. Hence, it is imperative that manufacturing engineers and technologists understand the capabilities of this technology and how it may be used effectively.

Programmable automation, as is shown in Figure 1-8, has evolved into three distinct technologies:

- Computer numerical control (CNC) technology
- Robotic technology
- Programmable logic control (PLC) technology.

Computer numerical control (CNC) technology utilizes a combination of mechanical, electrical, and computer technology to move a tool relative to a workpiece to perform some

type of processing. It is most often related to the machining processes, such as milling, turning, and grinding. However, it can be used in any process that requires precise control of a tool relative to a workpiece. Non-material removal examples include wire-bending machines and pen plotters. Some CNC technology examples are shown in Figure 1-9.

Figure 1-8 Programmable

automation

Robotic technology is very similar to CNC technology in that it utilizes mechanical, electrical, and computer technology to move a manipulator in three-dimensional space. Also, in many applications it uses a tool to perform processing on a workpiece, an example of which is a welding robot: the robot moves the welding tool through a specific path over the workpiece. However, in many other applications the robot does not use a tool. It merely provides material handling capabilities such as moving a workpiece from one machine to another and/or stacking the workpiece in a specific pattern on a pallet. In either case, the robot is performing a task that could also be performed by a human. In

Figure 1-9 CNC technology

fact, the origin of the term "robot" is credited to a play, which premiered in 1921, about a factory that made artificial people devoid of feelings. These artificial people were called robots. The word was derived from the Czech word *robota*, meaning serf labor, thereby

implying servitude and hard work. Thus, robots are often distinguishable from other types of automation in that they possess humanlike characteristics (e.g., a robot arm) and perform tasks often completed by humans. Robotic technology examples are shown in Figures 1-10 and 1-13.

Cartesian Coordinate Robot

SCARA Robot

Articulated Arm Robot

Figure 1-10 Robotic technology

Whereas CNC and robotic technologies provide motion control, *programmable logic control* (PLC) technology imparts automatic control over tasks or events through the use of electrical and computer technology. This is accomplished by monitoring the status of a given system through sensors that input information to the PLC. Based on the status of these inputs, the PLC will make decisions and take appropriate action on the system by outputting information to actuators. The output to the system is based solely on the status of the inputs. This is called a *discrete process control system*. A discrete system has inputs and outputs that are binary with two possible values: on or off. The status of these inputs and outputs change at discrete moments in time. Thus, PLC technology provides control over event-driven changes to the system. As events occur that change the status of the inputs, the outputs automatically change. Figure 1-11 shows an example of a PLC.

Figure 1-11 PLC technology

These three technologies are the foundation upon which modern automation is built. The automation system shown in Figure 1-12 makes good use of all three technologies. The figure depicts a typical *manufacturing cell*. A manufacturing cell is defined as an interconnected group of manufacturing processes tended by a material handling system. In this particular case, the manufacturing processes consist of a CNC lathe and a CNC mill. These two machines are tended by a robot, which loads raw material into one machine, transfers it to the next, and then unloads and stacks the processed material. A programmable logic controller (PLC) controls and coordinates activities between the CNC

machines and robot. (Note that PLC is the acronym for both programmable logic control and programmable logic controller.)

Figure 1-12 Manufacturing cell

The CNC equipment, robot, and PLC must possess intelligence in order for the cell to function properly. The intelligence is expressed in terms of decision-making ability. Each machine in the cell must be able to accept input, make a decision based on that input, and implement the decision. For example, consider the CNC lathe. When it is time to process material, it must first recognize that it is ready to process more material and then prepare itself to receive material by opening safety gates, moving tooling out of the way, and opening the lathe chuck. Next, it must inform the PLC that it is ready to accept material for processing. Once the material is loaded, the CNC lathe must recognize that it is loaded and then process the material. When the processing is complete, it must prepare itself to be unloaded. Finally, it will inform the PLC that it is ready to be unloaded. The robot functions similarly. It receives input from the PLC that the lathe is ready to be loaded. It then executes a sequence of movements to pick up the raw material and load it into the lathe. It will then inform the PLC that it has loaded the material and is ready for the next instruction. Thus, the PLC is, essentially, the brain of the cell. It controls the timing and sequence of all events that occur within the cell. It monitors the status of the cell and informs each piece of equipment when and where actions are to be performed.

Figure 1-13 depicts another manufacturing cell. This cell consists of a hydraulically actuated press, a shuttle system, and a robot. In this cell, products (round disks) are molded inside the press then moved outside the press (with the shuttle system), where the robot unloads and finishes the product. A PLC controls the cell. Note that even though a CNC machine was not part of this particular cell, CNC technology had an impact on the

cell. A CNC machine was used to produce the mold in the press and the gripper on the end of the robot arm.

Figure 1-13 Press manufacturing cell

Thus, as shown in these two examples, it is easy to envision how programmable automation is present, either directly or indirectly, in almost all modern automation systems.

1.4 Manufacturing Performance Measures

In order to understand when and where to apply automation in general and programmable automation in particular, it is essential to comprehend how manufacturing production performance is measured. From these performance measures one can evaluate and justify the use of automation. As will be seen in the next chapter, the performance measures used most often to quantify production include:

Production rate—measure of products per hour (pc/hr).

Setup time—measure of the amount of time to prepare a machine or process to make a product (hrs).

Production capacity—measure of the maximum amount of product that can be produced by a manufacturing facility, system, cell, or process in a specified period of time (output units/time period).

Utilization—ratio of the actual amount of output from a manufacturing facility, system, cell, or process in a specified period of time to the production capacity over the same time period (%).

Manufacturing lead time—total time to process a product through a manufacturing facility, system, cell, or process.

Each of these measures provides a picture of how certain individual aspects of the manufacturing process and system are performing. These are vital and critical measures to be evaluated when one considers automation. However, each only provides a small segment of the overall picture. Thus, the measures need to be evaluated collectively to effectively evaluate and justify the use of automation. There are other factors as well that should be considered, including burden rates of equipment and labor costs. Thus, it may be difficult to get a clear comprehensive picture of the performance of a process or system. However, there is one measure that effectively combines and summarizes many of the individual measures into one all-encompassing metric: That measure is *productivity*.

1.4.1 Productivity

The term "productivity" is often cited in the news media as an important economic indicator of the health of the nation's economy. The U.S. Department of Labor, Bureau of Labor Statistics, collects and publishes productivity data for many elements of the U.S. economy. Per the Bureau of Labor Statistics website (www.bls.gov/bls/productivity.htm):

Productivity and related cost measures are designed for use in economic analysis and public and private policy planning. The data are used to forecast and analyze changes in prices, wages, and technology. (p. 1)

Thus, the measure plays an important role in the development of private sector and government economic policies. The interaction of productivity measurements on price changes, wages, and technology may be complicated, but the definition of productivity is not. While the term may have a slightly different connotation to an economist compared to an industrial engineer/technologist, the basic definition is clear. This simple definition is why this measurement, used as a means of quantifying manufacturing production, can be used in all levels of manufacturing. It is the author's contention that the productivity measure is perhaps the best indicator of when and where to utilize automation.

As David J. Sumanth observed in *Productivity Engineering and Management*:

Productivity is concerned with the efficient utilization of resources (inputs) in producing goods and/or services (output.) (p. 4)

Note that "inputs" refer to all the resources (labor, material, capital, etc.) that go into producing the goods or service, and "output" is whatever is produced by the system under consideration. Thus, the productivity of a manufacturing system can be determined simply by the ratio:

$$productivity = output/input$$

For the purposes of this text, output will be expressed in units of parts/hr and input in terms of $/hr.

In order for the productivity ratio to rise, either the output must increase (more parts/hour) or input must decline (amount of product remains the same with less resources). Productivity increases are positively viewed because they indicate that more is produced with less dollar input to the system. Therefore, if the selling price of the product does not change, the producer realizes an increase in profit. Conversely, a decrease in productivity occurs when not as much product is being produced and/or the input cost increases. Typically, all inputs (labor, raw materials, and energy costs) will increase over time. To prevent these increases in inputs from being passed along to the consumer as a price increase, more products must be produced. Note that this is the link of productivity to inflation. Therefore, it is obvious that improving productivity is of vital importance to manufacturing. Equally as obvious is how automation can improve productivity by allowing more products (output) to be made or by reducing the cost (input) of production.

The way a company can use productivity and the other manufacturing measures to justify automation will be addressed in detail in Chapter 2.

1.5 Benefits of Automation

Specific reasons to automate one process may be very different from the reasons to automate another. However, the goal of any automation is to produce a tangible benefit. Common to all automation is the benefit of productivity improvement. Listed below are seven common reasons to automate:

Increase labor output

Increasing labor output has a direct effect on increasing productivity. If, through automation, the amount of product is increased, the productivity also increases. Essentially, automation focused on improving the effectiveness of the labor, thereby increasing the amount of product made over a specific time period, will lead to increases in labor output and, thus, productivity improvements. Examples include addition of a robot to handle material or use of a PLC to control a manual process. Each is intended to "free up" the worker from a task, thereby enabling him or her to produce more.

Reduce labor cost

Reducing labor cost also has a direct effect on increasing productivity. Because labor cost is an input in the productivity formula, reducing it increases productivity. In fact, from a productivity viewpoint it is difficult to distinguish reducing labor cost from increasing labor output because the net effect is the same. However, it is listed as separate to emphasize that some automation strategies focus on improving the amount of output produced from the current number of workers whereas others specifically target reducing labor costs. Examples of labor cost reduction include any type of automation that reduces the number of workers or the time each worker spends in production.

Reduce or eliminate effects of labor shortages

Depending on the state of the local economy, a plant may have an abundance of available workers or a severe shortage. If the manufacturing process is particularly labor intensive, lack of workers can result in machine down time, less product, and overtime for the current workforce, each of which can have a detrimental effect on productivity. Making the process less labor intensive through automation allows it to better withstand periods of labor shortages. Thus, automation strategies geared to increase labor productivity and/or reduce labor costs should be considered.

Reduce or eliminate routine manual and clerical tasks

Reduction or elimination of routine tasks is often the first avenue to improving a process's productivity. Automating these types of tasks, once again, frees up the worker to perform more value-added tasks. This inevitably leads to productivity improvements through reduced labor costs and/or improved worker productivity. A good example is the automation of the engineering drawing process with computer-aided drafting (CAD).

Improve worker safety

Any opportunity to provide a safer worker environment is a worthwhile investment in any process, not only from the obvious benefit of worker protection—management's ethical responsibility—but also from a productivity standpoint. Down time due to accidents also decreases productivity by limiting output from the process. Yet, physical safeguards intended to protect the worker might also hamper output. A better approach would be to utilize automation and completely remove the worker from the dangerous work environment. This should be attempted even if a productivity gain is realized or not. An example of such precautionary automation might be utilization of a robot to remove parts from an injection press. For a worker to remove parts from the mold, the press door must be opened, which activates mechanical interlocks that prevent the mold from closing as the worker removes the parts. But, if a robot removes the parts, the worker is no longer required to reach in the press, thus removing him from the dangerous environment. Additionally, the press cycle time improves because opening and closing the door is removed from the process since the robot typically accesses the mold from the top of the machine.

Improve product quality

Improving a product's quality yields many benefits to the manufacturer, including reduced waste—a plus for both the business and the environment, which makes for better brand image and higher sales. The impact on productivity is equally impressive. Reduced waste reduces material costs, which decreases inputs to the process, thus increasing productivity.

Reduce manufacturing lead time

Manufacturing lead time is a measure of how long it takes to create a product from the time the order is received by a manufacturer until the product is shipped. If through automation the manufacturing lead time is reduced for a process or a series of processes,

output will increase over a given time period. If all other inputs remain unchanged, productivity will increase.

Some other reasons for automation often referenced in the literature, which are tied indirectly to productivity, include: (1) the high cost of not automating and (2) the existence of processes that simply cannot be done manually. The first is a somewhat obvious statement, regarding which the above list makes a strong case. In today's economy productivity improvements are perhaps the only way to remain competitive. If a company is not competitive, it will not survive. Therefore, the cost of not automating will be in terms of lost customers and profits.

Consider the second of these, performing processes that cannot be done manually. Some processes may require too high of a degree of precision or be too small for the human hand to effect or have too complex a geometry. Ponder the manufacture of computer chips. Arun Radhakrishnan, writing in http://blogs.techrepublic.com.com/tech-news/?p=2050), pointed out that in 2008 Intel announced a new computer chip containing 2 billion transistors. Obviously, this can only be produced with the aid of automated machines; without the automation to manufacture it, the product could not be made and thus productivity would be zero.

Thus, there may be many reasons to automate, but the primary benefit to automating is improved productivity. When a company continuously improves productivity it is better able to absorb raw material cost increases, labor cost increases, increased energy prices, and other inflationary types of cost pressures—without passing those increases along to the customer. Thus, by improving productivity the company may realize other benefits including higher sales, better customer relations, and a larger market share. Although automation is not the only method to improve productivity, it is often a very effective method and should therefore be strongly investigated.

1.6 Automation Strategies

The *why* and *where* of automation are listed in the preceding section. We now turn to the *how* of automation. Typically, how a plant would automate is one of the more challenging aspects of the automation implementation process. One example was given already, that of employing a robot to perform material handling or adding a PLC to control the process. It is the intent of this section to provide some strategies that can be used to determine how a particular process might be automated.

In the aforementioned *Automation, Production Systems, and Computer-Aided Manufacturing*, *2nd ed.* Groover introduced 10 strategies concerning how automation can be applied to manufacturing processes. Based on this list, five condensed "how to" strategies, geared specifically for programmable automation, are given here:

Minimize manufacturing process steps

As explained, a series of manufacturing process steps convert raw materials into some other higher value form. This strategy seeks to minimize the number of process

steps. It does so by combining process steps, as might be done by the performance of more than one process on a single machine. Once operations are combined, it may then be possible to perform processes simultaneously. If processes cannot be combined, it may be possible to integrate several processes into a single machine or work cell. The integration could be in the form of several machines linked together with automated material handling devices. Thus, the cell will have the appearance of a single machine. Whichever method is used the result is the same: manufacturing process steps are minimized. Minimizing process steps can lead to large productivity gains by reducing input to the process and perhaps improving output rates from the minimized process.

Increase process flexibility

Improving process flexibility enables a machine or operation to process more product variety. The flexibility is achieved by minimizing or eliminating setup time typically required in changing a machine over to another product line. This is the essence of flexible automation systems. When a particular machine is able to process more product variety, the machine's utilization increases, manufacturing lead time decreases and work in process is reduced. This can result in a substantial increase in productivity because inputs (time, labor, …) are reduced, assuming of course that system output remains the same or increases. This strategy will also likely involve use of the all three programmable automation technologies.

Optimize material handling

Material handling is a non-value-added component of the material conversion process and thus should be optimized. Machines can often move material more consistently, accurately, and reliably than manual labor. Additionally, labor, freed up from performing material handling tasks can be displaced to perform value-added tasks. Thus, the increase in productivity may come from reduced labor costs, increased production rates, and reduction in scrap and rework. Optimizing material handling may involve a combination of mechanical technology systems, including conveyor systems, indexing units, and pick-n-place units, with robotic and PLC technologies.

Automate inspection

Product inspection determines if a product is within specifications. Often performed offline or outside of the process, inspection gives feedback about how a process is performing, and information gathered from inspection is typically used to adjust a process as needed. The feedback loop—make a part, inspect it, adjust process—can be rather long. Hence, products outside of specification ("off spec") could be completed before the process that made them is adjusted. Automatic inspection is an attempt to minimize the feedback loop and thereby reduce scrap and rework. And, of course, a more consistent, higher quality product is produced. To sum up, raw material usage is reduced and output is increased, resulting in substantial productivity improvements. Automatic inspection systems may utilize material handling technology, including robotics, electronic vision systems, electronic sensors, actuators, and PLC technology.

Implement process control

To produce a high quality product it is necessary to have a consistent, repeatable, and reliable process. To achieve this, a process must be rigorously controlled. Programmable logic controllers are capable of providing this level of control, providing it over event-driven changes to the process. Based on the status of these events, the PLC will make decisions and take appropriate action on the system. This enables fast, reliable control of the process and greatly improves its efficiency. In addition to greater efficiency, process output and product quality are also improved.

Note that a particular automation project may focus on only one strategy or include all five. Keep in mind that not all processes can or should be automated. Some processes may be too technologically difficult to economically automate. For others, the product life cycle is so short that automation cannot be justified. In some cases it may appear that the cost of the automation is completely justified based on the anticipated productivity improvements, but later it may be discovered that is simply not the case. Thus, it is imperative one have a sound method of justifying where and when to use automation. The next chapter deals with this subject in detail.

1.7 Summary

Programmable automation technology is the combination of mechanical, electrical, and computer technology developed to have very specific automation capabilities. Programmable automation consists of three individual technologies that are linked together by their capacity to be programmed. These technologies include computer numerical control (CNC) technology, robotics technology, and programmable logic control (PLC) technology. These technologies are in use in almost all modern manufacturing facilities and together make up the technological foundation of automation.

Manufacturing converts a raw material into a more useful product that can be sold for a profit. The manufacturing operations that are necessary to produce a particular product include manufacturing processes, material handling, quality control, and manufacturing support. Manufacturing processes are the manufacturing steps that perform the actual physical conversion. They can be classified as shaping processes, property-enhancing processes, and assembly processes. A particular manufacturing process will follow a systematic sequence of operations called a program of instructions. The way in which the manufacturing operations and workers are organized defines the manufacturing system used by the facility.

The manufacturing system in use in a facility is dictated by the finished product's product definition, which is determined by its complexity, variety, and quantity. Product complexity relates to the difficulty of its production process. Product variety refers to how many different product designs, versions, or models are to be produced within the facility. Product variety can be either hard or soft. Soft product variety indicates there are only

subtle differences between product models. Hard product variety indicates vastly different products are to be produced.

The four standard manufacturing systems include fixed-position, process, quantity, and flow-line. Fixed-position manufacturing systems are used to produce large complex products. The process manufacturing system is used when product complexity is relatively low and there is hard product variety. The quantity manufacturing system and the flow-line manufacturing system produce mass quantities of products and are thus called "mass production systems." The quantity manufacturing system is used with low product complexity and hard product variety. Flow-line manufacturing is for products with high product complexity and soft product variety.

Manufacturing support systems provide the management of the business operations of the facility and of the manufacturing system; they process information necessary to accomplish the conversion of the raw material into the finished product.

Automation is defined as the application of mechanical, electrical and/or computer technology to reduce the level of human participation in performing tasks. Fixed automation, programmable automation, and flexible automation are the three automation types. Fixed automation equipment typically consists of processing stations linked together with some form of material handling, which progressively moves the workpiece through the processing steps. Whereas fixed automation is "fixed" to a specific operation progression, programmable automation has the capability to alter both the type of operation to be performed and the order in which it is to be executed. Flexible automation possesses some of the features of both fixed and programmable automation with an added characteristic of no lost time for changeover between products. It is essentially a fixed automation machine that can process soft product variety with no setup.

CNC technology is one of the three distinct types of programmable automation that utilize a combination of mechanical, electrical, and computer technology to move a tool relative to a workpiece to perform some type of processing. Robotic technology is very similar to CNC technology in that it utilizes mechanical, electrical, and computer technology to move a manipulator in three-dimensional space. Whereas CNC and robotic technology provide motion control, PLC technology imparts automatic control over tasks and events through the use of electrical and computer technology. These three technologies are the foundation upon which modern automation is built.

The benefits of implementing automation come in many forms, each of which can almost certainly be expressed in terms of productivity improvement. The productivity of a manufacturing system is expressed as a ratio (%) of system output to system inputs. Productivity measure is perhaps the best indicator of when and where to use automation. Methods of improving productivity include increasing labor output, reducing labor input, reducing or eliminating labor shortages, reducing or eliminating routine manual and clerical tasks, improving worker safety, improving product quality, and reducing manufacturing lead time.

Some strategies to consider for implementing automation include minimizing manufacturing process steps, increasing process flexibility, optimizing material handling, automating inspection, and implementing process control.

1.8 Key Words

assembly line manufacturing system
automation
batches
computer numerical control (CNC) technology
continuous products
conversion process
discrete process control system
discrete products
fixed automation
fixed position manufacturing system
flexible automation
flow-line manufacturing system
hard product variety
job shop manufacturing system
lead time
lot size
manufacturing
manufacturing cell
manufacturing lead time
manufacturing operations
manufacturing processes

manufacturing setup
manufacturing support systems
manufacturing systems
process manufacturing system
product complexity
product definition
product quantity
product variety
product volume
production capacity
production rate
productivity
program of instructions
programmable automation
programmable logic control (PLC) technology
quantity manufacturing system
robotic technology
soft product variety
utilization

1.9 Review Questions

1. Define programmable automation.
2. What hampered programmable automation's initial use?
3. What member of the engineering staff is best suited to implement programmable automation?
4. Discuss how a manufacturing facility takes raw material and transforms it into a finished product.
5. Define the term "program of instructions."
6. List and discuss the four typical manufacturing operations.

7. List and explain the factors that determine a finished product's product definition.
8. Define and discuss the four standard manufacturing systems. Focus your explanation on the type of products produced by each system.
9. What is the difference between a manufacturing system and a manufacturing support system?
10. Define automation and the three major types.
11. What technologies fall under the programmable automation category? Include a definition of each.
12. Discuss the five performance measures of manufacturing.
13. What manufacturing performance measure combines and summarizes many of the individual measures into one all-encompassing metric?
14. Define productivity and provide three examples of how to improve it.
15. Discuss the five automation strategies defined in the chapter.

1.10 Bibliography

1. http://www.websters-online-dictionary.org/definition/automation
   Groover, M.P. (2001) *Automation, Production Systems and Computer-Integrated*
2. *Manufacturing, 2nd ed.,* Prentice Hall, Upper Saddle River, New Jersey
3. Sumanth, David J. (1994). *Productivity Engineering and Management*, McGraw-Hill
4. Kandray, Daniel E. (2004). Comparison of fixed automation and flexible automation from a productivity standpoint, Society of Manufacturing Engineers Technical Paper TP04PUB206
5. *Machinery's Handbook, 25th ed.* (1996). Industrial Press, Inc., New York, New York
6. http://www.websters-online-dictionary.org/definition/productivity
7. Radhakrishnan, Arun (2008). Intel Announces Two Billion Transistor Computer Chip, IT News Digest, February
8, http://blogs.techrepublic.com.com/tech-news/?p=2050

Chapter 2

Automation Justification and Productivity Concepts

Contents

Objective

The objective of this chapter is to demonstrate how to use productivity calculations to identify, evaluate, and justify automation.

2.1 Automation Justification and Productivity

Implementing programmable automation typically requires substantial investment, or *capital expenditure*, which upper management must deem justified. Most collegiate engineering and technology programs offer engineering economic analysis courses that present numerous methods of justifying capital expenditures, such as developing cash flows over the life of the project and considering the time value of money. However, the decision of whether or not to invest in an automation project is still very difficult for a firm to make because of the large number of variables to be considered. Many larger firms have arcane justification methodologies. A *productivity analysis*, on the other hand, is a simple, single metric that can clearly show when an automation project should be funded. It compares the performance of the system before and after the automation is applied. In fact, it provides such great scrutiny of a system that it should be used prior to any automation plans. In so doing, the automation strategies, defined in Chapter 1, can be accurately applied. This chapter focuses on how productivity calculations are used to identify, evaluate and justify automation.

2.2 Productivity Calculations

Recall that the productivity of a manufacturing system is determined by the simple ratio,

$$productivity = output/input,$$

where, as we will see, the input and output units are number of parts per monetary unit.

For manufacturers of discrete products a system's output is the number of parts produced over a certain time frame. The system inputs are those resources needed to acquire and convert raw material into a finished product over that same time frame. Typical resource input comprises labor, capital, material, and energy. Even though each of these inputs is vastly different, they can be expressed in monetary terms. Thus, productivity will be expressed in terms of the number of parts produced per dollar of input (# of parts/$ input). This is a simple and effective means of accessing a manufacturing operation, machine, process, system or facility's performance.

Obviously, the time frame over which a form of input is measured should be the same as the output that results. For products manufactured in high quantities, the time unit is usually hours; however, time units of days, weeks, or years can be used as well.

A *partial productivity* (P_P) calculation considers only one input (such as labor). It is defined as:

$$P_P = P_O/P_I,$$

where

P_O = number of parts output from a process in a specified time frame (# of parts/hr); this is often termed the *production rate* of the process

P_I = amount of money input into the process over the same time frame ($/hr).

A *combined productivity* (P_C) calculation considers two or more inputs. Accordingly, combined productivity is given by the following equation:

$$P_C = P_O /SP_I$$

where

SP_I = monetary sum of partial productivity measures input over a given time frame ($/hr).

Thus,

$$SP_I = P_{I\ labor} + P_{I\ cap} + P_{I\ mat} + P_{I\ energy}$$

The following examples demonstrate calculation of partial and combined productivity.

Example 2.1
A manufacturing process can produce 120 parts per hour. The process requires two laborers, each earning $18/hour. What is labor partial productivity of the process?

Solution
The governing equation is:

$$P_{P\ labor} = P_O /P_{I\ labor}$$

The output in parts per hour is

$$P_O = 120 \text{ parts/hr}$$

Since there are 2 laborers, the labor input is

$$P_{I\,labor} = 2 \text{ laborers} \times (\$18/hr)/\text{laborer} = \$36/hr.$$

Thus,

$$P_{P\,labor} = P_O/P_{I\,labor} = (120 \text{ parts/hr})/(\$36/hr) = 3.33 \text{ parts/\$.}$$

So here the process can produce 3.33 parts for every dollar of labor input.

Example 2.2

The manufacturing process described in Example 2.1 uses a machine with capital cost of \$45/hr. The machine requires 75 kW of power to operate. Cost of electricity is \$0.057/kW-hour (kWh). The machine processes 150 lb of material per hour. The material costs \$0.45/lb. Using the labor input costs found in Example 2.1, calculate the combined productivity of the process.

Solution

The governing equations are

$$P_C = P_O /SP_I$$

$$SP_I = P_{I\,labor} + P_{I\,cap} + P_{I\,mat} + P_{I\,energy}.$$

The output in terms of parts per hour was given as

$$P_O = 120 \text{ parts/hr.}$$

The labor partial productivity input was determined in Example 2.1 to be

$$P_{I\,labor} = \$36/hr.$$

The cost of capital to run the machine is given as

$$P_{I\,cap} = \$45/hr.$$

The cost of energy input per hour is found by converting power into energy. For 1 hour of operation the energy used will be:

$$\text{energy} = (\text{power})(\text{time}) = (75 \text{ kW})(1 \text{ hr}) = 75 \text{ kWh.}$$

Thus, the machine will use 75 kWh for every hour of use. Therefore, the cost of the energy input into the process will be

$$P_{I\,energy} = (\text{energy use/hr})(\text{electricity cost}) = (75\ \text{kWh/hr})(\$0.057/\text{kWh}) = \$4.28/\text{hr}.$$

The material cost input into the process is determined by multiplying the amount of material used per hour by the cost of the material:

$$P_{I\,mat} = (\text{material use per hr})(\text{material cost}) = (150\ \text{lb/hr})(\$0.45/\text{lb}) = \$67.50/\text{hr}.$$

Therefore,

$$SP_I = P_{I\,labor} + P_{I\,cap} + P_{I\,mat} + P_{I\,energy}$$

$$= \$36/\text{hr} + \$45/\text{hr} + \$67.50/\text{hr} + \$4.28/\text{hr} = \$152.78/\text{hr}.$$

Correspondingly,

$$P_C = P_O/SP_I = (120\ \text{parts/hr})/(\$152.75/\text{hr}) = 0.79\ \text{part/\$}.$$

When all the inputs to the system are considered, the process produces less than one part (0.79) for every dollar of input.

In the examples listed above, the process output in parts/hr was provided. Typically this information is determined through calculations. The following section provides mathematical concepts to quantify production and thereby provide a method of calculating output in parts/hr. Subsequent sections demonstrate how to develop input capital costs of automated machines.

2.3 Process Outputs and Mathematical Concepts for Quantifying Production

In order to determine if an automation strategy selected will provide the desired productivity improvements we must first *quantify* the current and proposed manufacturing process. Or, in other words, we must measure and document each process's performance. This serves as the output for the productivity calculations. The performance of the automation can then be quantified, and productivity calculations as well as direct comparison of the automation to the existing process can be made.

There is one measure, as observed in the last section, which is of prime importance. That measure is called the *production rate* (P_O) of the process. It is a measure of how many parts are produced over a specific time period, typically expressed in parts per hour.

This measures the output of the process. By combining this measure with other factors, several other mathematical quantifying concepts, in addition to productivity, can be examined.

2.3.1 Production Rate

Prior to determining production rate, one must determine the *operational cycle time* of a process. The operational cycle time includes all time element activities involved in producing one part.

Figure 2-0 Milling process example

Consider the machining process operation in Figure 2-0. This particular operation involves drilling two holes and milling a slot into the part shown. The operational cycle time is the time from the start of processing a part to the point at which the next part is started. The time elements for this operation include loading and unloading the part into the machine, machining the part, and changing tools as needed. Thus, the *operational cycle time* for this process can be given as

$$t_c = t_o + t_{wh} + t_{th}$$

where,

t_c = operational cycle time, expressed in min/part

t_o = time of actual processing, expressed in min/part

t_{wh} = workpiece handling time, expressed in min/part

t_{th} = tool handling time, expressed in min/part.

The *actual processing time* (t_o) and *workpiece handling time* (t_{wh}) occur for each part processed. The *tool handling time* (t_{th}), however, may not occur for each part. Perhaps a tool can process 100 parts before it needs to be changed. Thus, the time it takes to change the tool must be divided or averaged over those 100 parts.

The above equation is valid not only for machining type processes but also for assembly, molding, or almost any type of discrete manufacturing process. Consider the following examples.

Example 2.3

The following table lists the steps for a machining process. The times listed are those needed to load, unload, and process one part. Calculate the operational cycle time (t_c).

Process Step	Description	Time
1	Part loaded into machining fixture	0.75 min
2	First machining operation	2 min
3	part repositioned in fixture	0.5 min
4	second machining operation	3 min
5	part unloaded	0.25 min
Note: After 20 parts, the cutting tool is changed; 5 min time is required to change tools.		

Solution

The governing equation is

$$t_c = t_o + t_{wh} + t_{th}.$$

From the table above, combine steps 2 and 4 to determine the total processing time per part (t_o). Thus,

$$t_o = 2 \text{ min/part} + 3 \text{ min/part} = 5 \text{ min/part}.$$

The work handling time (t_{wh}) is a summation of results in steps 1, 3, and 5:

$$t_{wh} = 0.75 \text{ min/part} + 0.5 \text{ min/part} + 0.25 \text{ min/part} = 1.5 \text{ min/part}.$$

To determine the tool handling time (t_{th}), find the average time it takes to change tools over the 20 parts:

$$t_{th} = 5 \text{ min/20 parts} = 0.25 \text{ min/part}.$$

Thus the operational cycle time becomes

$$t_c = 5 \text{ min/part} + 1.5 \text{ min/part} + 0.25 \text{ min/part} = 6.75 \text{ min/part}.$$

Example 2.4

An injection molding machine processes an 8-cavity mold in 2.6 min per cycle. The parts are automatically ejected from the mold and travel by conveyor to the next process. Every 200 cycles the mold is cleaned and sprayed with mold release. This takes 15 min to complete. Calculate the operational cycle time (t_c).

Solution

The governing equation is

$$t_c = t_o + t_{wh} + t_{th}.$$

First, calculate the actual processing time per part (t_o). Recognize that the process produces 8 parts each cycle and that each cycle takes 2.6 min. Thus,

$$t_o = (2.6 \text{ min/cycle})(\text{cycle/8 parts}) = 0.325 \text{ min/part}.$$

Since the parts are automatically ejected from the mold, the workpiece handling time is zero:

$$t_{wh} = 0.$$

Mold cleaning and reapplication of mold release is equivalent to changing a tool in a machining process. Thus, the time it takes to accomplish these tasks needs to be averaged over each part produced in those 200 cycles:

$$t_{th} = 15 \text{ min/}[(200 \text{ cycles})(8 \text{ parts/cycle})] = 0.0094 \text{ min/part}.$$

Therefore, the operational cycle time becomes

$$t_c = 0.325 \text{ min/part} + 0 + 0.0094 \text{ min/part} = 0.334 \text{ min/part}.$$

Consider a process manufacturing system. In this system parts are produced in batches or lots. Each time a part is to be produced the machines that produce the part must be set up to process that particular part. This setup time needs to be captured in the production rate calculations. Therefore, for process manufacturing systems the time to process a batch of parts is calculated and then converted into average production time. The equation to calculate the *batch processing time* is

$$t_b = t_{su} + Qt_c,$$

where

t_b = batch processing time (min)
t_{su} = time to set up machine to produce batch (min)
t_c = operational cycle time per part (min/part)
Q = number of parts in batch (parts).

The *average production time* then becomes

$$t_p = t_b / Q,$$

where

t_p = average production time (min/part)

Q = number of parts in batch (parts).

Once the cycle time of the process is known, the production rate can be calculated. Note that the production rate depends on the manufacturing system employed. Hence, the variable R is used to represent the average production rate for the various processes discussed. The average production rate for a process can be determined by taking the reciprocal of the average production time:

$$R_p = 1/t_p,$$

where

R_p = average production rate (parts/min)

t_p = average production time (min/part).

Note that it is often more desirable to express the average production rate (R_p) in units of parts/hr.

Example 2.5

Calculate the production rate (R_p) of the process listed in Example 2.3 in units of parts/hr, assuming the part is produced in batches of 3000 parts and it takes 4 hr to set up the machine to produce the batch.

Solution

First, determine the batch production time (t_b) and then the average production time (t_p). From the average production time, calculate production rate (R_p) in parts/hr. The governing equations are

$$t_b = t_{su} + Qt_c$$

$$t_p = t_b/Q$$

$$R_p = 1/t_p,$$

where the values are

$\quad t_c = 6.75$ min/part (calculated in Example 2.3)

$\quad t_{su} = 4$ hr

$\quad Q = 3000$ parts.

It is important to keep consistent units: convert operational cycle time to units of hr/part. Thus,

$$t_c = (6.75 \text{ min/part})(1 \text{ hr/60 min}) = 0.1125 \text{ hr/part}.$$

The batch production time is then

$$t_b = 4 \text{ hr} + (3000 \text{ parts})(0.1125 \text{ hr/part}) = 4 \text{ hr} + 337.5 \text{ hr} = 341.5 \text{ hr}.$$

The average production time is

$$t_p = 341.5 \text{ hr}/3000 \text{ parts} = 0.1138 \text{ hr/part}.$$

Therefore the production rate is

$$R_p = 1/t_p = 1/0.1138 \text{ hr/part} = 8.78 \text{ parts/hr}.$$

The last example highlights how to determine production rate of any type of manufacturing process in which parts are run in batches and setup time is a significant portion of batch production time. As setup time decreases and quantities processed increases, operational cycle time approaches the same value as average production time. This is the case in quantity type manufacturing systems. Thus, the production rate can be determined directly from the operational cycle time:

$$t_p = t_b/Q = (t_{su} + Qt_c)/Q.$$

Since setup time becomes small relative to the product of quantity and operational cycle time t_c, then clearly

$$t_p \sim t_c.$$

Then

$$R_{pq} = 1/t_c,$$

where

R_{pq} = average production rate for a quantity manufacturing systems (parts/min)

t_c = operational cycle time (min/part).

Example 2.6
Calculate the average production rate (R_{pq}) of the injection molding process in Example 2.4, in units of parts/hr.

Solution
The injection molding process is a quantity type manufacturing process. Therefore, the governing equation is

$$R_{pq} = 1/t_c.$$

Taking the operational cycle time from Example 2.3 and converting the units to hr/part yields

$$t_c = (0.334 \text{ min/part})(1 \text{ hr/60 min}) = 0.00567 \text{ hr/part}.$$

Therefore

$$R_{pq} = 1/(0.00567 \text{ hr/part}) = 179.64 \text{ parts/hr}.$$

Consider the flow-line type manufacturing system shown in Figure 2.1. In it the product is traveling to each workstation via a conveyor belt. At each workstation the product is processed accordingly. Upon completion of the processing, the product is moved to the next station. The transporting of the part is coordinated with the time it takes to complete the processing at each workstation. Some workstations finish processing sooner than others. However, the conveyor cannot move the parts until the slowest process (i.e., the process that takes the most time) is completed. This workstation is called the *bottleneck station*.

Figure 2-1 Flow-line manufacturing system

When all the stations have processed the product, it exits the conveyor belt. Thus, at specific time intervals a finished product is produced. The time interval may be expressed in minutes, hours, days, weeks, months, or even years. The production rate has been defined as the number of parts produced per hour, which corresponds to number of parts that drop off the conveyor line in an hour in this example. Therefore, to calculate the production rate it is necessary one determine how often a part falls from the conveyor. This operational cycle time of the flow line (t_c) is the sum of the time to move the product between the workstations and the actual processing time at the bottleneck station. In equation form:

$$t_{cf} = t_r + \max t_o$$

where

t_{cf} = operational cycle time of flow line system (min/part)

t_r = time to transfer parts between stations (min/part)

max t_o = actual processing time of bottleneck workstation (parts/min).

The *production rate* or *cycle rate* of the flow line then becomes

$$R_c = 1/t_{cf},$$

where R_c = cycle rate of a flow-line manufacturing system (parts/min).

Example 2.7

Calculate the cycle rate (R_c) of the flow-line manufacturing system shown in Figure 2-1; assume the transfer rate is 3 sec per part and the processing time for each work station is as shown in the table.

Workstation	Processing time (min/pc)
1	1.5
2	0.75
3	1.25
4	1.5
5	0.5

Solution

The governing equations are

$$t_{cf} = t_r + \max t_o$$

$$R_c = 1/t_{cf}.$$

The transfer rate was given as

$$t_r = 3 \text{ sec/part} \quad \text{or} \quad 0.05 \text{ min/part}.$$

The actual process time for each workstation is given in the table. Workstation 4 has the maximum process time of 1.5 min/part. Thus,

$$\max t_o = 1.5 \text{ min/part.}$$

The operational cycle time of the flow-line is then

$$t_{cf} = 0.05 \text{ min/part} + 1.5 \text{ min/part} = 1.55 \text{ min/part.}$$

The cycle rate is then:

$$R_c = 1/1.55 \text{ min/part} = 0.645 \text{ part/min} \quad \text{or} \quad 38.7 \text{ parts/hr.}$$

2.3.2 Other Mathematical Quantifying Concepts

Although productivity is of primary importance in justifying automation, other quantifying concepts come into play as well. These include *production capacity*, *utilization, availability*, and *manufacturing lead-time.*

Production capacity is the maximum rate of output of a particular product for a manufacturing system over a specified time period. The time can be expressed in days, weeks, months, or years. The system under consideration could be the whole plant, a production line, or a manufacturing cell. The calculation takes the production rate of the system under consideration and multiplies it by the number of hours worked during the specified time interval and the number of subsystems producing at that production rate. The general form of the equation is:

$$P_c = R n_m hrs_w,$$

where

R = production rate of system (R_p, R_{pq}, or R_c) in parts/hr

n_m = number of machines or work centers producing at that rate

hrs_w = hours worked during the specified time interval.

Example 2.8

Calculate the monthly production capacity (P_c) of a product produced by the injection molding process described in Example 2.4. Assume the plant uses 3 injection molding machines and molds to produce the part. Also, assume the plant operates in three

8-hour shifts per day, 5 days per week. Suggest a plan by which the plant may increase production capacity in the short term. How could it do so in the long term?

Solution

The governing equation is

$$P_c = Rn_m hrs_w,$$

where

R_{pq} = 179.64 parts/hr (calculated in Example 2.6)

$n_m = 3$

hrs_w = (3 shifts)(8 hr/shift)(5 days/week)(4 weeks/month) = 480 hr/month.

Thus,

$$P_c = (179.64 \text{ parts/hr})(3)(480 \text{ hr/month}) = 258,681 \text{ parts/month}.$$

To increase production in the short term the plant could run the injection presses on weekends. Doing so would increase hours worked (hrs_w). Long term solutions might include building more molds to run on more injection molding machines (increase n_m) and/or increase the production rate (R). This could be accomplished by building molds with more cavities or decreasing the process's operational cycle time (t_c).

Production capacity is a theoretical value. In practice, actual production may be significantly less due to lack of orders, lack of supplies, processing problems, or labor issues. Thus, management will often evaluate the utilization of a manufacturing system. *Utilization U* is defined as the ratio of the actual number of products divided by production capacity. Thus,

$$U = 100Q /P_c,$$

where

Q = actual production over specified time frame

P_c = production capacity over specified time frame.

Note that U is expressed as a percentage.

Example 2.9

Calculate the utilization of the injection molding process described in Example 2.8 if actual production in the previous month was 175,000 parts.

Solution

The governing equation is

$$U = 100Q/P_c,$$

where

$Q = 175,000$ parts
$P_c = 258,681$ parts/month.

Thus,

$$U = (100)(175,000 \text{ parts})/258,681 \text{ parts} = 67.7\%.$$

Additionally, a manufacturing system under repair may not be fully used. Thus, the availability of a system, expressed as a percentage, can be calculated. It is determined by the equation

$$A = 100(t_{mtbf} - t_{mtbr})/t_{mtbf}$$

where

A = availability
t_{mtbf} = mean time between failures (hr)
t_{mtbr} = mean time to repair (hr).

These two measures provide solid insight into a manufacturing system and can also help in identifying automation opportunities. Additionally, if utilization and availability information is known within a facility, realistic actual production values can be calculated. Consider the following example.

Example 2.10

A manufacturing system has a theoretical production capacity of 100,000 parts/ month. Typical utilization of the system is 80% and availability is 93%. What is the anticipated actual monthly production of the system?

Solution

Rearranging the equation for utilization and factoring in the availability of the system yields the following equation:

$$Q = UP_cA.$$

Thus,

$$Q = (80\%)(100{,}000 \text{ parts/month})(93\%) = 74{,}400 \text{ parts/month.}$$

Another important quantifying measure of production is manufacturing lead-time. Manufacturing lead-time is the total time it takes to convert raw material into a finished product. Thus, it is the summation of the time of each individual manufacturing process that the product passes through. Note, however, that a product is not processed continually. There is also non-operation time associated with each operation. Examples of non-operation times include those for moving and queuing of parts between operations, waiting for materials, waiting for tools, and so on. These must be accounted for in the calculation of manufacturing lead-time. Additionally, the time to set up the process, where appropriate, must also be considered. Accordingly, the equation for the manufacturing lead-time of a process manufacturing system consisting of operations (indexed by i) is

$$t_{\text{mlt}} = \text{sum}_i(t_{\text{su}} + Qt_c + t_{\text{nop}})_i,$$

where

t_{mlt} = manufacturing lead-time for batch

t_{su} = setup time for a process

Q = number of parts in batch

t_c = operational cycle time of a process

t_{nop} = non-operation time of a process

Note that this equation can be used for other types of manufacturing systems as well. However, some of the terms may be insignificant. Consider a quantity manufacturing system. The setup time and non-operation time may become very small compared to the batch size. Additionally, in the flow-line system the setup and non-operation time are essentially nonexistent.

Example 2.11

A part is routed through 4 machines in lot sizes of 500 parts/batch. Average non-operation time is 6 hr. Setup and operational cycle times are shown in the table below. Calculate the manufacturing lead-time for the part.

Machine	Setup time (hrs)	Operation time (min)
1	1	3
2	6	8
3	1.5	4
4	4	3

Solution

The governing equation is

$$t_{mlt} = sum_i(t_{su} + Qt_c + t_{nop})_i.$$

Calculate the manufacturing lead-time for each operation:

t_{mlt1} = 1 hr/batch + (500 parts/batch)(3 min/part)(1 hr/60 min) + 6 hr/batch = 32 hr/batch
t_{mlt2} = 6 hr/batch + (500 parts/batch)(8 min/part)(1 hr/60 min + 6 hr/batch =78.67 hr/batch
t_{mlt3}= 1.5 hr/batch + (500 parts)(4 min/part)(1 hr/60 min + 6 hr/batch = 40.83 hr/batch
t_{mlt4} = (4 hr/batch + 500 parts)(3 min/part)(1 hr/60 min) + 6 hr/batch = 35 hr/batch

Summing operation lead-times gives

$t_{mlt}= t_{mlt1} + t_{mlt2} + t_{mlt3}+ t_{mlt4} = 32 + 78.67 + 40.83 + 35 = 186.5$ hr/batch.

It is relatively easy to visualize how improvements in productivity result in corresponding improvements in these measures. Therefore, such measures can also be used in making the case for automation.

2.4 Process Inputs and Manufacturing Costs

The previous section demonstrated how one would quantify the output of a process with a measure (production rate) that can be used in productivity calculations. In this section, methods of quantifying the input to the process are developed. As shown in Section 2.2, input into the productivity calculation (P_I) is amount of money required for the process step under consideration, which is input into the process over the same time frame as that of the output measurement. Both measurements are in units of $/hr.

Inputs to a process are typically broken down into categories consisting of capital, energy, labor, and material. These categories are termed *partial productivity measures*. Consideration of a breakdown of the costs to manufacture a product is shown in Figure 2-2.

Figure 2-2 Manufacturing process expenses

Figure 2-2 is a pie chart showing the relative percentage of expenses that make up the final selling price of a representative product. Note that the manufacturing cost is only 40%. Figure 2-3 is a pie chart of the relative percentage of expenses that make up total manufacturing cost for this product. Notice how the categories in Figure 2-3 relate to the partial productivity measures. Direct labor coincides with labor, capital equipment costs with capital; indirect labor is often absorbed into the capital equipment or direct labor costs; materials and supplies category would represent both material and energy. When an automation project is undertaken, its goal is to decrease one or more of the expenses shown in the figure. Thus, these expenses need to be accurately reflected in the productivity calculations. One accomplishes this by expressing the partial productivity measures in $/hr.

Figure 2-3 Manufacturing cost percentages

When one evaluates an existing process, labor rate, energy cost, and raw material costs are typically readily available from the manufacturing firm's accounting office. Additionally, the capital costs of the existing equipment would be available as well. These rates will include allocated overhead costs. However, capital costs of an alternative—new automated equipment—process must be estimated.

Estimation of capital costs of a proposed automation can be done through simple calculations that take into account the time value of money in conjunction with allocated factory overhead. When a manufacturing firm invests in a capital expenditure, it expects the investment will yield a return. Most firms have a standard rate of return. This figure, expressed as a percentage, should be readily available from a firm's upper management. With the rate in hand, the automation engineer can begin to make estimated capital cost calculations.

The goal of the capital cost calculations is to represent the cost of the proposed automation in terms that can be used in the productivity calculations. Thus, cost needs to be expressed in terms of $/hr. The calculation breaks the initial cost of the equipment into an annual cost, then spreads that annual cost over the hours the machine is estimated to run in a year; finally, it adds in factory overhead expenses. The estimated hourly capital cost of the automation can be calculated with the following equation:

$$C_c = C_a(1 + r_{foh}),$$

where

C_c = estimated hourly capital cost of the automation ($/hr)

C_a = estimated hourly cost of automation ($/hr)

r_{foh} = factory overhead rate.

The hourly estimated cost of the automation can be determined from the equation

$$C_a = (C_I f_{cr})/h_a,$$

where

C_I = initial cost of the automation ($)

f_{cr} = capital recovery factor

h_a = time that machine is in operation annually (hr).

The initial cost of the automation (C_I) will be known and hours of machine operation annually (h_a) can be readily determined. The capital recovery factor (f_{cr}) is determined by the equation

$$f_{cr} = r(1 + r)^n / [(1 + r)^n - 1],$$

where

r = desired rate of return (%)

n = number of years of the service life of the machine.

Figure 2-4 Typical factory overhead costs

Factory overhead rate (r_{foh}) is the ratio of factory overhead costs to those of the machine under consideration. This is found by distributing the overhead over some variable such as direct labor costs. Typical factory overhead costs are given in Figure 2-4. For any given factory this can be accomplished by taking all of the overhead costs of the

firm for one year and dividing it by the total cost spent on direct labor. The formula for this calculation is:

$$r_{foh} = C_{foh}/C_{dl},$$

where

C_{foh} = annual cost of factory overhead ($/yr)

C_{dl} = annual direct labor costs ($/yr).

The following example demonstrates the use of these formulas.

Example 2.12

An automated work cell is being considered to replace an existing process. The cell will cost $150,000 to purchase and is anticipated to have a 4-year service life. The machine will operate for 2080 hours per year. The company spent $2,300,000 on factory overhead and $6,500,000 on direct labor costs last year. Estimate the hourly capital cost to operate the new automated work cell if the manufacturing firm desires a 15% return on its investment.

Solution

The governing equations are

$$C_c = C_a(1 + r_{foh})$$
$$C_a = (C_I f_{cr})/h_a$$
$$f_{cr} = r(1 + r)^n/[(1 + r)^n - 1]$$
$$r_{foh} = C_{foh}/C_{dl}.$$

The values are given as

r_{foh} = 35%

C_I = $150,000

h_a = 2080 hr

r = 15%

n = 4 yr

C_{foh} = $2,300,000/yr

C_{dl} = $6,500,000/yr.

First, calculate capital recovery factor:

$$f_{cr} = r(1 + r)^n/[(1 + r)^n - 1]$$

$$= 0.15(1 + 0.15)^4/[(1 + 0.15)^4 - 1)$$
$$= (0.15)(1.749)/(1.749 - 1) = 0.26235/0.749 = 0.3503.$$

Next, calculate the estimated hourly rate of the work cell (C_a):

$$C_a = C_I f_{cr}/h_a = (\$150,000)(0.3503)/2080 \text{ hr} = \$25.26/\text{hr}.$$

Calculate factory overhead rate:

$$r_{foh} = (\$2,300,000/\text{yr})/(\$6,500,000/\text{yr}) = 35.4\%.$$

Finally, calculate the hourly capital cost:

$$C_c = C_a(1 + r_{foh}) = (\$25.26/\text{hr})(1 + 0.354) = \$34.20/\text{hr}.$$

Thus, the estimated capital cost for operating the new automated work cell is $34.20/hr.

2.5 Comparing Alternatives with Productivity Calculations

Section 2.2 demonstrated the basic procedure for calculating the productivity of a process. Sections 2.3 and 2.4 demonstrated how to quantify the output and inputs of a process for use in the productivity calculations. This section is devoted to developing a methodology of performing the actual comparison of the alternatives. This essentially involves organizing the data in a logical, comprehensive manner in which the alternatives can be directly compared. The author has found that this is easily accomplished by organizing the data in a spreadsheet as shown in Figure 2-5.

Combined Productivity Comparison					
Description	**Variable**	**Units**	**Current Method**	**Proposed Method**	**Formula or Comments**
Production rate	P_O	pcs/hr			R_p or R_{pq} or R_c
Labor Partial Productivity					
Labor cost/hr	$P_{I\,labor}$	\$/hr			Given
Labor productivity	$P_{P\,labor}$	pcs/\$			$P_O / P_{I\,labor}$
Labor productivity index	I_{labor}	-	1.0		$(P_{p\,labor})$ proposed $/(P_{p\,labor})$ current
Capital Partial Productivity					
Capital cost/hr	$P_{I\,capital}$	\$/hr			Given or calculated as C_c
Capital productivity	$P_{P\,capital}$	pcs/\$			$P_O / P_{I\,capital}$
Capital productivity index	$I_{capital}$	-	1.0		$(P_{p\,capital})$ proposed $/(P_{p\,capital})$ current
Raw Material Partial Productivity					
Raw material cost/hr	$P_{I\,material}$	\$/hr			Given or calculated
Raw material productivity	$P_{P\,material}$	parts/\$			$P_O / P_{I\,material}$
Raw material productivity index	$I_{material}$	-	1.0		$(P_{p\,material})$ proposed $/(P_{p\,material})$ current
Energy Partial Productivity					
Energy cost/hr	$P_{I\,energy}$	\$/hr			Given or calculated
Energy productivity	$P_{P\,energy}$	pcs/\$			$P_O / P_{I\,energy}$
Energy productivity index	I_{energy}	-	1.0		$(P_{p\,energy})$ proposed $/(P_{p\,energy})$ current
Combined Productivity					
Sum of partial productivity inputs	SP_I	\$/hr			$P_{I\,labor} + P_{I\,capital} + P_{I\,material} + P_{I\,energy}$
Combined productivity	P_c	pcs/\$			P_O / SP_I
Combined productivity index	I_c	-	1.0		(P_c) proposed $/(P_c)$ current

Figure 2-5 Combined productivity comparison spreadsheet

The spreadsheet, aptly titled "Combined Productivity Comparison," organizes the productivity data in rows and columns. The column headings are shown at the top in boldface print. The first column lists the description of the measure, the second is the variable used for the measure, and the third column displays the units. The next two columns, "Current Method" and "Proposed Method," hold the data and calculation results for each method. The close proximity of these two columns enables swift comparison of the two options. The sixth column is reserved for formulas or comments (where certain cells hold formulas for performing the calculations). In the first row is entered the production rate for each method. This rate is determined through calculations dependent on the type of manufacturing system, as was discussed in Section 2.3.1. The next 12 rows are separated into groups corresponding to the partial productivity measures discussed in Section 2.2. The organization of the rows culminates with several combined productivity measures, the group at the bottom of the spreadsheet.

Note that each of the five productivity measure groups contains a new measure, not previously discussed: *productivity index*. The productivity index is a clear and concise method for comparing partial and combined productivity measures of the two options. Observe that for the so-called current method, each productivity index row contains a value of 1.0. This is because current method is used as a baseline against which the proposed method will be compared. The productivity index for the new method is determined by dividing the proposed method's productivity (partial or combined) by the current method's productivity. For example, the formula for the combined productivity index (I_c) is given by the equation

$$I_c = (P_C) \ proposed/(P_C) \ current$$

where

I_c = combined productivity index

P_C) *proposed* = combined productivity of the proposed method (parts/\$)

(P_C) *current* = combined productivity of the current method (parts/\$).

Thus, if the proposed method has a productivity index greater than 1, it can be said that it is more productive than the current method. Conversely, a productivity measure of less than 1 indicates the proposed method is less productive than the current measure. Recall that for showing productivity improvement a combined productivity index greater than 1.0 is the key to justifying an investment in automation.

One should always look at combined productivity when comparing two methods. Consideration of only a partial productivity measure can often result in misleading results. However, if partial productivity measures are the same for the two methods being compared, they can be omitted from the calculations. But, odds are that there will always be more than one partial productivity measure to consider.

The following examples demonstrate the use of the spreadsheet.

Example 2.13

A manufacturing firm uses a manual machine for production. Production rate is 100 parts/hr. This current method utilizes two operators at a labor wage rate of \$18/hr. The manual machine's capital cost (including cost of electricity) of operation is \$25/hr. This firm is considering replacing the manual machine with a programmable automation work cell. The new cell requires only one operator, but has a capital cost (including cost of electricity) of \$65/hr. The production rate of the machine is 125 parts/hour. Perform a combined productivity analysis to determine if the firm should purchase the automated work cell.

Solution

The governing equations are listed in the spreadsheet shown in Figure 2-5. Entering production rate and calculating the labor partial productivity yields:

Combined Productivity Comparison					
Description	Variable	Units	Current Method	Proposed Method	Formula or Comments
Production rate	P_O	pcs/hr	100	125	Given
Labor Partial Productivity					
Labor cost/hr	$P_{I\,labor}$	$/hr	$36.00	$18.00	Given
Labor productivity	$P_{P\,labor}$	pcs/$	2.78	6.94	$P_O / P_{I\,labor}$
Labor productivity index	I_{labor}	-	1.0	2.50	Considering only labor, the proposed method is 250% as productive

Eliminating the operator and increasing the production rate results in the proposed method that is 250% as productive as the current method from a labor perspective. Considering this measure alone, the proposed method looks very attractive. However, as mentioned, we must evaluate all of the partial productivities and then calculate the combined productivity prior to passing final judgment. No information on raw material was given, thus it will be omitted from the calculations. Additionally, the cost of energy was given in the capital cost per hour. Thus, the only remaining partial productivity to evaluate is capital.

The increased capital hourly cost of the proposed method in conjunction with only a marginal increase in production rate makes the proposed method is only 48% as productive as the current method, from a capital perspective. Calculating the combined productivity yields:

Capital Partial Productivity					
Capital cost/hr	$P_{\text{I capital}}$	$/hr	$25.00	$65.00	Given
Capital productivity	$P_{\text{P capital}}$	pcs/$	4.00	1.92	P_O / $P_{\text{I capital}}$
Capital productivity index	I_{capital}	-	1.0	0.48	Considering only capital costs of the equipment, the proposed method is 48% as productive

Combined Productivity					
Sum of partial productivity inputs	SP_I	$/hr	$61.00	$83.00	$P_{\text{I labor}}$ + $P_{\text{I capital}}$
Combined productivity	P_c	pcs/$	1.64	1.51	P_O / SP_I
Combined productivity index	I_c	-	1.0	0.92	Combined productivity indicates the proposed method is 92% as productive!

Thus, the proposed method is 92% as productive as the current method. Thus, the proposed method is not justified and the firm should not purchase the automated work cell. The completed spreadsheet is shown in Figure 2-6.

Combined Productivity Comparison					
Description	**Variable**	**Units**	**Current Method**	**Proposed Method**	**Formula or Comments**
Production rate	P_O	pcs/hr	100	125	Given
Labor Partial Productivity					
Labor cost/hr	$P_{I\,labor}$	$/hr	$36.00	$18.00	Given
Labor productivity	$P_{P\,labor}$	pcs/$	2.78	6.94	$P_O / P_{I\,labor}$
Labor productivity index	I_{labor}	-	1.0	2.50	Considering only labor, the proposed method is 250% as productive
Capital Partial Productivity					
Capital cost/hr	$P_{I\,capital}$	$/hr	$25.00	$65.00	Given
Capital productivity	$P_{P\,capital}$	pcs/$	4.00	1.92	$P_O / P_{I\,capital}$
Capital productivity index	$I_{capital}$	-	1.0	0.48	Considering only capital costs of the equipment, the proposed method is 48% as productive
Combined Productivity					
Sum of partial productivity inputs	SP_I	$/hr	$61.00	$83.00	$P_{I\,labor} + P_{I\,capital}$
Combined productivity	P_c	pcs/$	1.64	1.51	P_O / SP_I
Combined productivity index	I_c	-	1.0	0.92	Combined productivity indicates the proposed method is 92% as productive!

Figure 2.6 Example 2.13, combined productivity calculation

This is the solution.

The last example demonstrates how to use the combined productivity comparison spreadsheet and highlights the importance of calculating the combined productivity before passing judgment on the proposed method. Another interesting benefit of the combined productivity comparison spreadsheet is that it can be a starting point or roadmap for identifying the type and quantity of improvements necessary to justify automation.

For example, one might ask, "If the proposed method is not justified (viz. Figure 2-6), what improvements would make it justifiable?" Obviously, if the work cell's production rate would be increased substantially and/or capital cost decreased, the purchase of the work cell might be justified. Thus, by tweaking the values in the spreadsheet, target values for production rate and capital costs can be identified. These targets can then be presented to the suppliers of the work cell as required performance specifications. Consider the following example.

Example 2.14

Based on the results of Example 2.13, determine the following:
a) Minimum production rate of the proposed method to yield a 20% productivity improvement. Assume all other values are as before.
b) Maximum capital cost per hour of the proposed method to yield a 20% productivity improvement. Assume all other values are as before.

Solution

Both of these can be determined by two methods. The first is to solve directly by using algebra and rearranging the governing equations accordingly. The other method is a trial and error method that uses the spreadsheet to manually increment the variable in question until the desired result is achieved. For part (a) the result will be solved directly. Trial and error will be used to solve part (b).

Part (a)

The governing equations are

$$I_c = P_C \text{ proposed}/P_C \text{ current}$$

$$P_C = P_O/SP_I$$

$$SP_I = P_{I \text{ labor}} + P_{I \text{ cap}} + P_{I \text{ mat}} + P_{I \text{ energy}}.$$

Note that none of the values in the spreadsheet for the current method changes. Also, all the partial productivity inputs for the proposed method stay the same. Therefore, the following values are given:

$$I_c = 1.20$$

$$P_C \; current = 1.64 \; parts/\$$$

$$SP_I = \$83/hr.$$

Setting up the equations:

$$1.20 = P_C \; proposed/1.64 \; parts/\$.$$

Rearranging yields

$$P_C \; proposed = (1.20)(1.64 \; parts/\$) = 1.968 \; parts/\$.$$

But $P_C \; proposed$ is determined from the equation

$$P_C \; proposed = (P_O/SP_I) \; proposed.$$

Dropping "*proposed*" and entering the correct values gives

$$1.968 \; parts/\$ = P_O / \$83/hr.$$

Thus, the minimum production rate is

$$P_O = (1.968 \; parts/\$)(\$83/hr) = 163.34 \; parts/hr.$$

The result is confirmed in the spreadsheet shown in Figure 2-7.

Combined Productivity Comparison					
Description	Variable	Units	Current Method	Proposed Method	Formula or Comments
Production rate	P_O	pcs/hr	100	163	Given
Labor Partial Productivity					
Labor cost/hr	$P_{I\,labor}$	$/hr	$36.00	$18.00	Given
Labor productivity	$P_{P\,labor}$	pcs/$	2.78	9.07	$P_O / P_{I\,labor}$
Labor productivity index	I_{labor}	-	1.0	3.27	
Capital Partial Productivity					
Capital cost/hr	$P_{I\,capital}$	$/hr	$25.00	$65.00	Given
Capital productivity	$P_{P\,capital}$	pcs/$	4.00	2.51	$P_O / P_{I\,capital}$
Capital productivity index	$I_{capital}$	-	1.0	0.63	
Combined Productivity					
Sum of partial productivity inputs	SP_I	$/hr	$61.00	$83.00	$P_{I\,labor} + P_{I\,capital}$
Combined productivity	P_c	pcs/$	1.64	1.97	P_O / SP_I
Combined productivity index	I_c	-	1.0	1.20	

Figure 2-7 Example 2.14, part (a) solution

Part (b)

For this solution, trial and error will be used with the spreadsheet. Start by decrementing capital cost/hr in $10/hr increments. As productivity approaches desired value, decrease the increment until the final number is arrived at, which is approximately $45.50/hr. The result is shown in Figure 2-8.

Combined Productivity Comparison					
Description	Variable	Units	Current Method	Proposed Method	Formula or Comments
Production rate	P_O	pcs/hr	100	125	Given
Labor Partial Productivity					
Labor cost/hr	$P_{I\,labor}$	$/hr	$36.00	$18.00	Given
Labor productivity	$P_{P\,labor}$	pcs/$	2.78	6.94	$P_O / P_{I\,labor}$
Labor productivity index	I_{labor}	-	1.0	2.50	
Capital Partial Productivity					
Capital cost/hr	$P_{I\,capital}$	$/hr	$25.00	$45.50	Determined through trial and error
Capital productivity	$P_{P\,capital}$	pcs/$	4.00	2.75	$P_O / P_{I\,capital}$
Capital productivity index	$I_{capital}$	-	1.0	0.69	
Combined Productivity					
Sum of partial productivity inputs	SP_I	$/hr	$61.00	$63.50	$P_{I\,labor} + P_{I\,capital}$
Combined productivity	P_c	pcs/$	1.64	1.97	P_O / SP_I
Combined productivity index	I_c	-	1.0	1.20	

Figure 2-8 Example 2-14, part (b) solution

This is the solution.

The previous example demonstrates how the spreadsheet variables can be tweaked to identify how the proposed method could be enhanced to make it a more attractive option from a productivity standpoint. Other variables could also be adjusted including reducing or eliminating the operator altogether and looking for material savings with the proposed method. Thus, the spreadsheet can be used as a roadmap to identifying other improvements the proposed method may have to offer.

When a combined productivity analysis indicates that the proposed method is justified, there is still one measure we should consider. That measure is *production volume*. Its impact on choosing alternatives is discussed in the next section.

2.6 The Impact of Production Volume on Alternatives

Thus far we have assumed that product volume—both current and future—of the process under consideration for automation is sufficient to support an automation investment. In general, when product volumes are low, manual methods are more cost effective. A manufacturing firm does not want to invest significant funds in the manufacture of a product that will no longer be produced in 6 months to year, or for which the annual volume is not great enough. Although predicting future volume, called *forecasting*, is a risky venture, it is an essential part of doing business. Product forecasts are typically available from marketing or upper management. The risky nature of production volume forecasting is one reason a manufacturing firm needs to see a quick payback of the automation investment. Thus, one must ascertain whether there is sufficient production volume to justify the investment in the automation. This can be accomplished by considering current and proposed methods' fixed and variable manufacturing costs and performing a production volume breakeven point analysis.

A product's manufacturing cost can be broken into two categories, *fixed costs* and *variable costs*. Fixed costs are costs that are independent of the quantity of product; i.e., they are incurred whether one part or one million parts are produced. These typically include building rent or mortgage costs, property taxes, equipment costs, and equipment maintenance, to name a few. Fixed costs are most conveniently expressed on an annual basis.

Variable costs, on the other hand, are dependent on the quantity of product. The higher the production, the more the cost incurred. Variable costs include direct labor, raw material costs, and energy costs to operate the equipment. Utilizing these concepts the total annual cost of a product can be represented by the following equation:

$$C_T = C_F + QC_V,$$

where

C_T = total cost incurred on an annual basis ($/yr)

$$C_F = \text{fixed cost of product on an annual basis (\$/yr)}$$

$$Q = \text{quantity of parts produced per year (parts/yr)}$$

$$C_V = \text{variable cost per part (\$/part)}.$$

This formula's use is demonstrated in the following example.

Example 2.15

Referring to the manual manufacturing method and information in Example 2.13, and given that the annual cost of maintenance for the machine is $8000 and the raw material cost is $1.25 per part, calculate total annual cost of producing 100,000 parts per year

Solution

The given information from the problem statement and taken from Example 2.13 is as follows:

$$P_O = 100 \text{ parts/hr (production rate)}$$

$$P_{I\,labor} = \$36/\text{hr (2 operators at \$18/hr)}$$

$$P_{I\,capital} = \$25/\text{hr}$$

$$Q = 100,000 \text{ parts/yr}$$

$$C_{F\,maint} = \$8000$$

$$\text{raw material cost} = \$1.25/\text{part}.$$

The first step is to identify the variable costs, which include the labor wage rate, the machine's capital cost, and the raw material cost. The labor and machine capital costs must be converted to units of $/part. This is accomplished by dividing the hourly cost by the production rate as follows:

$$C_{V\,labor} = P_{I\,labor}/P_O = (\$36/\text{hr})/(100 \text{ parts/hr}) = \$0.36/\text{part}$$

$$C_{V\,capital} = P_{I\,capital}/P_O = (\$25/\text{hr})/(100 \text{ parts/hr}) = \$0.25/\text{part}.$$

Next determine the total variable costs by summing each of the individual variable costs:

$$C_V = C_{V\,labor} + C_{V\,capital} + C_{V\,material}$$
$$= \$0.36/\text{part} + \$0.25/\text{part} + \$1.25/\text{part} = \$1.86/\text{part}.$$

The fixed cost is simply the annual maintenance costs:

$$C_F = \$8000/\text{yr}.$$

Solving for the total annual cost to make the product yields

$$C_T \; = \; C_F + QC_V = \; \$8000/\text{yr} + (100{,}000)(\$1.86/\text{part}) = \; \$194{,}000/\text{yr}.$$

Performing a quantity breakeven analysis of two alternatives involves determining the number of parts that have to be produced that would realize the benefits of the alternative. This is significant because manual methods typically have a lower fixed cost and higher variable costs. Thus, when product volumes are low, manual methods are more cost effective. As production volumes increase the advantage goes to automated methods, which typically have a lower variable cost and higher fixed cost. This is illustrated in Figure 2-9.

Total Annual Cost Comparison
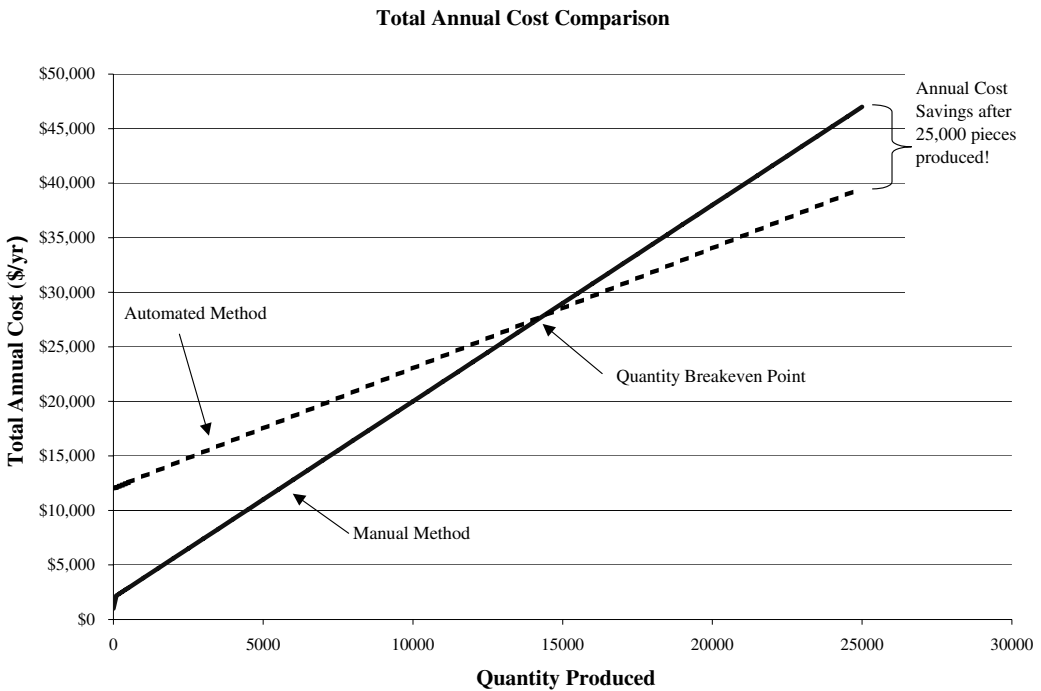

Figure 2-9 Quantity breakeven point

Figure 2-9 is a plot showing the total annual cost of an automated method versus a manual method for the same theoretical task. The manual method has a lower fixed cost but higher variable costs. The lower fixed cost is evident in the graph by the lower starting point (zero parts produced). The steeper incline indicates higher variable costs. At around

15,000 parts the two lines cross, indicating the two methods have the same total annual cost. This is termed the *quantity breakeven point*. As the quantity produced is increased from this point, the automated method has a lower total annual cost. If the annual production is 25,000 parts, the cost savings that the automated method provides is the *y*-axis value difference between the two lines.

Thus, to determine the quantity breakeven point of two methods, the total annual cost equations are set equal to one another (i.e., equate the costs) and solved for the quantity (Q). This is the quantity at which the proposed and the current methods have the same production cost. Anything above this quantity favors the automated method over the manual method for cost efficiency. This is demonstrated in the following example.

Example 2.16

A new automated method is being developed to replace the manual method described in Example 2.15. The new method has a production rate of 165 parts/hr, requires only one operator, and has a capital cost of $45.50/hr. Additionally, the new method decreases material waste, thus reducing raw material costs to $1.00/part. Because of machine sophistication, yearly maintenance costs will increase to $16,000 per year. Perform a productivity analysis to compare the two alternatives for annual production of 100,000 parts/yr. Is the proposed method more productive? Calculate the quantity breakeven point. What is the total annual cost savings if the proposed method were to be used?

Solution

First, perform the productivity analysis (i.e., compare the current method's hours and maintenance costs with the proposed). Most of the information can be substituted directly into a productivity calculation spreadsheet, with the exception of the capital input. The capital or machine rates for both alternatives are given. However, neither includes maintenance cost. Thus, the maintenance cost, spread over the 100,000 parts, will be added to the hourly machine rate. To proceed, calculate the number of hours it would take to produce 100,000 parts for each method, then divide the maintenance cost by this number.

For the current method,

$$hrs_{curr} = (100{,}000 \text{ parts/yr})/(100 \text{ parts/hr}) = 1000 \text{ hr}$$

$$(P_{\text{I maint}})_{curr} = \$8000/1000 \text{ hr} = \$8/\text{hr}.$$

For the proposed method,

$$hrs_{prop} = (100{,}000 \text{ parts/yr})/(165 \text{ parts/hr}) = 606.1 \text{ hr}$$

$$(P_{\text{I maint}})_{prop} = \$16{,}000/606.1 \text{ hr} = \$26.40/\text{hr}.$$

Add these values to the capital hourly rates for both current and proposed methods:

$$(P_{\text{I capital}})_{\text{curr}} = \$25/\text{hr} + \$8/\text{hr} = \$33.00/\text{hr}$$

$$(P_{\text{I capital}})_{\text{prop}} = \$45.50/\text{hr} + \$26.40/\text{hr} = \$71.90/\text{hr}.$$

Substituting these values into a productivity analysis spreadsheet yields the following:

Thus, the proposed method is 126% as productive as the current method. The quantity breakeven point indicates the production quantity after which the proposed method becomes more productive. These calculations are shown in the following spreadsheet:

Combined Productivity Comparison					
Description	Variable	Units	Current Method	Proposed Method	Formula or Comments
Production rate	P_{O}	pcs/hr	100	165	Given
Labor Partial Productivity					
Labor cost/hr	$P_{\text{I labor}}$	$/hr	$36.00	$18.00	Given
Labor productivity	$P_{\text{P labor}}$	pcs/$	2.78	9.17	$P_{\text{O}} / P_{\text{I labor}}$
Labor productivity index	I_{labor}	-	1.0	3.30	
Capital Partial Productivity					
Capital cost/hr	$P_{\text{I capital}}$	$/hr	$33.00	$71.90	hourly rate + maintenance cost per hour to produce 100,000 pcs
Capital productivity	$P_{\text{P capital}}$	pcs/$	3.03	2.29	$P_{\text{O}} / P_{\text{I capital}}$
Capital productivity index	I_{capital}	-	1.0	0.76	
Material Partial Productivity					
Raw material cost/hr	$P_{\text{I material}}$	$/hr	$125.00	$165.00	mat'l cost ($/pc) x production rate
Raw material productivity	$P_{\text{P material}}$	pcs/$	0.80	1.00	$P_{\text{O}} / P_{\text{I material}}$
Raw material productivity index	I_{material}	-	1.0	1.25	
Combined Productivity					
Sum of partial productivity inputs	SP_{I}	$/hr	$194.00	$254.90	$P_{\text{I labor}} + P_{\text{I capital}} + P_{\text{I material}}$
Combined productivity	P_{c}	pcs/$	0.52	0.65	$P_{\text{O}} / SP_{\text{I}}$
Combined productivity index	I_{c}	-	1.0	1.26	

The quantity breakeven point occurs at 16,837 parts. The annual cost savings of the new method will be $39,515 (from $194,000 – $154,485).

Description			Present Method	Proposed Method	Formula or Comments
Production rate	P_O	pcs/hr	100.0	165.0	= 1/T_c * 60 min/hr
Number of operators		-	2	1	Machine will only be attended by an operator 1/3 of the time
Labor wage rate	$P_{I\,labor}$	$/hr	$18.00	$18.00	
Machine wage rate	$P_{I\,capital}$	$/hr	$25.00	$45.50	
Yearly maintenance cost	$C_{F\,maint}$	$/yr	$8,000.00	$16,000.00	
Annual production quantity	Q	pcs/yr	100,000.00	100,000.00	
Quantity Breakeven Point Caclulations -- $C_T = C_F + Q$ x C_V					
Fixed cost	C_F	$/yr	$8,000	$16,000	maintenance cost + UAC
Material cost per part	$C_{V\,material}$	$/pc	$1.25	$1.00	material usage per part (lbs/pc) * material cost ($/lbs)
Labor cost per part	$C_{V\,labor}$	$/pc	$0.36	$0.11	(number of operators * $P_{I\,labor}$) / P_O
Machine cost per part	$C_{V\,capital}$	$/pc	$0.25	$0.28	$P_{I\,capital}$ / P_O
Variable cost	C_V	$/pc	$1.86	$1.38	$C_{V\,material} + C_{V\,labor} + C_{V\,capital}$
Quantity breakeven point	Q_{BE}	pcs		16,837	[(C_F)proposed - (C_F)current] / [(C_V) current - (C_V) proposed]
Total annual cost	C_T	$/yr	$194,000	$154,485	$C_F + Q\,C_V$

2.7 Productivity and the USA Principle

The benefits of productivity analysis in justifying automation of manufacturing processes have been extensively highlighted throughout this chapter. Additionally, we hint at how productivity analysis can be used to identify other enhancements or improvements to the proposed automation. In this section, we discuss using the *USA automation strategy* in conjunction with a productivity analysis as the starting point for productivity improvements through automation. Thus, instead of performing the productivity analysis *after* an automated method has been proposed, the analysis will be used *during* the development of the automated method.

Groover outlined the basic tenets of the USA principle in *Automation, Production Systems and Computer-Integrated Manufacturing*. It is a simple, common sense approach to developing an automation strategy. "USA" is an acronym for the method's steps:

Understand the process. This is the crucial first step. There is no better way to understand an existing process than to calculate its productivity. It compels the determination of cycle times, production rates, material costs, and so on. These data can be assembled through time studies, video analysis, and through other data collection techniques. Once all the data are assembled, a preliminary productivity analysis is performed, one that uses the spreadsheet presented in Section 2.6. Data collection and productivity analysis give one a thorough grasp and deep understanding of the existing process.

Simplify the process. It is likely that a process under consideration has never been as extensively evaluated as it will be with these methods. Thus, simple improvements or modifications identified in step 1 may greatly enhance the performance of the existing process. Wasted movements, actions, or procedures can be eliminated and the process reevaluated, the idea being that the new process should be as streamlined as possible and have every opportunity to succeed. Once the new process has stabilized, another productivity analysis using the new data is performed.

Automate the process. Armed with extensive, in-depth knowledge of the simplified existing process, one begins identifying ways to improve productivity through automation, using the automation strategies identified in Section 1.6 as a starting point. Once a general strategy has been selected, a productivity analysis is done, one that compares the existing process and the proposed automated one. The productivity analysis spreadsheet is used and the automated method's performance adjusted until the desired productivity improvement is achieved.

After the USA principle is applied, data from the productivity analysis in step 3 are the specifications for the new, automated process; these data are used for cost quoting purposes. Once quotations are received, the productivity analysis can be reevaluated. Thus, when it comes time to submit a proposal to upper management for the automation project, justification will have already been completed. Using the USA principle in conjunction with productivity analysis greatly enhances the probability of a successful automation project.

2.8 Summary

Productivity calculations provide a very effective means for identifying, evaluating, and justifying the use of automation in a manufacturing facility. Productivity of a manufacturing system is determined by the ratio of the process outputs divided to the process inputs. If only one input (such as labor) is considered in the calculation, then the calculation is called a partial productivity (P_P) calculation. When two or more inputs are included, the calculation is called a combined productivity (P_C) calculation.

Process measures are used to quantify manufacturing processes. These measures then fill the role of outputs in productivity calculations. The most important process measure in terms of productivity calculations is production rate—the measure of how many parts are produced over a specific time frame, typically expressed in parts per hour. Production rate is calculated from the operational cycle time that includes all time element activities involved in producing one part.

Other important mathematical quantifying concepts include production capacity (the maximum rate of output of a particular product for a manufacturing system over a specified time period), utilization (the ratio of the actual number of products divided by production capacity, expressed in percent), and availability (how often a machine is

actually available to perform processing). Manufacturing lead-time is the total time it takes to convert the raw material into the finished product.

Input of the productivity calculation (P_I) is the amount of money into the process over the same time frame used in the output measurement. Inputs to the process are typically broken down into categories consisting of capital, energy, labor, and material. All of these inputs need to be expressed in terms of dollars per hour. For energy, labor, and material the calculations are straightforward. Capital costs of automation are determined by breaking initial cost of equipment into annual cost spread over annual hours the machine is estimated to run, the result added to factory overhead expenses.

Productivity calculations are a very effective method of comparing automation alternatives. Productivity index is then calculated, giving a clear and concise method for comparing partial and combined productivity measures of the two options being evaluated. One of the options, typically the current method, is assigned a baseline productivity index of 1. If a proposed option has a combined productivity index greater than 1, it can be said that it is more productive than the current method. A combined productivity comparison can serve as a starting point or roadmap for identifying the type and quantity of improvements necessary to justify automation.

To ascertain whether there is sufficient production volume to justify an automation investment, it is important to consider the current and proposed methods' fixed and variable manufacturing costs. Fixed costs are independent of production quantity; variable costs, on the other hand, are dependent on the quantity. A production volume breakeven point analysis calculates the volume that justifies automation. The quantity breakeven point of two methods is found by setting the total annual cost equations equal and solving for quantity (Q), at which the manual and automated methods cost the same. In general, when product volumes are low, manual methods are more cost effective. As production volumes increase the advantage goes to automated methods.

The USA automation strategy directs us to **u**nderstand an existing process, to **s**implify it, and if it is called for, to **a**utomate it. Using USA in conjunction with productivity analysis greatly enhances the probability of a successful automation project.

2.9 Key Words

actual processing time
availability
average production time
batch processing time
bottleneck station
capital expenditure
combined productivity
fixed costs
manufacturing lead-time
operational cycle time
partial productivity
production capacity
production rate
productivity
productivity index
quantify
quantity breakeven point
tool handling time
USA principle
utilization
variable costs
workpiece handling time

2.10 Review Questions

1. How can productivity calculations aid in identifying, evaluating, and justifying automation?
2. Explain the difference between partial and combined productivity.
3. A manufacturing process can produce 640 parts/hr. The process requires three laborers each earning $26/hr. What is the labor partial productivity of the process?
4. The manufacturing process described in review question 3 uses a machine that has a capital cost of $95/hr. The machine operates on 150 kW of power. Cost of electricity energy is $0.057/kWh. The machine processes 215 lb material/hr. The material costs $0.85/lb. Using the labor input costs determined in Example 2.1, calculate the combined productivity of the process.
5. The following table lists the steps for a machining process. The listed times are required to load, unload, and process one part. Calculate the operational cycle time (t_c) and production rate (R_p).

Process Steps	Description	Time
1	Part loaded into machining fixture	4 min
2	First machining operation	4.5 min
3	Part repositioned in fixture	0.5 min
4	Second machining operation	8.5 min
5	Part unloaded	3 min
Note: After 50 parts the cutting tool is changed, a step that takes 5 min.		

6) An injection molding machine processes a 24-cavity mold in 1.3 min/cycle. The parts are automatically ejected from the mold and travel by conveyor to the next process. After every 500 cycles the mold is cleaned and sprayed with mold release. This takes 8 min to complete. Calculate the operational cycle time (t_c) and production rate (R_{pq}).

7) Calculate the cycle rate (R_c) of the flow-line manufacturing system shown in Figure 2-11, assuming the transfer rate is 5 sec/part and the processing time for each work station is as shown in table below.

Workstation	Processing time (min/pc)
1	0.5
2	1.25
3	1.25
4	1.85
5	0.95

Figure 2-11 Flow-line manufacturing system

8) Calculate the monthly production capacity (P_c) of a product made by the injection molding process described in review question 6. Assume the plant uses 8 injection molding machines and molds to produce the part. Also, assume the plant operates in three 8-hour shifts per day, 5 days per week. How could the plant increase production capacity in the short term? In the long term?

9) A manufacturing system has a theoretical production capacity of 500,000 parts/month. Typical use of the system is 97% and availability is 99%. What is the anticipated actual monthly production of the system?

10) A part is routed through 6 machines in lot sizes of 300 parts/batch. Average non-operation time is 5 hr. Setup and operational cycle times are shown in the table below. Calculate the manufacturing lead-time for the part.

Machine	Setup time (hr)	Operation time (min)
1	1	3
2	6	8
3	1.5	4
4	4	3
5	2	3
6	3	2

11) An automated work cell is being considered to replace an existing process. The cell will cost $350,000 to purchase and is anticipated to have a 7-year service life. The machine will operate for 2080 hr/yr. Also, consider that the company spent $4,600,000 on factory overhead and $8,250,000 on direct labor costs the preceding year. If the manufacturing firm desires a 10% return on its investment, estimate the hourly capital cost to operate the new automated work cell.

12) A manufacturing firm utilizes a manual machine to make a product. The production rate is 125 parts/hr. This current method uses three operators at a labor wage rate of $18/hr. The manual machine's capital operation cost (including cost of electricity) is $34/hr. The firm is considering replacing the manual machine with a programmable automation work cell. The new cell only requires one operator, but it has a capital cost (including cost of electricity) of $95/hr. The production rate of the machine is 225 parts/hr. Perform a combined productivity analysis to determine if the firm should purchase the automated work cell.

13) Referring to the manual manufacturing method information given in review question 12, calculate the total annual cost to produce 100,000 parts per year. Note that the annual cost of maintenance for the machine is $16,000. Additionally, the raw material cost is $2.50 per part.

14) A new automated method is being developed to replace the manual method described in review questions 12 and 13. The new method has a production rate of 245 parts/hr, requires only one operator, and has a capital cost of $65.50/hour. Additionally, the new method decreases material waste, thereby reducing raw material costs to $1.00/part. Because of machine sophistication, yearly maintenance costs will increase to

$26,000 per year. Perform a productivity analysis to compare the two alternatives if annual production is 100,000 parts/yr. Is the proposed method more productive? Calculate the quantity breakeven point. What is total annual cost savings if the proposed method is used?

2.11 Bibliography

1. Groover, M.P. (2001). *Automation, Production Systems and Computer-Integrated Manufacturing,* 2nd ed. Prentice Hall, Upper Saddle River, New Jersey.
2. Sumanth, David J. (1994). *Productivity Engineering and Management*. McGraw-Hill.
3. Kandray, Daniel E. (2004). Comparison of fixed automation and flexible automation from a productivity standpoint. Society of Manufacturing Engineers Technical Paper TP04PUB206.
4. *Machinery's Handbook,* 25th ed. (1996). Industrial Press, Inc., New York, New York.

Chapter 3

Introduction to Computer Numerical Control (CNC)

Contents

Objective

The objective of this chapter is to provide a thorough understanding of the terminology and basic operating concepts of computer numerical control (CNC) technology.

3.1 Introduction to CNC Technology

Computer numerical control (CNC) technology is, in the simplest of terms, the automation of traditional manual machining processes by electrical and computer technology. In traditional "manual" machining, a machinist (or operator) decides upon and directs the motion of a tool relative to the workpiece, thus creating the desired shape of a finished workpiece. In CNC technology, a *computer controller* plays the role of the machinist, so to speak, directing the motion of the tool by following a stored sequence of coded machine commands or directions called a *program of instructions*, or more traditionally, a *part program*. A sample CNC program of instructions is shown in Figure 3-0. The program directs the motion of the tool relative to the part and contains commands that control all essential machine functions, such as tool choice, spindle rotation speed, tool feed rate, and other functions. The program of instruction, written in a language that is understood by the CNC controller, is often called a *G-code program* because its commands are alphanumeric codes beginning with the letter "G." This is evident in Figure 3-0.

```
N10   G90 G70
N20   T17 M06
N30   G00 X0 Y0 Z1
N40   F90 S4000 M03
N50   G00 Z0.1
N60   G00 X1.5 Y1.0
N70   G01 Z-.375
N80   X1.6725
N90   G02 X1.6725 Y1.0 I-.1725 J0
N100  G01 X1.845
N110  G02 X1.845 Y1.0 I-.3.45 J0
N120  G01 X2.0303
N130  G01 X1.5 Y1.5303
N140  G01 X.9697 Y1.0
N150  G01 X1.5 Y.4697
N160  G01 X2.0303 Y1.0
N170  G00 Z0.1
N180  G00 X.5 Y.25
N190  G01 Z-.375
N200  G00 Z0.1
N210  G00 Y1.75
N220  G01 Z-.375
N230  G00 Z0.1
N240  G00 X2.5
N250  G01 Z-.375
N260  G00 Z0.1
N270  G00 Y.25
N280  G01 Z-.375
N290  G00 Z0.1
N300  G00 X0 Y0 Z1
N310  M05
N320  M30
```

Figure 3-0 Sample G-code program

3.1.1 Manual Machining and Numerical Control Technology

Manual machining is still used in industry for low volume applications, maintenance, and repair. In manual machining, mechanical technology in the form of *slides*, *gears*, *belts*, and *feed screws* implements a tool's movement relative to a workpiece. A typical manual vertical milling machine is shown in Figure 3-1.
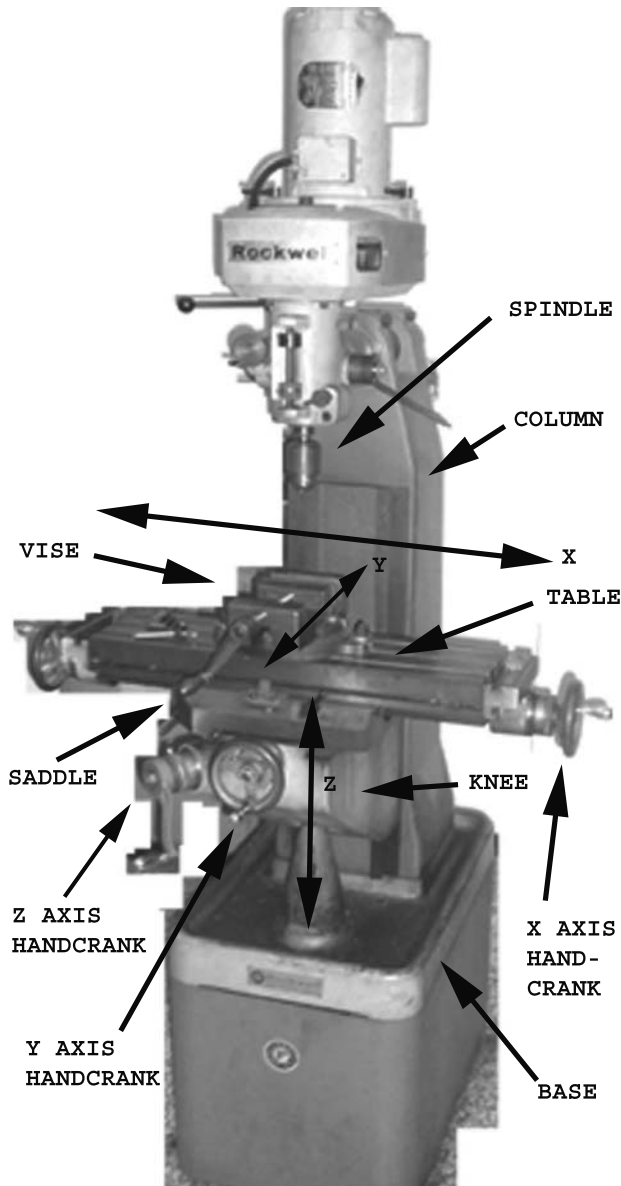

Figure 3-1 Manual vertical milling machine

A part is milled or machined by fastening the workpiece to the machine and moving the workpiece into the rotating cutter, held by the spindle, at a specific *feed rate* and *depth of cut*. Spindle rotational speed and direction is often controlled with gears or belts and pulleys. The workpiece is fastened to the machine with some type of *fixturing*. In Figure 3-1 the fixturing is a simple vise. The vise, in turn, is fastened to the *mill table*. The mill table, and hence the workpiece, can then be moved in three directions relative and perpendicular to the spindle.

The *Cartesian coordinate system* supplies the layout of the directions in which the mill table can be moved. Again, as shown in Figure 3-1, the mill table can move longitudinally across the front of the machine. This is shown as the x direction in the figure. The table can also be moved at a right angle to the x direction, into the machine, designated as the y direction. The third direction is along the spindle axis, and is shown as the z direction. Linear bearings, called *slides*, or *ways*, both short for "slipways," guide the movement of the table along each axis.

Figure 3-2 shows a manual vertical milling machine with exploded views of the x- and y-axis slides and *lead screws*. The table is moved along a specific axis by turning the appropriate hand crank. This in turn drives the lead screw (or feed screw), which pushes or pulls the table along the slides. Figure 3-3 shows a closer view of the slides. Note that *dovetail slides* are used to constrain motion perpendicular to the sliding direction. The feed screws for each axis can also be powered by the machine and moved at specific speeds or rates.

Figure 3-3 shows such a machine, with the x-axis equipped with a power feed. Typically, in manual milling, powered table movement occurs in only one direction at a time. Standard operations for manual vertical mills are slot cutting, planing, and hole drilling. Movement that occurs between any pair of axes during the cutting operation is not very accurate and is difficult to accomplish. Cutting complex surfaces may require movement in the direction of all three axes. However, such an operation is not possible on a traditional manual mill; *numerical control technology* was developed to specifically address this limitation.

During the 1940s a contractor to the U.S. Air Force by the name of John Parsons began experimenting with methods to produce more accurate inspection templates for helicopter blades. The inspection templates were a complex airfoil shape. Machining these shapes accurately was a challenge. Parsons' method involved calculating points along the airfoil's shape and then, using two operators (one for each axis), manually moving the machine tool to each of these points. Because the calculations were so complex, Parsons used a punch card tabulating machine to perform the calculations. The punch cards would be fed into a card reader at the machine, which would read the data, then pass the information on to a machine controller, which in turn directed the motion of each of the machine axes.
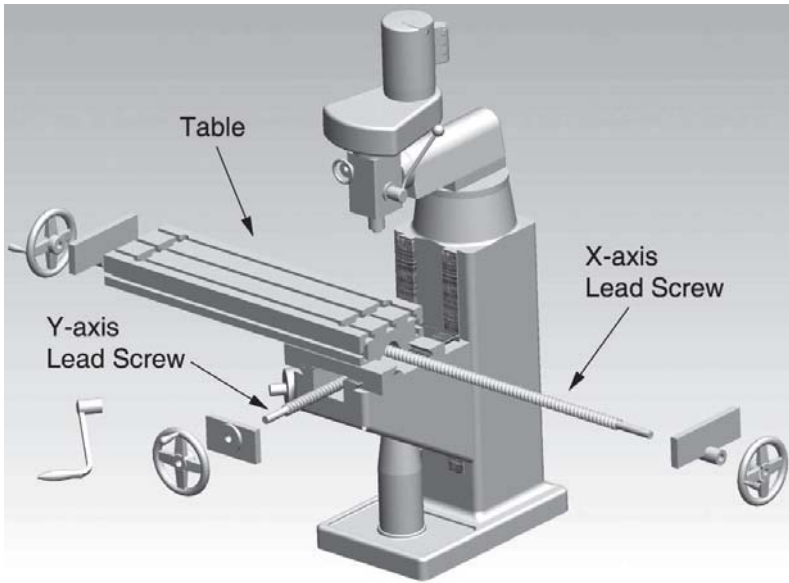
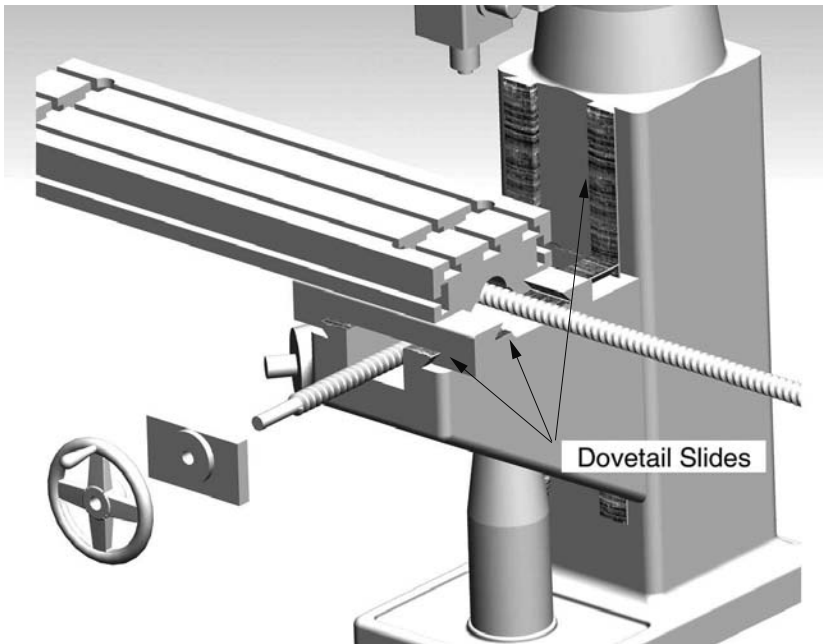*Figure 3-2 Exploded view of manual vertical milling machine*

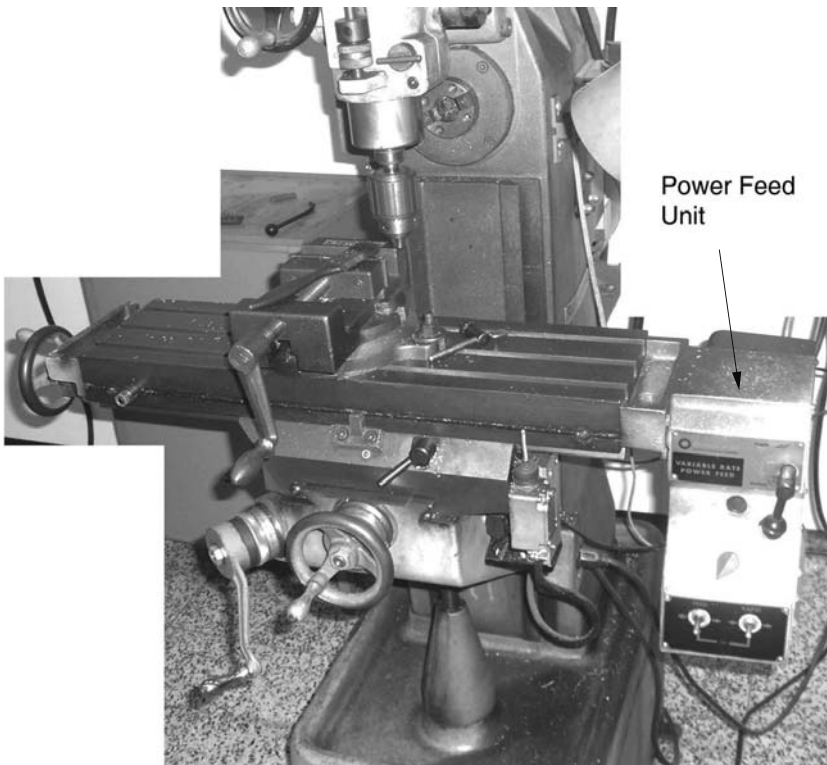*Figure 3-3 View of dovetail slides of manual vertical milling machine*

Figure 3-4 Manual machine with power-driven x-axis

This method, although much more accurate than manual machining, was still very time-consuming. However, the Air Force was much impressed and awarded Parsons a contract to develop a machine to provide automated control of the axes.

Parsons' machine concept is, fundamentally, the system that all-modern CNC equipment uses today. However, today's systems are substantially upgraded due to the rapid development of computer technology. Punch cards were replaced by magnetic tapes, which in turn were replaced by electronic files. These electronic files, or "part programs," are now either created directly at the machine or developed offline at a separate computer. Figure 3-5 shows an older numerical control—or "NC"—machine, which used programs stored on magnetic tape. Note the size of the tape reader/machine controller. Drive motors for the axes are also visible. Prior to the advent of computer technology most machines were referred to as "NC machines." They were hardwired using vacuum tubes, transistors, and relay technology. In the 1970s and 1980s, microcomputers replaced the aging hardwired technology. These machines were called *computer numerical control,* or *CNC*, machines. However, NC is still often used interchangeably with CNC. Figure 3-6 shows a modern CNC machine center.
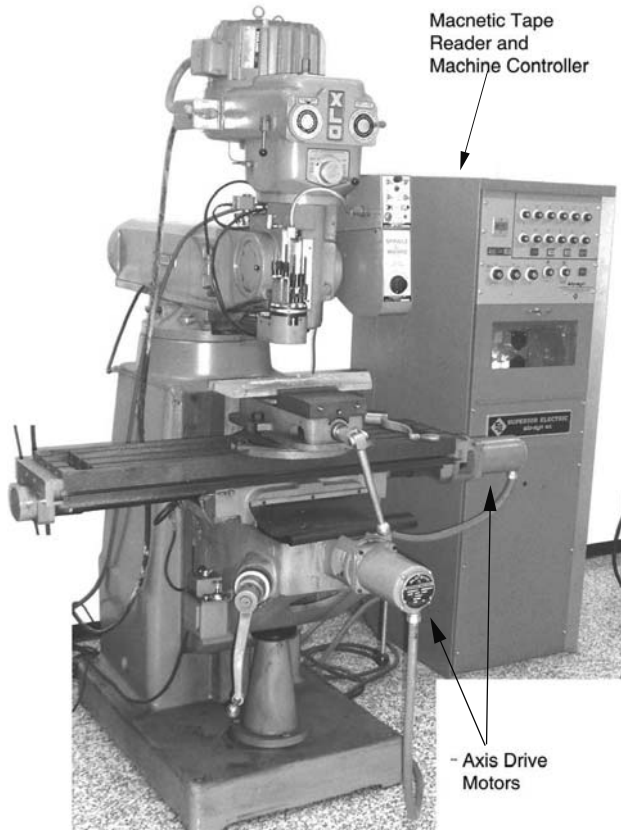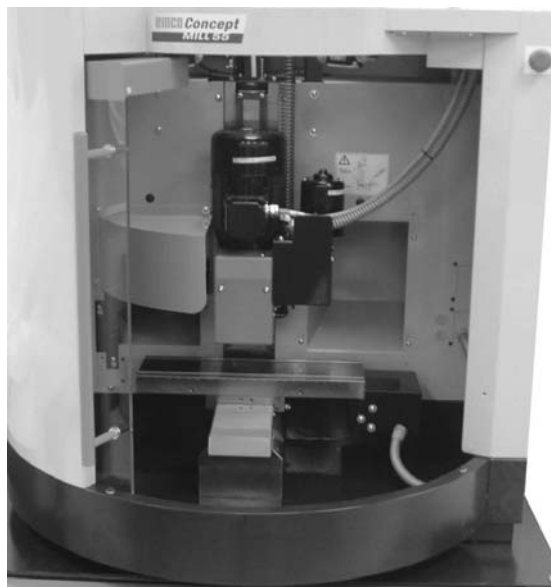
Figure 3-5 Vintage numerical control (NC) machine

Figure 3-6 Modern computer numerical control (CNC) milling machine

3.2 CNC System Components

CNC technology has been applied to numerous machine tools, including lathes, mills, electric discharge machines (EDM), and flame, laser, and plasma cutting machines. Non-machine tool examples include coordinate measuring machines (CMM), component insertion machines (assembly machines), wire-bending machines, and polymer composite filament winding machines. Figure 3-7 shows a numerical controlled wire-bending machine. Essentially, any type of processing equipment that needs to move a tool relative to a workpiece is a prime candidate for CNC technology. Thus, a CNC system can be broken into four major components:

    1. Processing equipment/machine tool
    2. Drive mechanism/positioning system
    3. CNC controller
    4. Program of instructions.

Figure 3-7 CNC wire-bending machine

3.2.1 Processing Equipment/Machine Tool

For processing equipment in general, the workpiece−tool relative motion is executed in two or three directions; however, as many as five different directions can be controlled in modern equipment. Each direction corresponds to an axis of the machine. A standard CNC mill has three axes: two horizontal axes corresponding to the x,y-plane and a vertical axis for movement of the spindle corresponding to the z-axis. The axis with the longest travel is generally labeled "x-axis." Another standard in the machine tool industry is the correspondence of the axis of the machine's spindle with the z-axis. Hence, for a lathe, the tool motion is specified in the direction of the x- and z-axes. The tool moves in the x direction, in and out of the workpiece orthogonal (at right angles) to its rotational axis. This is shown in Figure 3-8.
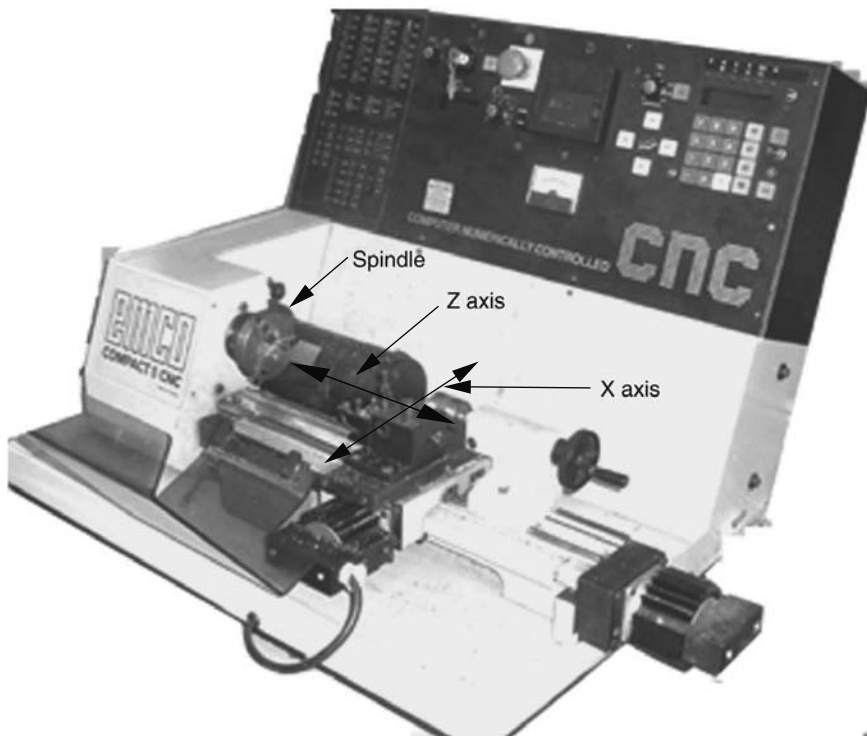
Figure 3-8 CNC lathe with axes labeled

Additional axes are often added to mills in the form of multiaxis rotational tables, or fixturing, resulting in four or five axis machines. This is discussed in greater detail in later chapters. As mentioned previously, any type of processing equipment where controlling the location of the workpiece relative to the location of a tool is important is a prime candidate for CNC control.

3.2.2 Drive Mechanism/Positioning System

Movement along the various axes of the machine is accomplished with mechanically guided high precision linear bearings, called slides or ways, and a lead screw. A carriage or table is moved along the slide (or axis) by the lead screw. The lead screw transforms rotational motion from an electric motor into linear movement along an axis. Early machines used hydraulic motors controlled with *servo valves*, an approach still used for very large machines.

The majority of modern CNC tools have electric *servomotors* to drive the lead screw. Figure 3-9 shows a cutaway view of a typical servomotor *closed loop system*, for controlling one axis. In order to perform complex contouring in the *x,y*-plane, two axes are required, one for each direction. This is accomplished by mounting one axis control

system on top of another. This is evident in Figure 3-9. One axis control system moves the table across the machine, the other moves the saddle, and hence the table, in and out of the machine. By adding vertical axis control to the spindle, complicated three-dimensional shapes can be machined. Orchestrating and synchronizing the motion in all three directions is the task of the CNC controller.
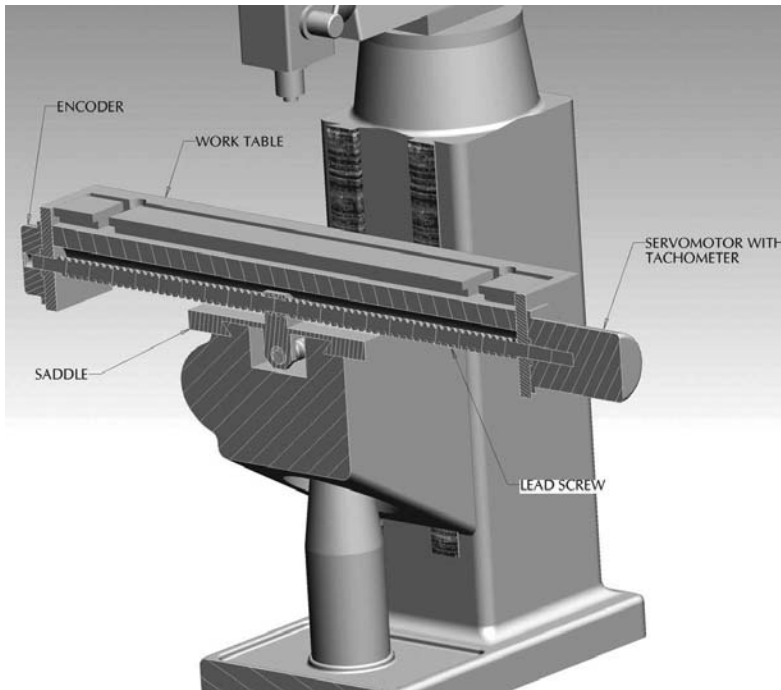

Figure 3-9 CNC mill servodrive components

3.2.3 CNC Controller

The CNC controller directs all machine functions while executing the program of instructions. It interfaces with the machine operator during machine and tool setup. It selects the desired tooling and positions the workpiece in the correct location relative to the tool. It turns on spindle and coolant flow and moves the workpiece and/or tool along the correct tool path at the specified feed rate, often directing the motion along three axes simultaneously. It can also make decisions and instruct other machines to perform a specified task. For example, it can communicate with a machine-tending robot to load or unload a workpiece. When it does this, it must cease machining operations, open any guards or gates to allow easy access to the workpiece, release the automated fixture, and instruct the robot to proceed. Not only does the controller have to execute all of these impressive tasks accurately and in the correct order, but also the tasks must be carried out very quickly. This is particularly true for the *motion control* of the axes, which is perhaps

the most important and complicated function the controller provides. Thus, the motion control is in a category among controller functions are listed. The other functions discussed above are categorized as either *auxiliary control* or part of the *operator interface*. This is shown in Figure 3-10.
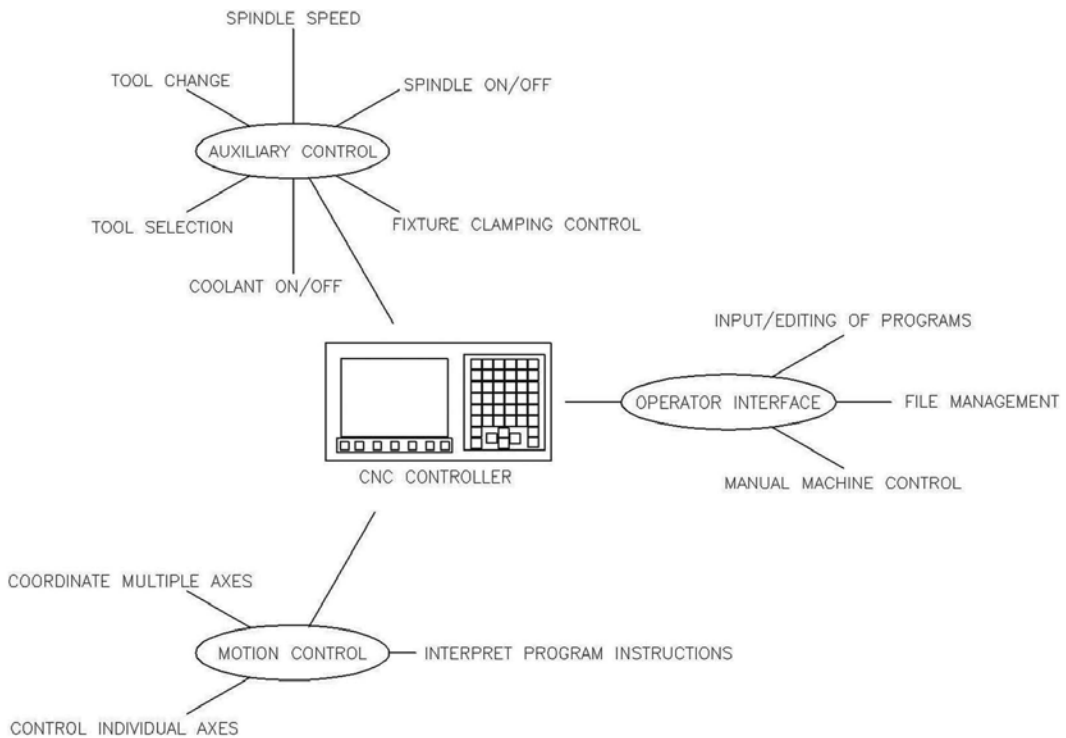

Figure 3-10 CNC controller functions

Although the performance capabilities of a CNC machine are impressive, the machine cannot perform a single task without first being properly set up. In the machining industry the term "setup" refers to preparation of the machine or machines to process a specific workpiece in a specified manner. Setup typically involves acquisition and installation of the correct cutting tools and fixturing. In CNC machining, setup also entails the correct part program be loaded into memory and specifies the *program reference zero* (*PRZ*). The PRZ is the location of the x, y, and z zero positions of the coordinate system that the program references when providing relative positioning of the tool and workpiece. Setup-related tasks are performed through the operator interface of the machine. The PRZ is discussed in more detail in subsequent sections.

The operator interface typically (and minimally) consists of a CRT or LCD screen and keypad. Small nonindustrial machines used by hobbyists or by operators in the

educational environment use a PC that runs operator interface software. Regardless of the type of operator interface, the functions it provides are the same, and they typically include manual motion control of the axes (for establishing the PRZ), manual control of spindle rotation speed and direction, tooling specifications and setup, starting and stopping the machine, program file management, and input and editing of programs—as well as other functions. An operator interface for a CNC lathe is shown in Figure 3-11.
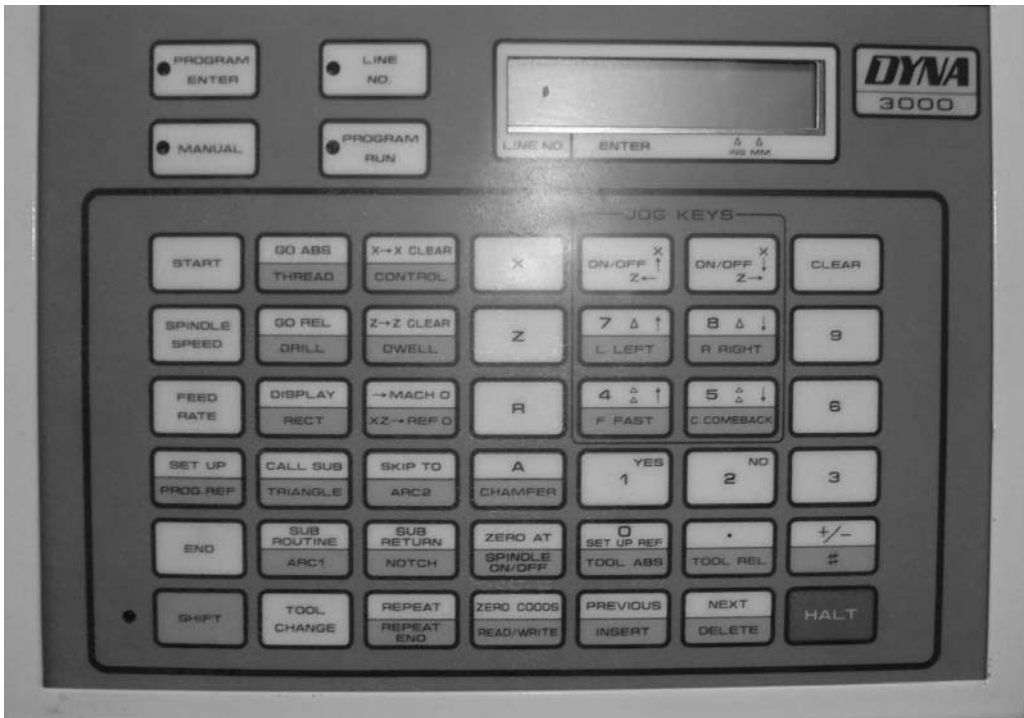

Figure 3-11 CNC operator interface

Auxiliary control is essentially *discrete* control over machine functions that support the machining or processing operation—functions that are not directly related to motion (*continuous*) control of the axes. This is an important distinction because it affects the coding used in the program of instructions (see Chapter 4). Auxiliary control is considered a form of discrete control in the sense that the status of the function (spindle motor) changes at discrete or distinct moments in time. Additionally, the status is binary (1 or 0). As an example, consider the status of the spindle motor during a machining operation. Its status is either on or off. This can be represented in binary terms by either a 1 (on) or 0 (off). This is in stark contrast to the motion control of the axes. The status or position of an axis is continuously changing. Hence, motion control is a form of continuous control.

Figure 3-12 shows the motion control system for a single axis of a vertical mill. Based on a program of instructions, the CNC controller sends the desired position and sets

the speed at which it travels to the *motor drive unit*. The motor drive unit then analyzes this signal and turns on and rotates the servomotor to obtain the desired position at the desired speed. The *encoder* informs, or feeds back, the position of the axis continuously to the motor drive unit, while the tachometer reports the rotational speed of the lead screw. The motor drive unit continuously compares the desired position and speed to the actual position and speed and adjusts the servomotor accordingly. This type of continuous control is termed a *closed loop feedback control system*. Most CNC machines used in industry employ this type of control. However, for smaller CNC machines used by hobbyists and in academia, an *open loop control system* is often used.
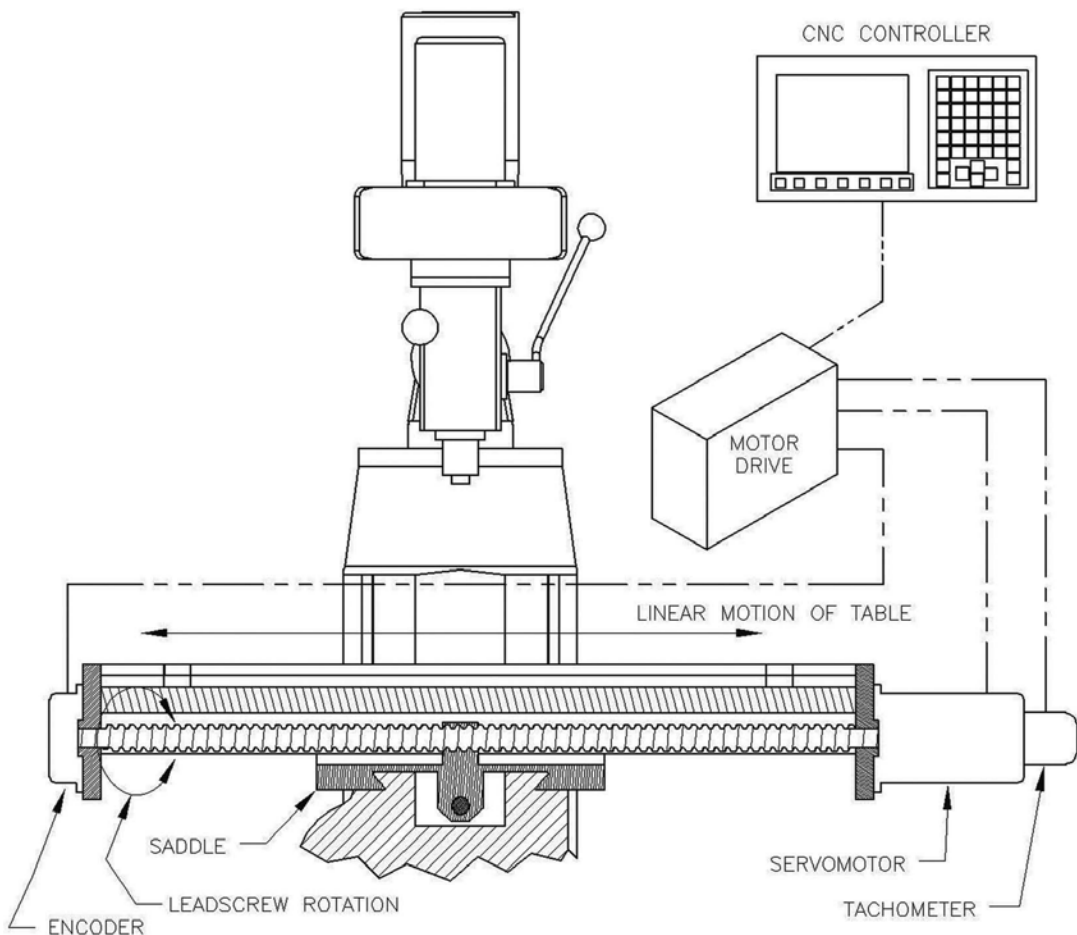

Figure 3-12 Single-axis closed loop motion control system

An open loop control system utilizes a *stepper motor* instead of a servomotor to control axis movement. Also, the system does not use a feedback device such as an

encoder or tachometer. Hence, it cannot provide feedback to the motor controller regarding the actual axis position and speed. Thus, the signal sent to the stepper motor is the only means by which the speed and position of the axis is controlled. The signal is sent, and the controller assumes the axis has achieved the desired position at the desired speed, but this assumption cannot be verified. Even though open loop control systems are much simpler and less expensive than closed loop feedback control systems, they are generally only used where resistance to axis motion is low. High resistive forces could prevent the axis from achieving the desired position. A single-axis open loop control system is depicted in Figure 3-13.
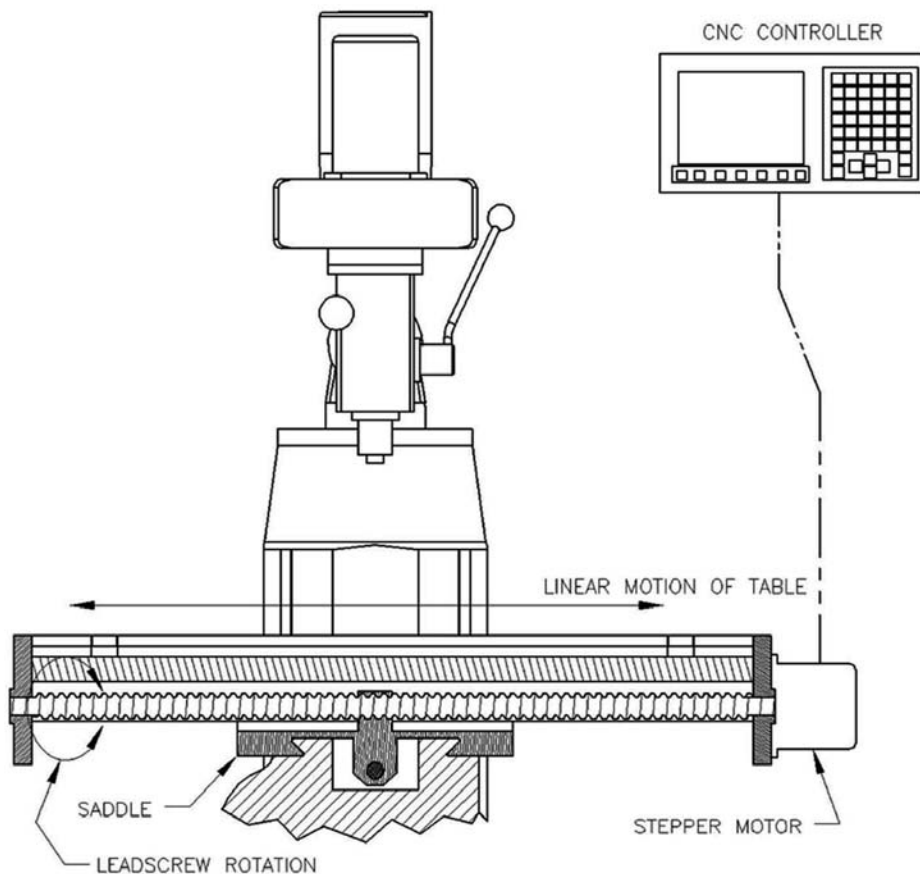

Figure 3-13 Single-axis open loop motion control system

In a typical machining operation, simultaneous control of more than one axis is required. For a CNC lathe or a CNC plasma cutter, two axes will be controlled simultaneously. For a mill, at least three and up to five axes have to be controlled. Therefore, the motion control system has to perform complicated geometric computations

to coordinate the motion of each axis to achieve the desired relative position of the tool to the workpiece. Additionally, the path taken to achieve that position may be of particular importance.

Consider Figure 3-14. Figure 3-14(a) shows the desired end product. Two holes are to be drilled at positions A and B. First, the tool is moved to position A and the hole drilled. Subsequently, the tool moves to position B to drill the second hole, as shown in Figure 3-14(b). The path taken to get to position B is of little importance. What is important is that tool position B be located accurately. This type of control, where the tool is moved from point A to point B without consideration of the path taken is called *point-to-point control*. To achieve this motion two single-axis motion control systems can be placed together, one to represent the *x*-axis and one to represent the *y*-axis. A signal is sent to the *x*-axis motor drive unit to move to position x_B and another is sent to the *y*-axis motor drive unit to move to position y_b. Each axis will move to the desired position without regard to the motion of the other axis. This is the essence of point-to-point control: to obtain the desired position without regard to the path taken. Figure 3-14(c) shows a graph of how the path may look on the (*x,y*)-coordinate system, depending on the values of x_B and y_B. This type of control is suitable for drilling operations, except when the path taken between the two positions is important, as is true in milling and turning (lathe) operations.

Figure 3-15(a) shows a block with a diagonal slot milled in it from point A to B. In order to perform this operation, the tool will be positioned above point A, then moved down into the workpiece to the desired depth. From this position it is moved to point B following a straight (linear) path, as shown in Figure 3-15(b). In this case, the path taken from point A to point B is important. Additionally, the speed at which the tool moves along this path is also very important because of speed's effect on the surface finish of the final product. The resulting tool path is shown graphically in Figure 3-15(c). Consequently, the CNC controller must perform complicated calculations, and do so very quickly, to ensure that the desired path is followed at the desired speed. This is called *continuous path control*, which the controller must "interpolate" to achieve. This idea is now explained.

As is true of point-to-point control, two single-axis motion control systems are placed together, one to represent the *x*-axis and the other to represent the *y*-axis. However, for continuous path control, the CNC controller will generate a series of intermediate positions along the path between positions A and B. This represents an *approximation* of the continuous path. An approximation is necessary because the CNC controller is digital. Therefore, to define a continuous path in digital terms, the programmed path must be broken up into smaller finite line segments that approximate the desired path. This process is called *interpolation*. The type of interpolation performed depends on the type of continuous path that needs to be approximated. For example, the straight line segment shown in Figure 3-15(c) requires *linear interpolation*. A circular path, such as a path along

an arc, requires *circle interpolation*. These are by far the two most common interpolation methods used in milling and turning operations.
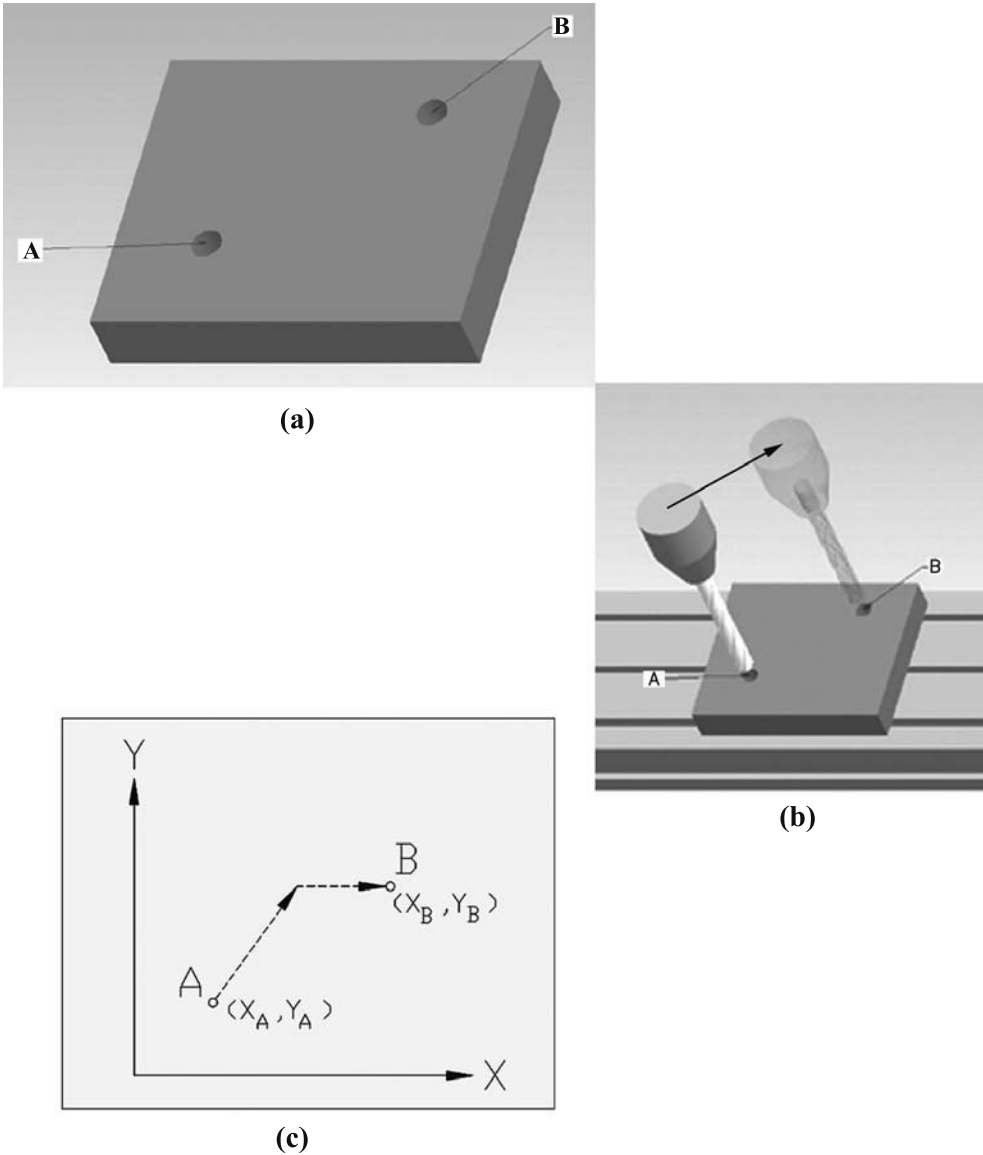

(a)

(b)

(c)

Figure 3-14 Point-to-point control example (a) End product (b) Point-to-point tool move

(c) Plot of axes movements
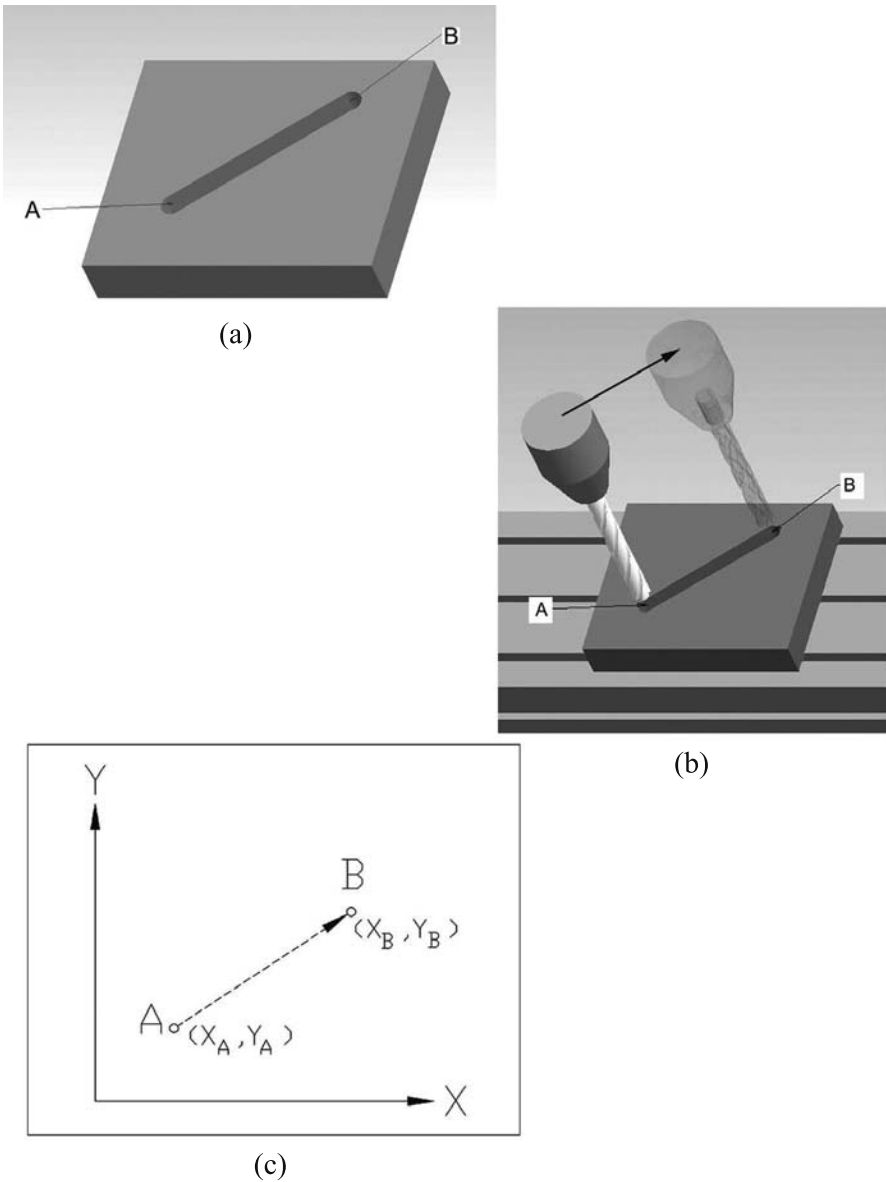
(a)

(b)

(c)

Figure 3-15 Continuous path control example (a) End product (b) Continuous path tool move

(c) Plot of axes movements

Other, not as common, interpolation methods include:

Helical interpolation—Combines linear interpolation in one axis and circle interpolation in two other axes.

Parabolic interpolation and *cubic interpolation*—These methods are used to represent higher order free form curves. The calculation methods are complex and time

consuming and are generally not suited for real-time calculation by the motion controller. Consequently, algorithms are used to simplify the curves into linear and circular segments before the program is downloaded to the motion control system.

Figure 3-16(a) shows a continuous path defined by an arc segment. The starting point A, ending point B, and the radius R suffice to define the path mathematically. Figure 3-16(b) shows how the curve is approximated with circle interpolation. The resolution or amount of error in the approximation is also evident in Figure 3-16(b). The higher the number of intermediate points, the smaller the error in the approximation. Typical industrial CNC equipment's interpolation error is very small, not discernable to the naked eye or touch.
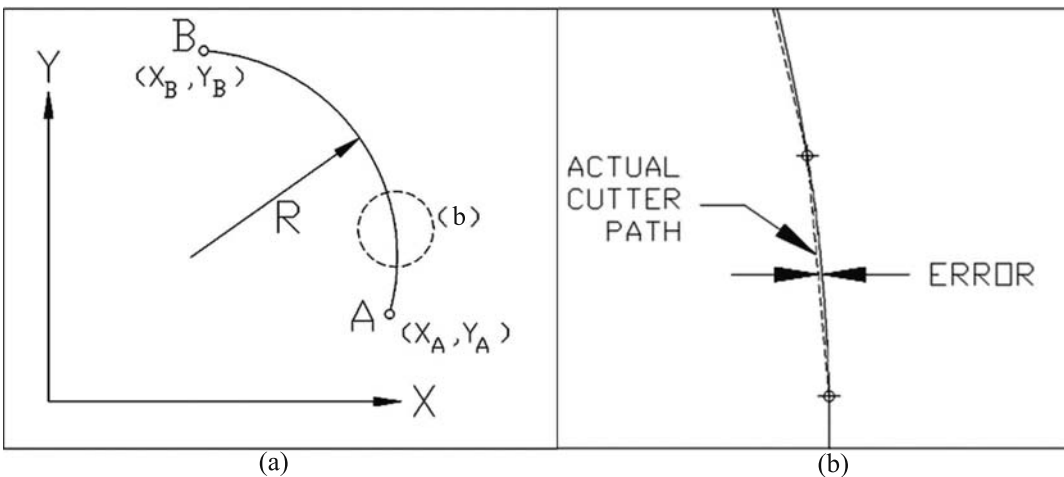

(a)   (b)

Figure 3-16 Circle interpolation example

The programmer gets an added benefit from the interpolation methods: the ability to define complicated shapes with relatively few input parameters. Again consider Figure 3-16. Without circle interpolation the programmer would have to manually calculate the intermediate points that approximate the curve and enter these intermediate positions in the program of instructions. This would be very time consuming, cumbersome, and make the program of instructions very large. With circle interpolation the programmer defines the path with only three parameters: the ending point (x_B, y_B), the radius, and the direction (clockwise or counterclockwise). The CNC controller, through circle interpolation, does the rest! This is discussed in greater detail in Chapter 4.

The use of continuous path control in CNC milling and turning has made necessary additional terminology regarding the type of cut performed. When continuous path control is used to move the tool along only one axis (x, y, or z), it is called a *straight cut*. The term *contouring cut* is used to define the machining operation in which continuous path control is used to move the tool along two or more axes at the same time.

CNC milling and turning operations utilize both point-to-point control and continuous path control. Whenever the tool is not engaged in the workpiece—i.e., not performing an actual machining operation—then point-to-point control is used. Examples include positioning the tool prior to performing an actual machining operation or moving the tool to a set position to perform a tool change. It is not important how one gets to the position, only that one gets to the position quickly. Thus, the path taken in point-to-point control is of little importance. Conversely, with respect to an actual contouring or straight cut, the path, and speed along the path are very important; hence continuous path control is utilized.

3.2.4 Program of Instructions

The program of instructions, known variously as the CNC program, G-code program, or simply "the program," is a sequential list of alphanumerically coded machine instructions. The CNC program instructs the CNC controller as to the tasks that are to be completed, when they are to be executed, and how they should be accomplished to perform the desired machining operation(s). The program is *coded*, that is, written in a language the CNC controller can understand. It is developed by an individual referred to as the CNC programmer. The programmer must possess extensive experience with the type of processing the machine performs and the code or language in which the instructions are written.

The program is accessed by the CNC controller through the machine interface, either by direct input or from a stored electronic file. The program contains all the auxiliary control and motion control instructions. The CNC controller executes these instructions sequentially, line (or block) by line, by interpreting the instructions as either auxiliary control or motion control functions.

The auxiliary control instructions are termed "miscellaneous" or "auxiliary" because they are necessary for the processing (or machining) but are not part of the tool moves. Miscellaneous functions are written with the letter "M" prefix followed by two numeric digits and are appropriately called *M-codes*. The motion control instructions are called "preparatory" because they prepare the CNC controller for an actual tool move. As previously noted, these are called G-codes. The number of G-codes in a CNC program typically far exceeds the number of M-codes. Consequently, "G-code program" is often synonymous for "CNC program" or "program of instructions."

The simple G-code program shown in Figure 3-18 was written to produce the part shown in Figure 3-17. Notice that the program is broken down into three major sections, titled *program setup*, *material removal,* and *system shutdown*.

The program setup section, shown in Figure 3-18, contains instructions to prepare the machine for the material removal processing that is performed in the next section of the program. There are both M-codes and G-codes in this section. The M-codes perform functions such as selecting the correct tool (for machines that have automatic tool changers), turning on the coolant flow (if available), and turning on the spindle in the

correct direction at a specified speed. The G-codes in this section involve positioning the tool in preparation for cutting using predominantly point-to-point control. Additionally, the units (inches or millimeters) and the type of coordinates used, either absolute or incremental, will be specified by G-codes.

Material removal contains mainly G-codes associated with actual tool moves executed by the CNC controller through continuous path control and interpolation. During the actual material removal process, the path and the speed along the path (feed rate) are very important to the quality of the finished product; hence, the many uses of continuous path control. However, when a tool is moved from one material removal operation to the next, it may have to be relocated. Consequently, some G-codes in the material removal portion of the program use point-to-point control.

The last section, system shutdown, contains the commands necessary to end the program. G-codes here use point-to-point control to move tooling away from a workpiece to facilitate workpiece removal. Also contained in this section are M-codes, intended to turn off miscellaneous functions activated during program setup. Additionally, an M-code to inform the CNC controller that the program is complete is listed.

The actual G- and M-codes and other letters listed in Figure 3-18 are discussed in great detail in Chapter 4.
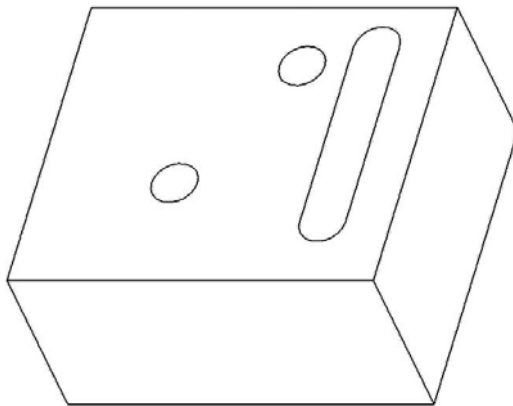

Figure 3-17 Part produced by Figure 3-18 program

```
N10 G90 G20
N20 M06 T22
N30 G00 X0 Y0 Z.1
N40 M03 S2000 F4
```
Program Setup

```
N50 G00 X1.5 Y1.25
N60 G01 Z-1.0
N70 G01 Z.1
N80 G00 X2.5 Y2.75
N90 G01 Z-1
N100 G00 Z0.1000
N110 G00 X3.2500 Y3.0000
N120 G01 Z-0.1250
N130 G01 X3.2500 Y0.7500
N140 G00 Z0.1000
N150 G00 X3.2500 Y3.0000
N160 G01 Z-0.2500
N170 G01 X3.2500 Y3.0000 Z-0.2500
N180 G01 X3.2500 Y0.7500
N190 G00 Z0.1000
N200 G00 X3.2500 Y3.0000
N210 G01 Z-0.3750
N220 G01 X3.2500 Y3.0000 Z-0.3750
N230 G01 X3.2500 Y0.7500
N240 G00 Z0.1000
N250 G00 X3.2500 Y3.0000
N260 G01 Z-0.5000
N270 G01 X3.2500 Y3.0000 Z-0.5000
N280 G01 X3.2500 Y0.7500
N290 G00 Z0.1000
N300 G00 X3.2500 Y3.0000
N310 G01 Z-0.6250
N320 G01 X3.2500 Y3.0000 Z-0.6250
N330 G01 X3.2500 Y0.7500
N340 G00 Z0.1000
N350 G00 X3.2500 Y3.0000
N360 G01 Z-0.7500
N370 G01 X3.2500 Y3.0000 Z-0.7500
N380 G01 X3.2500 Y0.7500
N390 G00 Z0.1000
N400 G00 X3.2500 Y3.0000
N410 G01 Z-0.8750
N420 G01 X3.2500 Y3.0000 Z-0.8750
N430 G01 X3.2500 Y0.7500
N440 G00 Z0.1000
N450 G00 X3.2500 Y3.0000
N460 G01 Z-1.0000
N470 G01 X3.2500 Y3.0000 Z-1.0000
N480 G01 X3.2500 Y0.7500
N490 G00 Z0.1000
```
Material
Removal

```
N500 G00 X0 Y0 Z1
N510 M05
N520 M30
```
System Shutdown

Figure 3-18 G-code program

3.3 Coordinate Systems and Reference Points

It has been discussed that the key aspect of CNC control is ability to control the position of the tool relative to, or in reference to, the position of the workpiece. Consequently, a CNC machine requires a method of specifying tool position in three-dimensional space. Additionally, this position must somehow reference the workpiece. Generally, this is accomplished with use of a machine coordinate system that defines the tool's position and a parallel workpiece coordinate system whose origin is PRZ point, specified in the machine coordinate system, which relates the tool position to the workpiece.

3.3.1 Machine Coordinate System

Refer to Figure 3-19 and recall from Section 3.1 how the Cartesian coordinate system is used to define the various directions in which a mill table can move. Each movement direction corresponds to an axis of the machine. In the case of the mill in Figure 3-19, the mill table and correspondingly the workpiece can move in three directions, orthogonal to one another and relative to the spindle and/or tool's position. Hence, this machine uses the Cartesian coordinate system, each of its axes corresponding to the x-, y-, or z-axis. For this particular example, the x direction corresponds to the longest axis movement (across the machine), the y direction aligns with motion in and out of the machine, and the z direction matches up with the spindle axis for vertical motion. This setup defines the equipment's machine coordinate system. Different equipment may have different axis configuration and thus different machine coordinate systems. Another factor that causes machine coordinate systems to vary among different machines is the industry standards used to label the axes. Typical industry standards specify that the spindle axis be labeled the z-axis and the longest travel axis be labeled the x-axis.

Figure 3-20 shows the machine coordinate system for a lathe. Only two axes are used, so following industry standards the coordinate system uses only the x and z axes.

Additional axes can be added, beyond x, y, and z, by inclining or rotating the tool relative to the workpiece. This is typically accomplished by adding a two-axis rotary attachment to a three-axis machine, as is shown in Figure 3-21(a). Rotational axes are typically labeled A and B. Figure 3-21(b) shows an additional configuration. These machines are called *five-axis machines*. Note the adherence to industry standard labeling.

The location of the origin of the machine coordinate system varies amongst machine configuration and manufacturer. It is the position where x, y, and z are all zero, and it is called the *machine reference zero* (*MRZ*). A possible location of the MRZ for a three-axis machine is determined at the point where each axis is fully retracted. This position is often transparent to the programmer but is used by the CNC controller to correlate defined positions of the workpiece coordinate system to the machine coordinate system.

Consider Figure 3-22, which is an engineering or product drawing of a workpiece to be milled. This workpiece has three features: two holes and a slot. Figure 3-23 shows the workpiece fixtured to the table of a three-axis mill. Note the location of the workpiece coordinate system and its origin, the program reference zero.
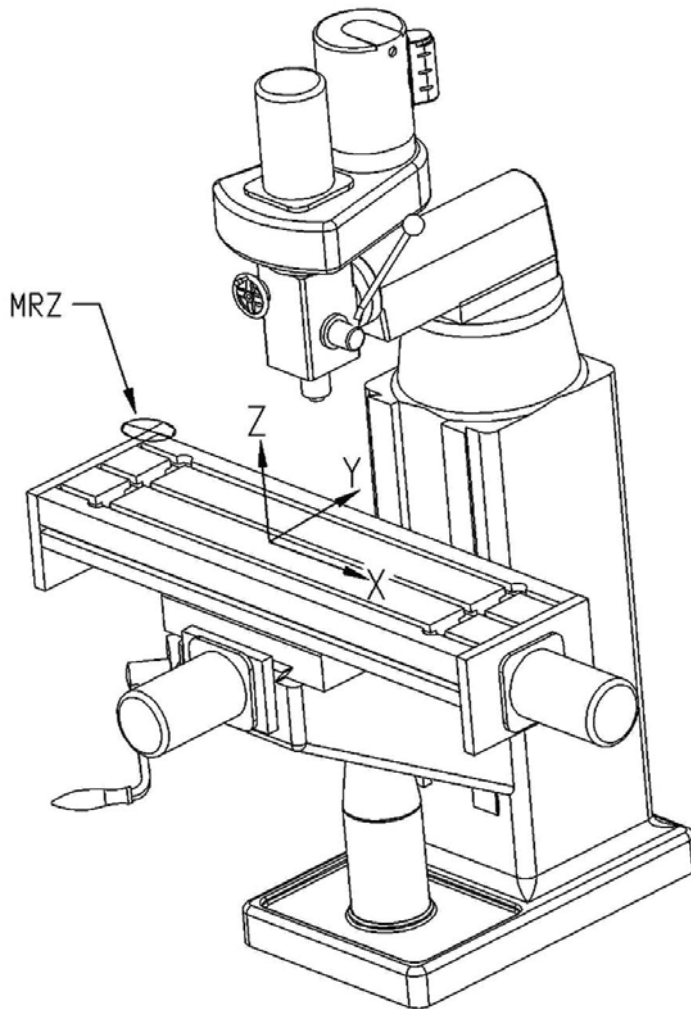

Figure 3-19 Coordinate system and origin location, called machine reference zero
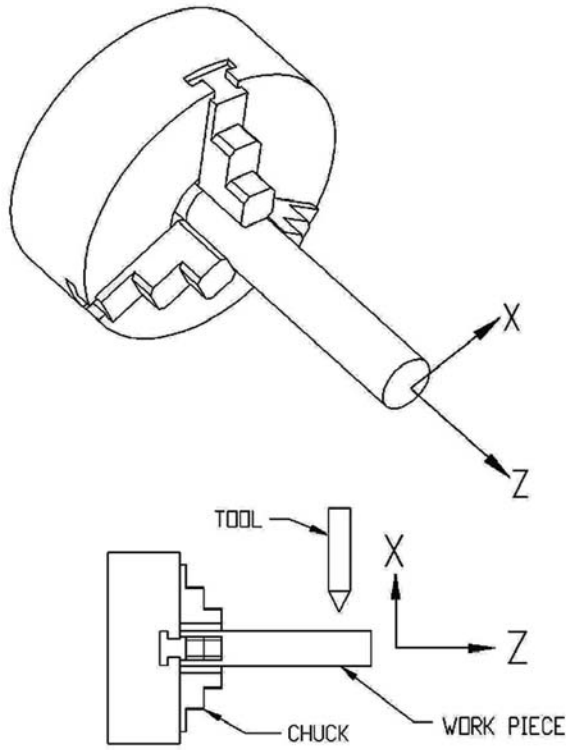
(MRZ) for three-axis mill
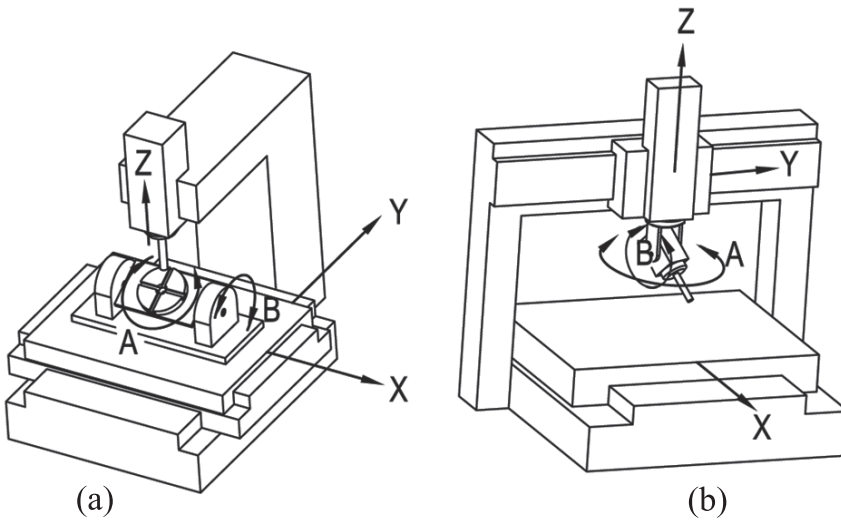
Figure 3-20 Coordinate system for a lathe

(a)   (b)
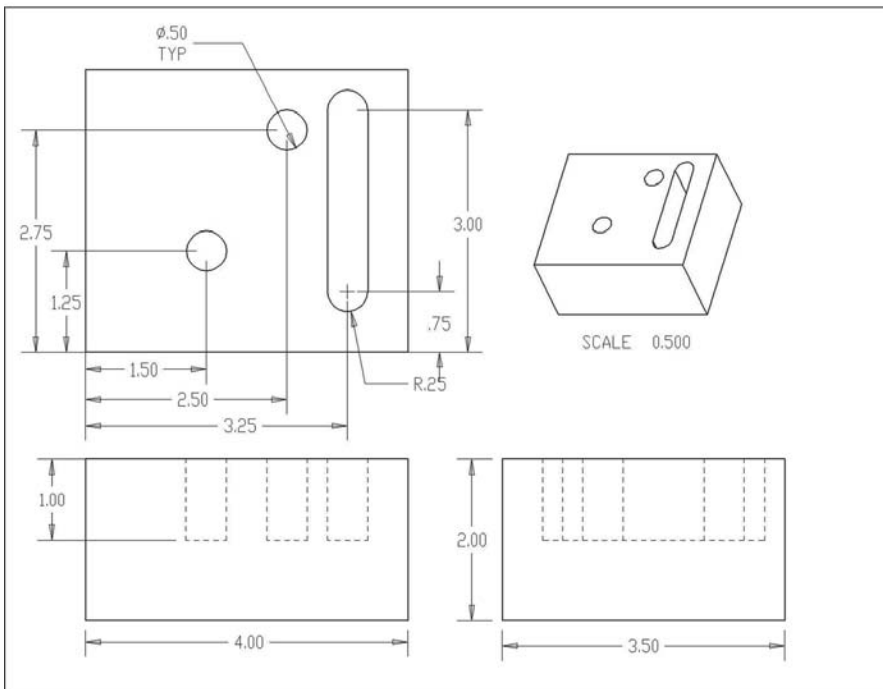
Figure 3-21 Five-axis machines

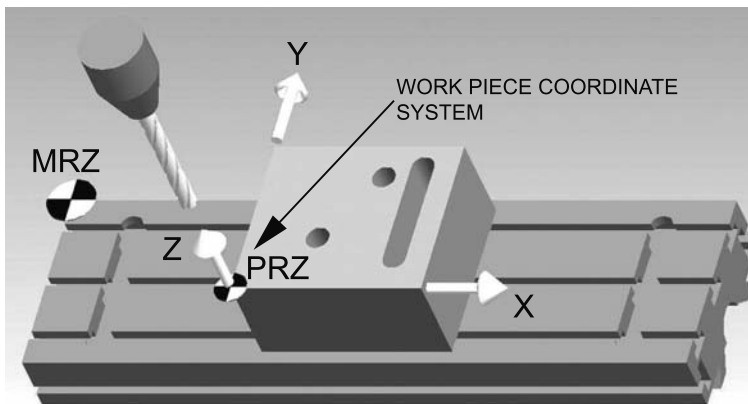*Figure 3-22 Milled workpiece product drawing*

Figure 3-23 Workpiece coordinate system and PRZ

3.3.2 Program Reference Zero (PRZ)

The PRZ point is the origin of an additional coordinate system (attached to the workpiece) that is parallel to the machine coordinate system. The establishment of the PRZ permits the programmer to easily relate the location of workpiece features to the machine coordinate system, thus allowing the correct positioning of the tool relative to the workpiece for feature production of the piece.

The standard location of the PRZ on a milled workpiece is the lower left corner and the top of the workpiece, as shown in Figure 3-23. This location choice is desirable so all features and corresponding tool locations are in quadrant I of the coordinate system, where both *x* and *y* are positive. Placing the PRZ on top of the workpiece is also convenient so that when the tool is actually removing material from the workpiece, the *z* value will be negative. Accordingly, when the *z* value is positive, the programmer knows the tool is clear of the workpiece. Figure 3-24 shows the PRZ of a turned part. Convention dictates that its location be along the spindle axis and at the far right face. The reason for this convention for turned parts is similar to that of milled workpieces; namely, positions with negative *z* values indicate material removal, as do moves directed in the negative *x* direction.
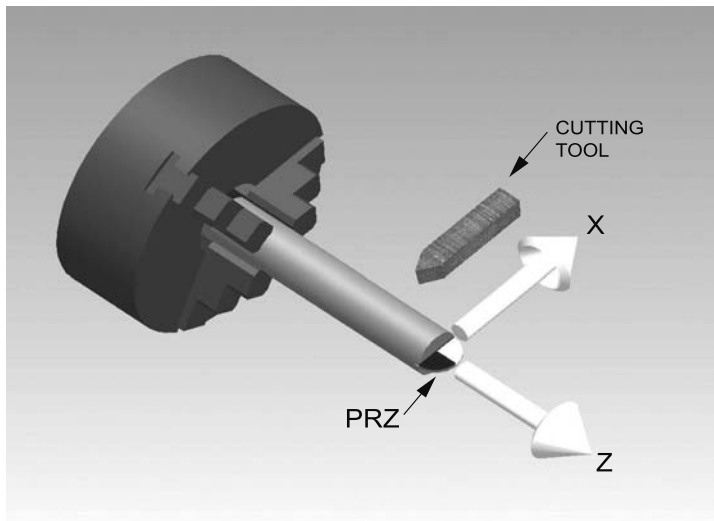

Figure 3-24 PRZ for a turned part

3.3.3 Absolute and Incremental Coordinates

The establishment of the workpiece coordinate system and its origin, the PRZ, provides a means to identify the location of workpiece features and correspondingly the path the tool will follow during the machining of the workpiece. The path the tool follows to produce the desired features of the workpiece—the *tool path*—is identified by a connected series of positions or points. The order in which the tool moves to each position, as decided by the programmer, defines the path. The ideal tool path is one in which the workpiece features are produced in the shortest amount of time.

Figure 3-25 shows the tool path to produce the workpiece previously identified in Figure 3-22. The tool path can be broken down into 11 individual tool moves to 9 positions as listed in Figure 3-26. The tool path starts and ends at position A. Each position along the path is identified with an *x*-, *y*-, and *z*-coordinate in the workpiece coordinate

system. These coordinates can be specified using either absolute or incremental coordinates.

Figure 3-25 Milled workpiece tool path

Move Number	Start Position		End Position
1	1	to	2
2	2	to	3
3	3	to	2
4	2	to	4
5	4	to	5
6	5	to	4
7	4	to	6
8	6	to	7
9	7	to	8
10	8	to	9
11	9	to	1

Figure 3-26 Tool path moves

Absolute coordinates define a position or point using absolute *x, y,* and *z* values relative to the *origin* (or PRZ) of the workpiece coordinate system. *Incremental coordinates*, however, define the *x, y,* and *z* values of a point along the path in a successive way, that is, relative to the *last point on the path.*

Consider Figure 3-27, which shows the tool path of Figure 3-25 viewed in only the (*x,y*)-coordinate system, with the origin at the PRZ. Positions C, E, G, and H are not shown because they possess the same *x* and *y* values as positions B, D, F, and I, respectively, differing in the *z* direction only. To express a position in absolute coordinates its location must be identified along each axis relative to the origin. Hence, position A has an *x*-coordinate of 0 and a *y*-coordinate of 0. To obtain position B, it is necessary the tool move from the origin 1.5 units in the positive *x* direction and 1.25 units in the positive *y* direction. Thus, position B has an *x*-coordinate of 1.5 and a *y*-coordinate of 1.25. The format for specifying position coordinates is to list the position letter (or number, if numbers are used to identify positions, though these can be confused with the coordinates themselves), followed by the *x*- and *y*- coordinates, encased in parentheses and separated by a comma. For example, the absolute coordinates for position A and B are (0, 0) and (1.5, 1.25), respectively. The absolute coordinates for each of the positions of Figure 3-27 are listed in Figure 3-28.
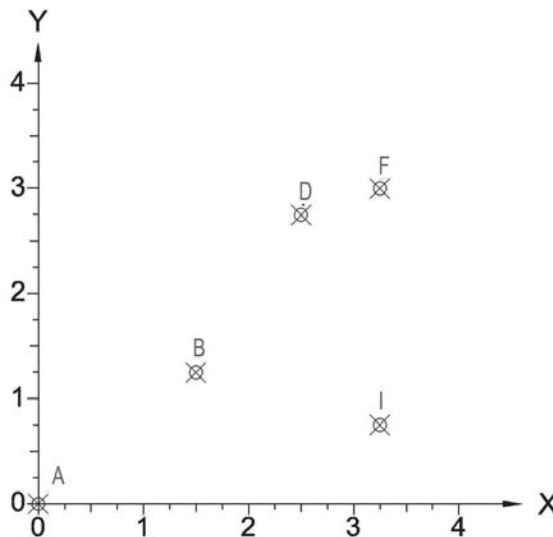

Figure 3-27 Tool path positions shown in x,y-plane of workpiece coordinate system

Incremental coordinates are specified using a different origin for each position. Unlike absolute coordinates, which reference the origin of the workpiece coordinate system, incremental coordinates are defined in reference to the last position along the tool

path. However, if the origin *is* the first position along the path, it is then the reference of the workpiece coordinate system.

Again consider Figure 3-27. In incremental coordinates, position A is (0, 0). Position B is determined in reference to position A; hence, its location is (1.5, 1.25). Each of these specifications is identical to the absolute coordinate location. This will always be true of the first position along the tool path, and often for the second position if the first position is at the origin of the workpiece coordinate system. As defined by the tool path, position D's location is determined in reference to position B. Therefore, to move to position D incrementally from position B, it is necessary to move the tool 1 unit in the positive x direction and 1.5 units in the positive y direction. Therefore, position D is defined in incremental coordinates as (1, 1.5). The coordinates of position F, as dictated by the tool path, are determined in reference to position D. Its position is (0.75, 0.25). Again, position I, as dictated by the tool path, is determined in reference to position F and is specified as (0, −2.25). The y value is negative because it is necessary to move in the negative y direction 1.5 units to move to position I from position F. Therefore, even though the positions are defined in quadrant I, negative x and negative y values are possible with incremental coordinates. The incremental coordinates of each position are listed in Figure 3-29.

Position No.	X	Y
A	0.00	0.00
B	1.50	1.25
D	2.50	2.75
F	3.25	3.00
I	3.25	0.75

Figure 3-28 Absolute coordinates of Figure 3-27 positions

A common mistake students make with incremental coordinates occurs when positions along the tool path fall in quadrants other than—and in addition to—quadrant I. Figure 3-30 shows such a tool path, with the absolute coordinates listed next to each position. To determine the incremental coordinates of a position relative to one located in another coordinate, one must consider the total distance traveled along the axis. For example, in incremental coordinates, position B is defined relative to position A as (−2, −0.375). Figure 3-31 shows the incremental coordinates of each position.

Position No.	X	Y
A	0.00	0.00
B	1.50	1.25
D	1.00	1.50
F	0.75	0.25
I	0.00	-2.25

Figure 3-29 Incremental coordinates of Figure 3-27 positions

Figure 3-30 Tool path in four quadrants

Position No.	X	Y
A	0.750	0.750
B	-2.000	-0.375
C	0.000	-1.125
D	1.750	-0.250

Figure 3-31 Incremental coordinates of Figure 3-30 positions

3.4 The Ten Steps of CNC Programming

In addition to introducing the reader to CNC technology and defining some important terminology, the last few sections also provide some insight into how CNC equipment is programmed. The overall process is rather involved, but as we shall see in the next section, advantages of producing parts via CNC equipment are well worth the effort. CNC programming can be broken into the following ten steps:

Evaluate the product drawing. The product drawing details the features to be produced by the program. Thus, evaluation of the product drawing is a critical first step. During this step each feature's location and associated data are identified and evaluated. Dimensioning schemes and drawing layout strongly influence the choice of fixturing methods as well as the orientation and location of the workpiece coordinate system. Additionally, a workpiece may require several different programs on various machines before it reaches the form defined by the product drawing. Thus, it is necessary to know the stage of manufacture the product is in when it arrives at the CNC machine for which the program is being written. The workpiece may already contain features that will also influence the fixturing method and the location of the PRZ.

Select the machine, required tooling and fixturing to produce the desired features. During this step, the machine for which the program is being written is selected. Based on its machine coordinate system and the information gathered from step 1, a fixturing method and required tooling are also identified.

Determine and calculate the appropriate cutting parameters. The quality and speed of the material removal process, dictated by the *cutting parameters*, are dependent on the tooling selected and the workpiece material. The cutting parameters are:

- cutting speed
- feed rate
- depth of cut

These parameters define how the tool moves into the material for each tooling pass. There are industry standard tables that list these parameters for specific tools, operations, and workpiece material. Additionally, calculations are necessary to convert parameters into a more usable form for the CNC controller.

Develop the program tool path. This involves developing the machining sequence and identifying the path the tool should follow to produce the desired features. This step is also called "developing the process flow." The locations of the features are identified, and the path to each of these locations, along with the path to produce the feature, is then mapped.

Determine program coordinates. The coordinates of each position along the tool path are identified and/or calculated based on tool path and workpiece coordinate system orientation. The use of either absolute or incremental coordinates is based on the requirements of the specific machine and/or the discretion of the programmer.

Write the program of instructions. This step entails converting the cutting parameters, tool path, and program coordinates into the language of the CNC controller.

Verify the program using CNC simulation software. Numerous simulation software programs have been developed to graphically verify the program of instructions prior to running the part on the machine with an actual workpiece. Because of the complexity of generating the program of instruction, simulation software was developed to identify programming errors prior to actual part production. Additionally, it is much easier to edit the program of instruction at this stage of development. Figure 3-32 is a screen shot of a sample simulation software program.
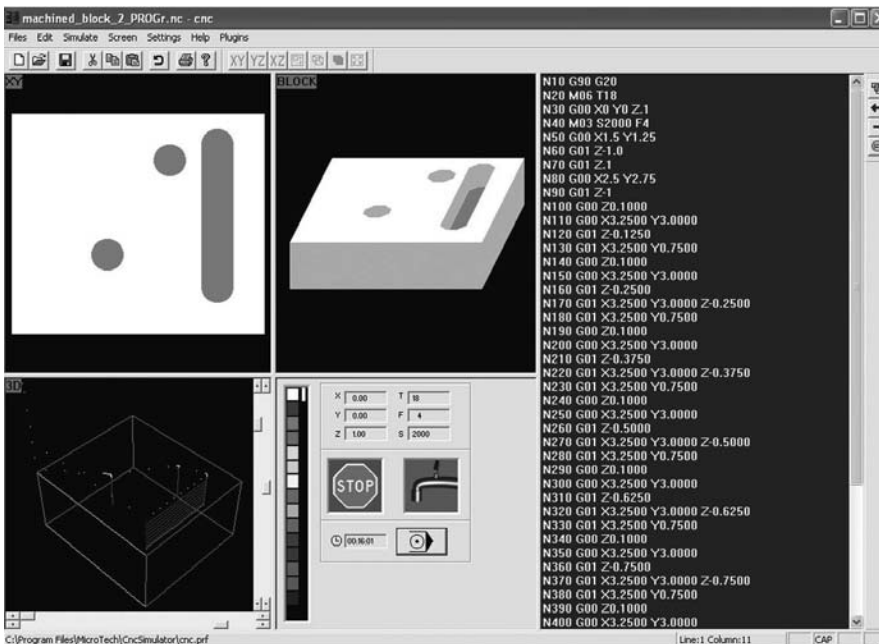

Figure 3-32 Simulation software example

Prepare setup sheets and tool lists. A setup sheet provides instruction to the person who sets up the machine for the program. The instructions explain how the workpiece is to be fixtured for the given program. It also identifies the location of the PRZ on the fixtured workpiece. Tool lists identify the tooling used by the program.

Verify the program on the actual machine. This step involves trying out the program on the machine for which the program was written. It entails setting up a sample workpiece per the setup sheets, loading the tooling per the tool lists, loading the program, and finally executing the program. This is the final check before the program is released from development. Often, the sample workpiece is made of a cheaper, easier to machine material, such as machinable wax. Thus, if there are mistakes in the program or problems with the fixturing, a less expensive material is scrapped.

Run the program. When this step is reached the program will be error-free and ready to run an actual workpiece. After the actual piece is run it is inspected and all dimensions verified against the product drawing. If all dimensions are acceptable, the program is released from development.

Steps 1 through 6 above are discussed at length in Chapter 4.

3.5 Advantages and Disadvantages of CNC Technology

CNC technology applied to the machining process offers numerous benefits over traditional manual machining. However, there is a price to pay for all the returns CNC technology yields. In this section we discuss the advantages and disadvantages of CNC technology, along with the most appropriate situations for utilizing CNC.

3.5.1 Advantages

As with most types of automation, the reduced level of human participation in the process, and thereby a reduction in labor costs, is one of the primary advantages. However, with CNC technology the advantages go well beyond mere labor savings. A list of six primary advantages with detailed explanations is given below. Note that in many cases, advantages are mutually inclusive.

1. *Increased productivity*. To review, "productivity" is the amount of output (parts produced)—i.e., obtained from a process—divided by the amount of input (labor, raw materials, time). Typically, both output and input are expressed in monetary terms. Hence, to improve productivity either more product needs to be produced (i.e., increased output) or less labor used, as well as less raw material or time (i.e., decreased input). Typically, with CNC technology the gain in productivity comes from a reduction in labor and/or raw material usage and through time savings.

Reduced labor and time savings from CNC are realized due to a reduction in nonproductive or noncutting time. Operations that were previously performed on separate manual machines can be combined into a single program on one CNC machine. This yields fewer setups, less time to setup for the simpler CNC fixturing, and reduced work handling. Additionally, time is saved with the use of automatic tool changes. CNC machines make use of tool turrets or tool belts. These turrets hold numerous cutting tools that are automatically loaded and unloaded during program execution by the CNC controller. A reduction in raw material usage is achieved with improved quality and a corresponding scrap reduction.

2. *Improved quality*. Processes done with a CNC controller applied to a tool relative to a workpiece are far more accurate and repeatable those that are done through manual processes. Use of CNC drastically reduces and/or eliminates part-to-part *and* batch-to-batch variation. The net result is typically reduced inspection requirements and lower scrap rates. The lower scrap rates mean that less raw material is required, which improves productivity.

3. *The ability to manufacture complex shapes*. The advent of CNC technology was the direct result of the great difficulty encountered in the manufacture of complex shapes. In some cases the manufacture was impossible. Recall how John Parson's first NC machine was developed to machine more accurate helicopter blade templates. Hence, the ability to manufacture complex shapes is still a primary advantage of CNC technology.

4. *Manufacturing cost structure improvements*. The reduction in the number of setups and the time to perform the setups by utilizing CNC technology yields tangible manufacturing cost improvements as well. Manufacturing cost improvements are also made with the reduction in floor space from the replacement of multiple manual machines with one or two CNC machines. Shorter manufacturing lead times, reduced part inventory, reduced work in process, and less floor space greatly impact the cost structure of the manufacturing firm.

5. *Reduced operator skill level*. As stated at the beginning of this chapter, in traditional "manual" machining a machinist (or operator) decides and directs the motion of the tool relative to the workpiece to create the desired shape of the finished part. The machinist's decisions were based on the part print and his or her experience and training. Consequently, operators of manual machines had to have intimate knowledge of machining processes, setup methods, and tooling. With CNC technology, the process planning, fixture design, and tooling selection are made well in advance by the programmer and/or manufacturing engineer. Additionally, the program directs the path of the tool. Hence, all the operator is required to do is simply load the workpiece, start the program, and unload the workpiece. An operator with less skill usually receives a lower wage, which consequently yields manufacturing cost structure and productivity improvements.

6. *Easily accommodate product design changes*. A high production manual machining process often involves moving the workpiece across numerous manual machines that use custom tooling specifically designed for that particular machine and workpiece. Hence, if the design of the finished product were changed, the custom tooling might have to be changed in addition to the manufacturing process. For significant design changes this is a very costly and time-consuming process. CNC technology, on the other hand, uses relatively simple fixturing and the program of instructions controls the processing. Therefore, significant design changes might only impact the program, which can be changed rapidly with little impact on the flow of the manufacturing process. With today's rapidly changing products and the impact of lean manufacturing concepts this is a significant advantage.

3.5.2 Disadvantages

Although the advantages of CNC technology are great, there are still some significant disadvantages that need to be considered. The most significant of these is how switching to CNC technology impacts manufacturing cost structure and manufacturing productivity. The three primary disadvantages of CNC technology are listed below.

1. *Higher investment cost*. CNC machines are quite expensive compared to manual machines. This is due to the addition of the electrical and computer technology that controls the position of the tool relative to the workpiece. This higher initial cost has a dramatic impact on the cost structure of the manufacturing firm. Thus, machine use can become an issue. Additionally, the higher purchase price impacts these machines' hourly operation rate, which can have a detrimental effect on machine productivity if not compensated for with other productivity improvements. Refer to Chapter 2 for more information on justifying automation.

2. *Higher maintenance cost*. The addition of electrical and computer technology make repair and upkeep of CNC machines more involved and costly. Reliability and mean-time-between-repair data should be carefully considered in the selection of a machine. Also, maintenance department skill level requirements change with the addition of a CNC machine and automated work cells. Stronger electrical and computer skills are required. Personnel with these skills often demand a higher pay rate, which also impacts maintenance costs.

3. *Addition of specialized personnel*. Adding one or more CNC machines to a manufacturing facility requires the addition of specialized personnel to develop the program of instructions. Although most programming can be performed at the machine through the user interface, the machine is "down" or "offline" during this process—hence, not producing. This situation essentially amounts to an extended setup. Thus, in most high production manufacturing firms a specialist will be employed to develop the program of instructions ahead of time. This specialist is called a "part programmer" or "CAM programmer" or simply a "programmer." Note that CAM stands for computer-aided manufacturing. Programmers are typically skilled in the manufacturing process in question, in its fixturing, tooling, blueprint reading, two- and three-dimensional CAD, G-code language, and in various other programming and simulation software programs. The addition of a programmer, or a whole department of programmers in larger firms, also impacts the flow of work through the facility, which has a direct effect on the lead-time of the product.

3.6 When to Use CNC Technology

For the addition of CNC equipment to be justified, the advantages of improved productivity, quality, production capability, and other advantages discussed previously have to exceed the disadvantages of higher initial investment, higher maintenance costs, and addition of personnel. Performing a productivity analysis is a good way to determine if CNC technology is justified, as discussed in Chapter 2. (This is true for most automation.) However, there are several scenarios that often indicate a conversion to CNC technology is warranted.

1. *Batch production with small to medium lot sizes*. The batch production in small to medium lot sizes indicates that programmable automation, which CNC technology

provides, is a good choice. Conversely, if very large batch sizes are required, fixed automation may be a better choice.

Batch production also indicates that parts are produced periodically. With CNC technology, once a program has been created and verified, conversion to a new batch (or order) can be accomplished much faster than with manual machines.

2. *Multiple separate machining operations*. Multiple machining operations require multiple setups. If these operations can be combined into one CNC operation only one setup will be required. Additionally, multiple operations may also indicate multiple machines. Thus, combining operations may also reduce the number of machines required.

3. *Complex part geometry*. As previously stated, in some situations CNC technology may be the only viable option for manufacturing products with exceptionally complex geometry.

3.7 Summary

Computer numerical control (CNC) technology is the automation of traditional manual machining processes with electrical and computer technology. Through the use of a program of instructions, a computer controller directs the motion of the tool relative to a workpiece, removing material to create the desired shape of the finished part. The program of instructions also contains commands to control all essential machine functions and is written in a language that the CNC controller can understand and is typically called a "part program" or simply a "program."

The components of a CNC system include some type of processing equipment/ machine tool, a drive mechanism/positioning system, a CNC controller, and the program of instructions.

CNC technology has been applied extensively to equipment used in the machining industry. However, the technology can be applied to essentially any type of processing equipment that needs to move a tool relative to a workpiece.

The directions in which the tool is moved relative to the workpiece are given in the Cartesian coordinate system. Coordinate directions correspond to the machine axes. Movement along the axes of the machine is accomplished with mechanically guided, high precision linear bearings called slides or ways and a lead screw. A carriage or table is moved along the slide (or axis) by the lead screw. The lead screw transforms rotational motion from an electric motor into linear movement along an axis.

The CNC controller directs all machine functions, including simultaneous axis motion, and acts as the operator interface for manual operations related to program and machine setup. The controller provides auxiliary control over discrete functions and simultaneous motion control of the machine axes. Two types of motion control are point-to-point control and continuous path control. Point-to-point control involves motion in which the choice of path that is taken to the desired location is of little importance; continuous path control involves motion in which the path taken is of prime importance.

The controller generates a continuous path by generating a series of intermediate positions along the path between the starting and ending positions. This is called interpolation. Linear and circle interpolations are the two most commonly used types of interpolation.

The program of instructions is a sequential list of alphanumerically coded machine instructions, that is, letters combined with numbers. The program of instructions can be broken down into program setup, material removal, and system shutdown sections.

A CNC machine controls the tool's position in three-dimensional space relative to the workpiece. This is accomplished with the use of a machine coordinate system, which defines the tool's position, and a parallel workpiece coordinate system, whose origin is the program reference point or program reference zero (PRZ), as specified in the machine coordinate system. The location of the origin in the machine coordinate system is the machine reference zero (MRZ). The MRZ varies from machine to machine. Standard location of the PRZ on a milled workpiece is the lower left corner of the workpiece top. The PRZ of a turned part is located along the spindle axis on the far right face of the workpiece.

Positions along the tool path can be specified using either absolute or incremental coordinates. Absolute coordinates define tool path points relative to the PRZ. Incremental coordinates define tool path points relative to the previous point along the tool path.

The CNC programming process starts with an evaluation of the product drawing followed by the selection of the machine, tooling, and fixturing required to produce the part. This is followed by calculation of the cutting parameters and development of the tool path. Once the tool path is determined the program coordinates can be calculated and the program written. The program is then simulated, setup sheets and tool lists prepared, and the program verified on the actual machine. Finally, the program will be run and released from development.

CNC technology offers the advantages of increased productivity, improved quality, capacity for machining complex shapes, reduction of required operator skill level, and the easy accommodation of design changes. However, the technology generally carries a higher investment and maintenance costs than traditional manual machines. Additionally, specialized personnel are required to develop CNC programs.

Use of CNC technology is particularly beneficial in batch production systems. It can also be used to combine multiple separate machining operations into one operation. In some cases, CNC technology may provide the only viable method of economical production.

3.8 Key Words

absolute coordinates	machine coordinate system
auxiliary control	material removal section
auxiliary functions	machine reference zero (MRZ)
	M-codes
CAM programming	mechanical technology
Cartesian coordinate system	mill table
circle interpolation	motion control
closed loop control system	motor drive unit
computer numerical control CNC	
computer technology	open loop control system
continuous control	operator interface
continuous path control	parabolic/cubic interpolation
contouring cut	part program
cutting parameters	point-to-point control
cutting speed	preparatory functions
depth of cut	program of instructions
discrete control	program reference zero (PRZ)
	program setup section
dovetail slides	programmer
electrical technology	
encoder	servomotors
feed rate	setup
fixturing	setup sheets
G-code	slides
G-code program	stepper motor
helical interpolation	straight cut
incremental coordinates	system shutdown section
John Parsons	tachometer
lead screw	tool lists
lead-time	tool path
linear interpolation	
machine axis	ways
machine controller	workpiece coordinate system

3.9 Review Questions

1. Define CNC technology.
2. What types of processing equipment are candidates for CNC technology?
3. What coordinate axes are used on a lathe?
4. By convention, which axis is in line with the spindle axis?
5. Which type of coordinate uses the last position as the reference point?

6. Define the machine coordinate system and the location of its origin.
7. Define the workpiece coordinate system and the location of its origin.
8. Define the various functions of a CNC controller.
9. What is the difference between discrete and continuous control?
10. Compare and contrast closed loop and open loop control systems.
11. Describe when use of point-to-point control is more appropriate than continuous path control.
12. List and describe interpolation methods.
13. List and describe the purpose of the various sections of a CNC program. What type of axis control is used in each section?
14. List the absolute coordinates of each of the points shown in Figure 3-33.
15. List the incremental coordinates of each of the points shown in Figure 3-33.
16. List and discuss the 10 steps of the CNC programming process.
17. List the advantages and disadvantages of CNC technology.
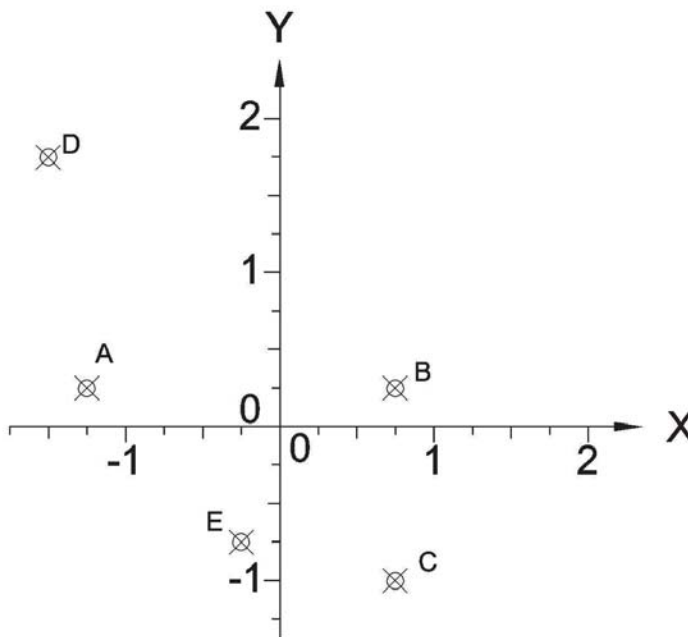18. Discuss when the use of CNC technology might be warranted.

Figure 3-33

3.10 Bibliography

1. Groover, M.P. (2001). *Automation, Production Systems and Computer-Integrated Manufacturing,* 2nd ed. Prentice Hall, Upper Saddle River, New Jersey.
2. Nanfara, F., Uccello, T., and Murphy, D. (2002). *The CNC Workshop.* Schroff Development Corporation, Mission, Kansas.
3. Gibbs, D. and Crandell, T.M. (1991). *An Introduction to CNC Machining and Programming.* Industrial Press, Inc., New York.
4. Stenerson, J. and Curran, K. (2007). *Computer Numerical Control*, 3rd ed. Prentice-Hall, Upper Saddle River, New Jersey.
5. Chang, T.C., Wysk, R.A., and Wang, H.P. (2005). *Computer-Aided Manufacturing*, 3rd ed. Prentice-Hall, Upper Saddle River, New Jersey.
6. Valentino, J.V. and Goldberg, J. (2003). *Introduction to Computer Numerical Control (CNC),* 3rd ed. Prentice-Hall, Upper Saddle River, New Jersey.

Chapter 4

CNC Programming

Contents

Objective

The objective of this chapter is to describe in detail the CNC programming process, which uses a programming language called "word address format," or more typically, "G-code."

115

4.1 Overview of CNC Programming

As we discussed in the previous chapter, there are many advantages of CNC technology over manual manufacturing processes. However, in order to realize these benefits, manufacturers must use CNC technology efficiently and effectively. This can only be accomplished with an accurate *program of instructions.* Over the years various methods have been developed to convert the program of instructions into machine instructions that the CNC controller can understand.

Developing the program of instructions and formatting them into the language of the CNC machine—i.e., programming—is the sole responsibility of the CNC programmer. A CNC programmer must be well-versed in the programming language of the various machines and have intimate knowledge of manufacturing processes. There are a number of ways a program of instructions is formatted so the CNC machine can understand and input the instruction into the CNC controller. The simplest method is *manual data input* (MDI), by which the machine operator inputs the program directly into the machine control unit using some type of electronic interface, device such as pushbutton keypad and/or a keyboard. For relatively simple operations this is a viable method; however, for a more complex program of instructions this method is time-consuming and cumbersome.

Conversational programming is the more modern, automated form of MDI. With this method the machine operator or programmer is prompted for specific information about part geometry, part material, tooling information, and tool path. Controllers who use this method typically have built-in cycles or "canned" routines to further simplify the programming process. Additionally, the program can often be displayed graphically on the controller's interface, which facilitates easier program editing. However, with this kind of programming, the operator cannot program and machine at the same time—a major disadvantage if optimal machine utilization is a prime concern. Additionally, conversational programming interface and methods vary among manufacturers and machines. Hence, operators and/or programmers might need to be proficient with several types of interfaces.

The disadvantages of conversational programming are overcome with manual part programming. This is a standardized format in which the programmer places a program into a form that the CNC controller can understand. Additionally, the programming can be performed "offline," away from the machine, so as not to interfere with production operations. The most common manual part programming language is word address format programming, which makes use of letters that define letter address commands, followed by a sequence of numbers. The letter and number combination form a word, which instructs the controller to perform a specific action. As mentioned in Chapter 3, this method is typically called "G-code programming" because of the repeated appearance of the G prefix in the word addresses, as evident in the program code shown in Figure 4-0

```
N10  G90 G70
N20  T17 M06
N30  G00 X0 Y0 Z1
N40  F90 S4000 M03
N50  G00 Z0.1
N60  G00 X1.5 Y1.0
N70  G01 Z-.375
N80  X1.6725
N90  G02 X1.6725 Y1.0 I-.1725 J0
N100  G01 X1.845
N110  G02 X1.845 Y1.0 I-.3.45 J0
N120  G01 X2.0303
N130  G01 X1.5 Y1.5303
N140  G01 X.9697 Y1.0
N150  G01 X1.5 Y.4697
N160  G01 X2.0303 Y1.0
N170  G00 Z0.1
N180  G00 X.5 Y.25
N190  G01 Z-.375
N200  G00 Z0.1
N210  G00 Y1.75
N220  G01 Z-.375
N230  G00 Z0.1
N240  G00 X2.5
N250  G01 Z-.375
N260  G00 Z0.1
N270  G00 Y.25
N280  G01 Z-.375
N290  G00 Z0.1
N300  G00 X0 Y0 Z1
N310  M05
N320  M30
```
.

Figure 4-0 Sample G-code program

    As the reader will see, the task of developing a word address format program is an
intricate and taxing process. Consequently, helpful computer-assisted methods have been
developed. The main aids are either language-based or graphic-based computer programs.
They were developed to simplify and expedite the conversion of the tool path to the word
address format. The most advanced aids are in the form of *computer-assisted
manufacturing* (*CAM*) programs. With CAM the tool paths are generated directly from the
part geometry, typically in three-dimensional solid model form. The CAM tool path is
then converted to the word address format for a specific machine by another software
program called a *postprocessor.* Postprocessors are machine-specific, as most machines

are not totally compliant with the ANSI/EIA RS-274-D standard. Hence, postprocessors are designed to bridge the gap between the standard and the specific machine.

The objective of this chapter is to teach the word address format programming language (G-code). In addition to learning how the code is assembled and formatted, the reader will gain knowledge of typical milling and turning manufacturing operations. Additionally, programming will be covered in great detail, and two program development examples are given.

4.1.1 Review of the Ten CNC Programming Process Steps

Of the ten steps of the CNC programming introduced in Chapter 3, steps 1 through 6 are discussed here and step 7 is covered in Chapter 5. (Steps 8, 9, and 10 are self-explanatory and not addressed further.) The steps, again, are:

1. Evaluate product drawing.
2. Select machine, required tooling, and fixturing to produce the desired features.
3. Determine and calculate appropriate cutting parameters.
4. Develop program tool path.
5. Determine program coordinates.
6. Write program of instructions.
7. Verify program using CNC simulation software.
8. Prepare setup sheets and tool lists.
9. Verify program on the actual machine.
10. Run program.

It is essential to the development of an efficient, effective program that a drawing be evaluated properly. Consider the product drawing in Figure 4-1.

A sample of questions a programmer might ask when evaluating the drawing are:

- How many operations are needed for the production of the item?
- Which CNC machine(s) is (are) most suitable to production?
- How many separate programs are needed?
- How will the product be fixtured?
- Where should the program reference zero (PRZ) be located?
- If multiple programs are to be used, which data are critical?
- Which cutting tools should be used?
- How many tool changes are required?
- What cutting speeds, feeds, and depths of cut are appropriate for the material, tool, and operation to be performed?
- Which features should be machined first? Second? etc.
- What path should the tool follow during actual material removal?
- What is path should the tool follow moving between features?

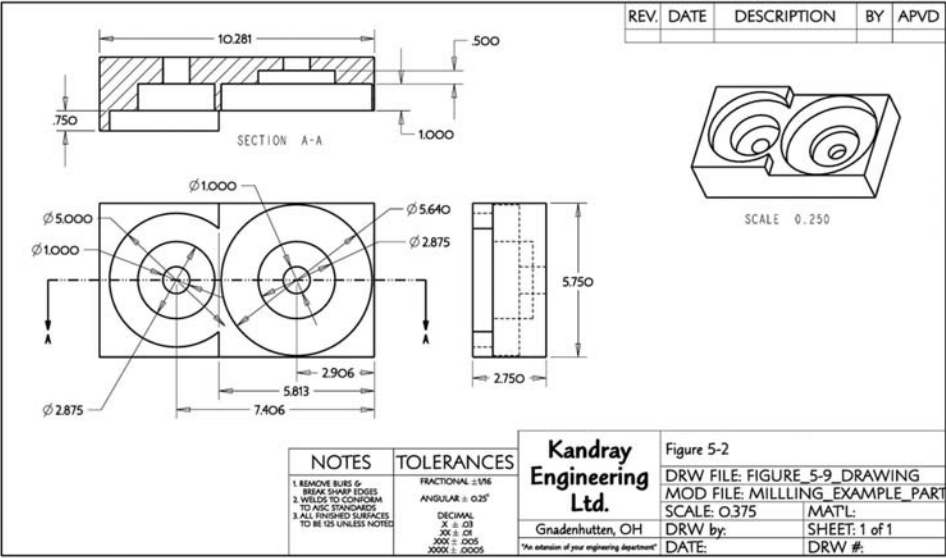Through such inquiry a programmer delves deeper into a program's contents and requirements.

Figure 4-1 Sample product drawing

4.1.2 Program Content

Recall from Chapter 3 that program code is organized into three sections: program setup, material removal, and system shutdown. Program setup may include:

- Coordinate system information, such as type of coordinates and PRZ information
- Type of units—inches or millimeters
- The plane that serves as default plane for circle interpolation operations; i.e., (*x,y*), (*y,z*), or (*x,z*)
- An instruction to change the tool and which tool to load
- Instructions to turn the part on the spindle; direction and speed of the spin
- Instructions to turn on powered clamping
- Instructions to turn on coolant flow
- Instructions for advanced programming functions.

Program setup functions (steps 1, 2, and 3) are coded as either *preparatory* or *auxiliary* (alternately, *miscellaneous*), shown in Figure 4-0 as G-codes and M-codes, respectively.

The vast majority of codes contained in the material removal section are of preparatory functions that direct the tool along the tool path (steps 4 and 5 of the CNC

programming process). Preparatory codes instruct the CNC controller to use either continuous path control with interpolation or point-to-point control to move the tool relative to the workpiece.

The system shutdown section of the program lists the instructions to complete the program and prepare the finished workpiece for removal. Hence, this program section contains auxiliary functions that turn off or disengage any function activated in the program setup section, including coolant turnoff, spindle shutdown, and release of any powered clamps. Additionally, point-to-point control preparatory codes may be listed that move the tool away from the workpiece, and an auxiliary command to inform the CNC controller that the program is complete must also be listed.

4.2 Program Code

The word address format, as dictated by ANSI/EIA RS-274-D and ISO 6983 standards, makes use of letter addresses combined with numbers to form word addresses. Word addresses are commands that instruct the CNC controller to perform some action. In some cases a word is a complete command.

Words are arranged into a block that represents one complete machine instruction. A series of blocks are assembled into a program. This format was standardized in 1980 as ANSI/EIA RS-274-D. Figure 4-2 shows how letter addresses are combined into words and assembled with other words to form a block of instructions. This particular block instructs the CNC controller to perform a linear interpolation move at a speed of 4 to position (1, 1.5) (assuming absolute coordinates) of the workpiece coordinate system.
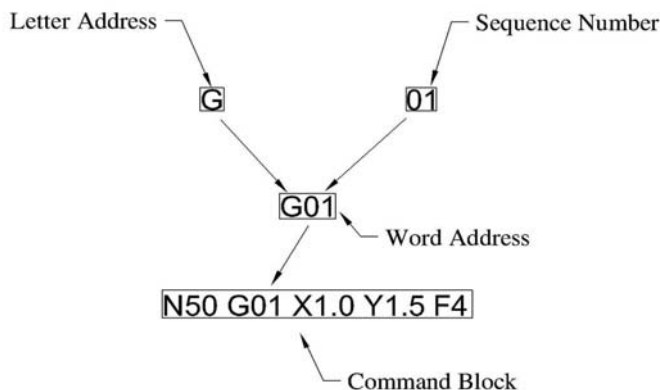

Figure 4-2 Combining letter addresses into words and command blocks

Consider the word M02. This word instructs the CNC controller to stop the machine. It is used to indicate the end of a program. Often, several words are required to instruct the CNC controller; when this is the case, multiple words are arranged into command blocks to form a complete instruction. Consider the following command block:

N10 G00 X1.0 Y2.0 Z-0.75

This command block is made up of five word addresses, which are executed in a specific order or sequence. The *sequence number* is always the first word in the command block. It identifies the kind of command and indicates when it is to be executed. The second word address here informs the CNC controller to move the tool relative to the workpiece to a specific location, but does not specify its destination; the next three word addresses do this. All five words are needed to provide a complete instruction. However, command blocks are not limited to one command. Several commands may be issued in one command block, as we will see.

To create the desired features in a workpiece, numerous CNC controller instructions and, hence, numerous command blocks are required. The next several sections explore letter addresses, words, and command blocks in detail.

4.2.1 Letter Addresses

Figure 4-3 is a listing and brief description of letter addresses as specified by the ANSI/EIA RS-274-D and ISO 6983 standards. The type of word address formed by the letter address is listed along with more specific information. Note that some letter addresses have multiple definitions. The applicable definition depends on the CNC controller and the type of machine it is controlling. The machine manual should always be consulted regarding letter address or word address definitions. The more common letter addresses in Figure 4-3 are shown in boldface.

4.2.2 Word Addresses

A word address is a combination of letter addresses with numeric values, such as G01. There are two main types of word addresses: *dimension words* and *non-dimension words*. Dimension words specify in either absolute or incremental coordinates the *destination* of the tool center point. For example, X1.0 is a dimension word. Dimension words can be given in linear or angular coordinates. Linear coordinates are expressed in decimal inches or millimeters; angular coordinates are expressed in decimal degrees or decimal parts of a revolution. Figure 4-3 includes letter addresses that form dimension words.

Non-dimension words (also shown Figure 4-3) include tool functions, feed functions, preparatory functions, interpolation parameters, miscellaneous functions, sequence numbers, and the spindle speed function. For example, M01 is a non-dimension word. Non-dimension words provide command instructions and other corresponding parameters

Letter Address	Type of Word	Description
A	Dimension word	Specifies angular dimension about the X-axis
B	Dimension word	Specifies angular dimension about the Y-axis
C	Dimension word	Specifies angular dimension about the Z-axis
D	Dimension word	Specifies angular units about a special axis
	Feed function	Third axis feed function
	Tool function	**Used to specifiy cutter compensation information**
E	Dimension word	Specifies angular units about a special axis
	Feed function	Second axis feed function
F	**Feed function**	**Primary or first axis feed function used to specify the feed rate for preparatory interpolation codes.**
G	**Prepartory function**	**Specifies information associated with actual tool moves**
H	Unassigned	
I	**Interpolation parameter**	**Used with circle interpolation to specify incremental location of arc center from start point in X direction**
	Interpolation parameter	Thread lead parallel to X
J	**Interpolation parameter**	**Used with circle interpolation to specify incremental location of arc center from start point in Y direction**
	Interpolation parameter	Thread lead parallel to Y
K	**Interpolation parameter**	**Used with circle interpolation to specify incremental location of arc center from start point in Z direction**
	Interpolation parameter	Thread lead parallel to Z
L	Unassigned	
M	**Miscellaneous function**	**Specifies important machine functions not associated with actual tool moves**
N	**Sequence Number**	**Identifies the command block and specifies the order in which command block is executed within the program**
O	Sequence Number	Specifies order in which command block is executed for secondary head only
P	Dimension word	Specifies third rapid-traverse dimension
	Dimension word	Specifies tertiary-motion dimension parallel to X axis
Q	Dimension word	Specifies second rapid-traverse dimension
	Dimension word	Specifies tertiary-motion dimension parallel to Y axis
R	Dimension word	Specifies first rapid-traverse dimension
	Dimension word	Specifies tertiary-motion dimension parallel to Z axis
	Dimension word	**Specifies radius of arc used in circle interpolation**
S	**Spindle speed function**	**Specifies the speed at which the spindle is to rotate**
T	**Tool function**	**Specifies tool number to be selected on machines that have an automatic tool changer**
U	Dimension word	Specifies secondary motion dimension parallel to X axis
V	Dimension word	Specifies secondary motion dimension parallel to Y axis
W	Dimension word	Specifies secondary motion dimension parallel to Z axis
X	**Dimension word**	**Specifies primary motion dimension in X direction**
Y	**Dimension word**	**Specifies primary motion dimension in Y direction**
Z	**Dimension word**	**Specifies primary motion dimension in Z direction**

Figure 4-3 Letter addresses

necessary for the CNC controller to accomplish the given task. Additionally, some non-dimension words are dependent on dimension words to complete an instruction.

For dimension words and most non-dimension words, numeric values are variable. Consider the code in Figure 4-4. Command block N20 has an X word address of X1.0. In command block N40, the new X word address is X2.0. Hence, numeric values depend on the dimension that defines the tool center location in the *x* direction. The same is true for the Y and Z words.

In fact, the only letter addresses that do not have variable numeric digits are the preparatory and miscellaneous functions. For these, each digit combination has a distinct meaning. The G letter in preparatory functions (or the M letter in miscellaneous functions) precedes two digits to form part of an instruction. For example, G01 is the word for linear interpolation, G00 is the word for a point-to-point or rapid move, and M03 indicates the operator is to turn the spindle on. Necessarily, there is repetition of G- and M-codes in any program.

```
          '

          '

          '
N20 G00 X1.0 Y1.5 Z0.125
N30 G01 Z-0.0621
N40 G01 X2.0 Y3.5
          '

          '

          '
```

Figure 4-4 Sample of G-code

4.2.3 Command Blocks

Command blocks contain, at a minimum, one complete instruction. However, as we have seen, most modern CNC machines allow a command block to contain more than one instruction. For example, Figure 4-0 shows a command block—N1—that has two instructions. The first word address, G90, specifies absolute coordinates. The second word address, G20, signifies that the units are inches. However, for the majority of the command blocks in this figure, one complete instruction is provided. This is often done for the sake of clarity and to simplify the debugging process. Note also in Figure 4-0 that some word addresses always appear with other word addresses, because, like preparatory codes, some word addresses depend on other word addresses to convey a complete an instruction.

The standard order of word addresses within a command block is as follows:

1. Sequence number: N word

2. Preparatory function: G word, more commonly, "G-code"

3. Dimension words (*coordinates*)

    a) X, Y, Z (in this order) for movement direction of a typical machine.

    b) X, Y, Z, U, V, W, P, Q, R, A, B, C, D, E (in this order) for machines with multiple axes.

4. Interpolation parameters

    a) I, J, K: These letter addresses can have multiple meanings, depending on the preparatory function preceding their placement in the command block .

5. Feed function

    a. F: The F word is used when the feed applies to the direction of the movement of the tool involving one or more axes. In this common event the F word will follow the last dimension word.

    b. If the feed is to be specified by axis with the F, E, and D words, the feed function will immediately follow the dimension word for that axis.

6. Spindle speed function: S word

7. Tool function: T word

    a. Tool number: The tool number is used with machines that have automatic tool changers.

    b. When the D word is used with tool compensation preparatory functions, it will immediately follow the T word.

8. Miscellaneous function: M word, more commonly "M-code."

     Figure 4-5 shows the order of words in a block in spreadsheet fashion. The code listed is taken from Figure 4-0. The ANSI/EIA RS-274-D standard specifies that words cannot be repeated within a block. Also, words may be omitted when they are not needed in a command block. This indicates to the CNC controller that there is no change of state of the omitted word.

     Note that some CNC controllers allow some deviation from the standard. Compare the code in Figure 4-0 with that of Figure 4-5. Notice that for command blocks N20 and N40 the order of words in Figure 4-0 is the reverse of the order given above and in Figure 4-5. In both cases the M word is first. Additionally, in command block N40 the S word appears before the F word. This contradicts the order listed above. For most controllers this is permissible and

understandable. Again consider command block N20. Code M06 calls for a tool change. Code T22 specifies the tool. In the development of a program it is more natural to call for the tool change first, followed by the tool to be loaded. The same is true of command block N40. Code M03 turns on the spindle and code S2000 sets the speed. In the operation of a manual machine it is conceivable that the operator would turn the spindle on first, then adjust the speed. Consequently, some deviations from the standard order of words in a command block, particularly those associated with M-codes, are often permissible. However, one should consult the machine manual for allowable exceptions.

N	G - Code			Axis Coordinate			Center of Arc			Radius	Feed	Speed	Tool	M-Code
				X	Y	Z	I	J	K	R	F	S	T	
N10	G90	G70												
N20													T1	M06
N30	G00			X0	Y0	Z0.1								
N40											F4	S2000		M03

Figure 4-5 Order of words in a block

Another important command block and word address concept is that of *modality*. Some word addresses are considered "modal": A modal word address stays in effect until a contradictory instruction is issued or until the first instruction is cancelled. Figure 4-6 contains two listings of program code. The code on the right executes commands identical to the code on the left, but the code on the right is written with modal word addresses. For example, in command blocks N100 and N110 of the code on the left, both blocks have G00 following the sequence number. However, in the code on the right, the G00 is not listed in command block N110. This is because G00, which instructs the CNC controller to perform a rapid move under point-to-point control, is modal. Thus, if a rapid move is to stay in effect in a successive command block, the only code that needs to be issued is that for which the process will change. Consequently, only the (x, y) position change is listed in command block N110. In the next command block, N120, a G01 is issued, which instructs the CNC controller to perform linear interpolation (continuous path control) to Z-0.125. Note that only the Z dimension is listed because the (x, y) position from the previous command block does not change. The G01 code is also modal. Hence, it is not listed in the following command block, N130. Modality of word addresses is further discussed in subsequent sections.

4.2.4 Preparatory Functions

G-codes prepare the controller to perform a move, and the subsequent word addresses in the command block provide the parameters associated with the move. As already explained, preparatory codes consist of the letter G followed by two digits indicating the preparatory function to be executed. Figure 4-7 is a listing of G-codes. Refer to the five column headings next to the Description column. An X in the Modal column indicates the code is modal. The next two columns, Milling and Turning, indicate where the code is used. Some codes are used only in milling applications, some in turning, and some in both. The last two columns are used to designate the compliance with the ANSI/EIA RS-274-D and/or the ISO 6983 standards.

Program Code NOT Using Modality of Word Address Commands	Program Code Utilizing Modality of Word Addresses
N10 G90 G70	N10 G90 G70
N20 M06 T22	N20 M06 T22
N30 G00 X0 Y0 Z.1	N30 G00 X0 Y0 Z.1
N40 M03 S2000 F4	N40 M03 S2000 F4
N50 G00 X1.5 Y1.25	N50 G00 X1.5 Y1.25
N60 G01 Z-1.0	N60 G01 Z-1.0
N70 G01 Z.1 ————————————→	N70 Z.1
N80 G00 X2.5 Y2.75	N80 G00 X2.5 Y2.75
N90 G01 Z-1	N90 G01 Z-1
N100 G00 Z0.1000	N100 G00 Z0.1000
N110 G00 X3.2500 Y3.0000 ————→	N110 X3.2500 Y3.0000
N120 G01 Z-0.1250	N120 G01 Z-0.1250
N130 G01 X3.2500 Y0.7500 ————→	N130 X3.2500 Y0.7500
N140 G00 Z0.1000	N140 G00 Z0.1000
N150 G00 X3.2500 Y3.0000 ————→	N150 X3.2500 Y3.0000
N160 G01 Z-0.2500	N160 G01 Z-0.2500
N170 G01 X3.2500 Y3.0000 Z-0.2500 —→	N170 X3.2500 Y3.0000 Z-0.2500
N180 G01 X3.2500 Y0.7500 ————→	N180 X3.2500 Y0.7500
N190 G00 Z0.1000	N190 G00 Z0.1000
N200 G00 X3.2500 Y3.0000 ————→	N200 X3.2500 Y3.0000
N210 G01 Z-0.3750	N210 G01 Z-0.3750
N220 G01 X3.2500 Y3.0000 Z-0.3750 —→	N220 X3.2500 Y3.0000 Z-0.3750
N230 G01 X3.2500 Y0.7500 ————→	N230 X3.2500 Y0.7500
N240 G00 Z0.1000	N240 G00 Z0.1000
N250 G00 X3.2500 Y3.0000 ————→	N250 X3.2500 Y3.0000
N260 G01 Z-0.5000	N260 G01 Z-0.5000
N270 G01 X3.2500 Y3.0000 Z-0.5000 —→	N270 X3.2500 Y3.0000 Z-0.5000
N280 G01 X3.2500 Y0.7500 ————→	N280 X3.2500 Y0.7500
N290 G00 Z0.1000	N290 G00 Z0.1000
N300 G00 X3.2500 Y3.0000 ————→	N300 X3.2500 Y3.0000
N310 G01 Z-0.6250	N310 G01 Z-0.6250
N320 G01 X3.2500 Y3.0000 Z-0.6250 —→	N320 X3.2500 Y3.0000 Z-0.6250
N330 G01 X3.2500 Y0.7500 ————→	N330 X3.2500 Y0.7500
N340 G00 Z0.1000	N340 G00 Z0.1000
N350 G00 X3.2500 Y3.0000 ————→	N350 X3.2500 Y3.0000
N360 G01 Z-0.7500	N360 G01 Z-0.7500
N370 G01 X3.2500 Y3.0000 Z-0.7500 —→	N370 X3.2500 Y3.0000 Z-0.7500
N380 G01 X3.2500 Y0.7500 ————→	N380 X3.2500 Y0.7500
N390 G00 Z0.1000	N390 G00 Z0.1000
N400 G00 X3.2500 Y3.0000 ————→	N400 X3.2500 Y3.0000
N410 G01 Z-0.8750	N410 G01 Z-0.8750
N420 G01 X3.2500 Y3.0000 Z-0.8750 —→	N420 X3.2500 Y3.0000 Z-0.8750
N430 G01 X3.2500 Y0.7500 ————→	N430 X3.2500 Y0.7500
N440 G00 Z0.1000	N440 G00 Z0.1000
N450 G00 X3.2500 Y3.0000 ————→	N450 X3.2500 Y3.0000
N460 G01 Z-1.0000	N460 G01 Z-1.0000
N470 G01 X3.2500 Y3.0000 Z-1.0000 —→	N470 X3.2500 Y3.0000 Z-1.0000
N480 G01 X3.2500 Y0.7500 ————→	N480 X3.2500 Y0.7500
N490 G00 Z0.1000	N490 G00 Z0.1000
N500 G00 X0 Y0 Z1 ————————→	N500 X0 Y0 Z1
N510 M05	N510 M05
N520 M30	N520 M30

Figure 4-6 Demonstration of modal word addresses

G Code	Description	Modal	Milling	Turning	ANSI/EIA RS274-D	ISO 6983
G00	Rapid or point-to-point move.	X	X	X	X	X
G01	Linear interpolation	X	X	X	X	X
G02	Circle interpolation in the clockwise direction	X	X	X	X	X
G03	Circle interpolation in the counter-clockwise direction	X	X	X	X	X
G04	Dwell		X	X	X	X
G06	Parabolic interpolation	X	X	X	X	X
G17	X-Y plane selection for circle interpolation and/or cutter compensation	X	X		X	X
G18	Z-X plane selection for circle interpolation and/or cutter compensation	X	X		X	X
G19	Y-Z plane selection for circle interpolation and/or cutter compensation	X	X		X	X
G20	Not assigned by the ANSI/EIA standard but used by some controllers to specify programming in inch units	X	X	X		
G21	Not assigned by the ANSI/EIA standard but used by some controllers to specify programming in metric units	X	X	X		
G28	Automatic return to reference point		X	X		
G29	Automatic return from reference point		X	X		
G32	Not assigned by the ANSI/EIA standard but used by some controllers to specify constant lead thread cutting	X		X		
G33	Constant lead thread cutting	X		X	X	X
G34	Increasing lead thread cutting	X		X	X	X
G35	Decreasing lead thread cutting	X		X	X	X
G40	Cancel cutter compensation/offset	X	X	X	X	X
G41	Cutter compensation left	X	X	X	X	X
G42	Cutter compensation right	X	X	X	X	X
G43	Cutter offset inside corner	X	X	X	X	X
G43	Tool length compensation plus	X	X	X		
G44	Cutter offset inside corner	X	X	X	X	X
G44	Tool length compensation minus	X	X	X		
G49	Not assigned by the ANSI/EIA standard but used by some controllers to cancel tool length compensation	X	X	X		
G53	Datum shift cancel	X	X	X		X
G54-G59	Datum Shifts	X	X	X		X
G54-G59	Preprogrammed work piece coordinate systems	X	X	X		
G70	Inch unit programming	X	X	X	X	X
G71	Metric unit programming	X	X	X	X	X
G72	Clockwise three dimensional circle interpolation	X	X		X	
G72	Finishing cut cycle along Z axis	X		X		

Figure 4-7 G-code list continued on next page

G Code	Description	Modal	Milling	Turning	ANSI/EIA RS274-D	ISO 6983
	Counter clockwise three dimensional circle interpolation	X	X		X	
G73	Deep hole peck drilling cycle	X	X			
	OD & ID roughing cyle along Z axis	X		X		
	Cancel multiquadrant circular interpolation	X	X	X	X	
G74	Move to home position	X	X	X		X
	Left hand tapping cycle	X	X			
	Rough facing cyle	X		X		
G75	Multiquadrant circle interpolation	X	X	X	X	
	Roughing routine for castings or forgings	X	X	X		
G80	Cancel fixed cycles	X	X	X	X	X
G81	No dwell and rapid out drill cycle	X	X	X	X	X
G82	Dwell and rapid out drill cycle	X	X	X	X	X
G83	Deep hole peck drillig cycle	X	X	X	X	X
G84	Right-hand tapping cycle	X	X	X	X	X
	Left-hand tapping cycle	X	X	X		
G85	No dwel, feed out boring cycle	X	X	X	X	X
G86	Spindle stop, rapid out boring cycle	X	X	X	X	X
G87	Manual retraction boring cycle	X	X	X	X	X
	Manual retraction, spindle stop, boring cycle	X	X	X	X	X
G88	Rectangular and circular pocket milling roughing cycle	X	X			
	Rectangular and circular pocket milling finishing cycle	X	X			
	Dwell and feed out boring cycle	X	X	X	X	X
G89	Irregular shape pocket milling roughing cycle	X	X	X	X	X
	Irregular shape pocket milling finishing cycle	X	X	X	X	X
G90	Absolute dimensions	X	X	X	X	X
G91	Incremental dimensions	X	X	X	X	X
G92	Preload registers to shift coordinate axes relative to the current tool position		X	X	X	X
G93	Inverse time feed rate (Velocity/distance)	X	X	X	X	X
G94	Feed rate in inches or millimeters per minute	X	X	X	X	X
G95	Feed rate in inches or millimeters per revolution	X	X	X	X	X
G96	Constant surface speed feed, feet per minute	X	X	X	X	X
G97	Spindle speed specified in rpm	X	X	X	X	X
G98	Set initial plane default		X			
G99	Return to retract plane		X			

Not all CNC machine manufacturers adhere strictly to ANSI/EIA RS-274-D and ISO 6983 standards. Many manufacturers use the less common or unassigned G-codes for commands that are not listed in the standard. Consequently, a program written for one machine does not necessarily work on a similar machine made by a different manufacturer. One should always consult a machine's user manual, in particular, the machine's format classification sheet and format detail, to ensure that the G-codes used by its CNC controller conform to the machine's.

4.2.5 Common Preparatory Functions

Detailed explanations of some of the more important and basic preparatory codes are listed in this section. These codes are termed "basic" because they apply to both milling and turning operations. The codes that follow are explained as they relate to the three-axis milling process. A turning example in Section 4.6.1 demonstrates application of the codes to lathe operations.

G00 Rapid Move
Function
G00 moves the tool center rapidly to a position without regard to path taken. Consequently it is a non-cutting move, used only to move the tool in air. The G00 code is modal.

Input Format
N___ G00 X___ Y___ Z___

Terminology
N___Sequence number
G00 G Code for a rapid move
X___x-coordinate of the destination point of the tool
Y___y-coordinate of the destination point of the tool
Z___z-coordinate of the destination point of the tool.

Explanation
The G00 word address is called a *rapid move* because it moves the tool rapidly to a new position. The CNC controller utilizes point-to-point control to execute this move. With this type of control each axis is moved to the desired coordinate without consideration of how other axes are moving. Consequently, the path taken by the tool center point is not predictable. Additionally, the move is intended to occur as quickly as possible, so it is executed by each axis at maximum motor speed (there are no parameters in the code to control the speed of the move). However, there is typically some manual maximum motor speed control available through the machine controller user interface.

Because of the unpredictability of the tool path associated with rapid moves, standard practice with milling operations is to program rapid moves in the z direction separate from

rapid moves in the (*x,y*)-plane. The same holds true for turning operations where rapid moves in the *z* direction are programmed separate from moves in the *x* direction. Figure 4-8 demonstrates a milling operation where the tool center point is moved from position A to position B for the purpose of drilling a hole in the workpiece. If the rapid move is programmed simultaneously in the (*x,y*)-plane and *z*-axis, the tool might crash into the workpiece (Figure 4-8a). Separating the (*x,y*)-plane motion from the *z*-axis motion controls the path of the tool, eliminating the crash (Figure 4-8b). Movement along all three axes is often permissible when the movement is well away from the workpiece.

Figure 4-8a Rapid move tool crash

Figure 4-8b Rapid move without tool crash

Example Program

We turn our attention to the milling operation G-code shown in Figure 4-9. The actual code is listed in the left column and a description of what the code does is shown on the right. The code listed is depicted graphically in Figure 4-10. This code mills a step into the end of the workpiece as shown. Note how the rapid move G00 is used in command blocks N30, N50, N60, and N100 to move the tool into position *above* the workpiece, but it is not used to move *into* the workpiece. Recall that a move into the workpiece is always identifiable because the *z*-coordinate will be negative, and that if a coordinate word address is not repeated in subsequent command blocks, its value does not change. This is why there is no *z*-coordinate listed in command blocks N60, N80, N100, and N120, and why there is no *x*- or *y*-coordinate in command blocks N50, N70, N90, N110, and N130. These command blocks also demonstrate the separation of (*x,y*)-plane rapid moves from *z*-axis rapid moves. Rapid moves are also commonly used to move or pull the tool out of the workpiece. Command blocks N90 and N130 are examples of this. Command blocks N30 and N1930 show how movement in all three axes can be used if the tool is moving away from the workpiece.

Code	Code Description
N10 G90 G70	Absolute coordinates in inch units
N20 T10 M06	Request Tool change with Tool # 10 selected
N30 **G00** X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 S2000 M03	Turn on spindle, clockwise at 2000 rpm
N50 **G00** Z0.1000	Rapid move to just above work (position (0, 0, 0.1))
N60 **G00** X11.1562 Y0.0000	Rapid move to just above cutting location (position (11.1562, 0, 0.1))
N70 G01 Z-0.0625 F4	Linear interpolation or move into work at a specified feed rate. (position (11.1562, 0, -0.0625))
N80 G01 X11.1562 Y5.7500	Linear move to position (11.1562, 5.75, -0.0625)
N90 **G00** Z0.1000	Rapid move to position (11.1562, 5.75, 0.1)
N100 **G00** X11.0312 Y0.0000	Rapid move out of work to position (11.0312, 0, 0.1)
N110 G01 Z-0.0625	Linear move into work at a specified feed rate (11.0312, 0, -0.0625)
N120 G01 X11.0312 Y5.7500	Linear move to position (11.0312, 5.75, -0.0625)
N130 **G00** Z0.1000	Rapid move out of work to position (11.0312, 5.75, 0.1)
.	
.	
.	
N1930 **G00** X0 Y0 Z1	Rapid to home position
N1940 M05	Shutdown spindle
N1950 M30	End Program

Figure 4-9 G00 sample code

Figure 4-10 Workpiece created with code from Figure 4-9

G01 − Linear Interpolation

Function

The G01 code is used to remove material from a workpiece along a linear or straight-line path at a specified speed, controlled by the feed rate. Hence, it is a cutting move. The tool path is not limited to a fixed plane. The destination of the move can be specified so that all three axes move simultaneously. Therefore, three-dimensional angular cuts can be created, as shown in Figure 4-11. The G01 code is modal.

Input Format
N___ G01 X___ Y___ Z___ F___

Terminology
N___Sequence number
G01 G Code for a linear interpolation move
X___*x*-coordinate of the destination point of the tool
Y___*y*-coordinate of the destination point of the tool
Z___*z*-coordinate of the destination point of the tool
F___Feed rate of the tool into the work in inches per minute (ipm).

Explanation

G01 is a cutting move because it allows one to control the speed at which the tool moves into the workpiece. The controller utilizes continuous path control to maintain this feed rate. Each axis is individually controlled to achieve the desired feed rate along the path. Because of the added control, an additional parameter, the F word, is specified along

with the coordinates of the destination point. The F word is modal. Thus, once specified it remains in effect until changed. Since the F word is modal it can be specified prior to the issuance of a G01 code in the program setup section of the code. Controlling the feed rate in combination with control of the spindle rotational speed (S word) enables effective material removal.

Figure 4-11 Three-dimensional linear cut

The G01 code is the code of choice to move the tool into the workpiece. Typically, the tool is moved first in air, to directly above the desired cutting location with rapid moves. The tool can then be plunged into the work (movement in the negative *z* direction) with the G01 code. Then the G01 code, in combination with other cutting moves, is used to produce the desired shape. A rapid move G00 then removes the tool from the work and moves it to the next desired cutting location.

Example Program

Figure 4-12 lists the same program shown previously in Figure 4-9 except in this figure the G01 moves are highlighted. Command blocks N70 through N100 specify one cutting cycle. A typical cutting cycle includes a linear move into the material, cutting moves in the material to produce the desired shape, a rapid move out of the material, and a rapid move to the next cutting position. For this example, command block N70 moves the cutting tool into the work with a G01 linear move. Note that the feed rate is specified only

within this command block. Command block N80 performs a cutting move to position (11.1562, 5.75, −0.0625). Only the *x*- and *y*- coordinates are specified because the *z*-coordinate does not change from the prior block. Block N90 is a rapid move out of the material to just above it, and N100 is a rapid move to the next cutting location. This cutting cycle is depicted in Figure 4-13. Another cutting cycle begins in command block N110 and concludes with command block N130.

Code	Code Description
N10 G90 G70	Absolute coordinates in inch units
N20 T10 M06	Request Tool change with Tool # 10 selected
N30 G00 X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 S2000 M03	Turn on spindle, clockwise at 2000 rpm
N50 G00 Z0.1000	Rapid move to just above work (position (0, 0, 0.1))
N60 G00 X11.1562 Y0.0000	Rapid move to just above cutting location (position (11.1562, 0, 0.1))
N70 **G01** Z-0.0625 **F4**	Linear interpolation or move into work at a specified feed rate. (position (11.1562, 0, -0.0625))
N80 **G01** X11.1562 Y5.7500	Linear move to position (11.1562, 5.75, -0.0625)
N90 G00 Z0.1000	Rapid move to position (11.1562, 5.75, 0.1)
N100 G00 X11.0312 Y0.0000	Rapid move out of work to position (11.0312, 0, 0.1)
N110 **G01** Z-0.0625	Linear move into work at a specified feed rate (11.0312, 0, -0.0625)
N120 **G01** X11.0312 Y5.7500	Linear move to position (11.0312, 5.75, -0.0625)
N130 G00 Z0.1000	Rapid move out of work to position (11.0312, 5.75, 0.1)
.	
.	
.	
N1930 G00 X0 Y0 Z1	Rapid to home position
N1940 M05	Shutdown spindle
N1950 M30	End Program

Figure 4-12 G01 sample code

Most programs, as in this example, consist of a series of such cutting cycles, which are necessary because only a specified amount of material may be removed at a time. This amount is dictated by several factors, including the kind of material being cut, the size and type of cutting tool, and desired speeds and feeds. Note also that cutting cycles may contain other interpolation cutting moves, including circular, helical, and parabolic.

Figure 4-13 Linear cutting cycle

G02 − Circle Interpolation: Clockwise Direction

Function

The G02 code is used to remove material from a workpiece in a circular path. Like G01, G02 is a cutting move. It directs the tool along an arc at a specified feed rate in the clockwise (CW) direction. The tool path is limited to a fixed plane such as (x,y)-, (y,z)-, or (x,z)-plane. For milling, the default plane is the (x,y)-plane. For turning, circle interpolation is limited to the (x,z)-plane. For milling operations the plane in which the cut occurs can be changed by G-codes G17, G18, or G19. The G02 code is modal. There are two input formats for arcs, which are dependent on the length of the arc.

Input Format

The following format can be used for any length of arc:

N___ G02 X___ Y___ Z___ I___ J___ K___ F___

The following format can only be used for arcs of 90 degrees or less:

N___ G02 X___ Y___ Z___ R___ F___

Terminology

N___Sequence number

G02 G Code for a circle interpolation move in the CW direction

X___x-coordinate of the destination point of the tool

Y___*y*-coordinate of the destination point of the tool

Z___*z*-coordinate of the destination point of the tool

I___Relative distance in the *x* direction from the start point of the arc to the center point of the radius

J___Relative distance in the *y* direction from the start point of the arc to the center point of the radius

K___Relative distance in the *z* direction from the start point of the arc to the center point of the radius

R___Radius of arc

F___Feed rate of the tool into the work in ipm.

Explanation

The first format from the above list can be used to move the cutter in an arc of any length, including a complete circle. For this format one must specify the location of the center point of the arc relative to the start point of the tool point center. This is accomplished by specifying the relative distance in the *x*, *y*, and *z* direction using I, J, and K words respectively. Figure 4-14 shows a CW circular tool path in the (*x,y*)-plane. The start point, end point, and center are specified along with the corresponding I and J directions. The I value is found by subtraction of the start point *x*-coordinate from the center point *x*-coordinate:

Center point	X:	2
minus start point	X:	$\underline{-1}$
	I =	1

A similar operation is performed to determine the J value:

centerpoint Y:	1.25
start point Y:	$\underline{-2.982}$
J =	−1.732

Input Format

The code for this cutting move would be:

$$\text{N99 G02 X3.732 Y0.25 I1 J-1.732 F4}$$

Since the cutting move is made in the (*x,y*)-plane, only the I and J parameters are specified. If the cut were to be made in the (*y,z*)-plane, only J and K would be specified. Correspondingly, for circle interpolation cuts made in the (*x,z*)-plane, as with turning operations, only I and K are used.

Figure 4-14 G02 CW arc format 1

For circle interpolation moves of less than 90 degrees, the second, much simpler format can be used. The only extra dimension word that needs to be specified is the radius of the arc. The code for the arc shown in Figure 4-15 would be:

N89 G02 X3.732 Y0.25 R2 F4

Recall that the feed word is modal. Therefore, if the F-word were specified in a prior *cutting move*, it would not need to be specified again if the same feed rate were to be used.

Figure 4-15 G02 CW arc format 2

Example Program

Figure 4-16 shows a portion of the program code to mill the circular pocket shown in Figure 4-17. Command block N1320 uses the CW circle interpolation command G02 to cut a 180-degree arc. Command blocks N1390 and N1440 cut the same arc at different depths. This is necessary because only a certain amount of material can be removed by the cutter with each cut. The depth of cut specification, which will be discussed in subsequent sections, controls how much material is removed with each cutting cycle or pass. In this case the cutter is removing 0.25 inch of material per cut. To reach a desired pocket depth of 0.75 inch from a starting depth of −1 (that is, −1.75 inches in the z-direction from the origin) requires three passes or three cutting cycles. The center of the arc is specified by the I and J words in the command block. For command block N1340, the center of the arc is located 2.32 inches above the start point of the arc.

Code	Code Description
N10 G90 G70	Absolute coordinates in inch units
N20 M06 T10	Request Tool change with Tool # 10 selected
N30 G00 X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 F4 S2000 M03	Feed rate 4 ipm and turn on spindle, clockwise at 2000
N50 G00 Z.1	Rapid move to just above work (position (0, 0, 0.1))
'	
'	
'	
N1310 G00 X7.3750 Y0.5550	Rapid move to position
N1320 G01 Z-1.2500	Linear cutting move into part
N1340 **G02** X7.3750 Y5.1950 **I0.0000 J2.3200**	Clockwise circle interpolation to position (7.375, 0.1950, -1.25). Arc center located at (0, 2.32) from tool start point
N1350 G00 Z-0.7400	Rapid out of material
N1360 G00 X7.3750 Y0.5550	Move to start of next cut
N1370 G01 Z-1.5000	Linear cutting move into part
N1390 **G02** X7.3750 Y5.1950 **I0.0000 J2.3200**	Clockwise circle interpolation to position (7.375,0.1950, -1.50). Arc center located at (0, 2.32) from tool start point
N1400 G00 Z-0.7400	Rapid out of material
N1410 G00 X7.3750 Y0.5550	Move to start of next cut
N1420 G01 Z-1.7500	Linear cutting move into part
N1440 **G02** X7.3750 Y5.1950 **I0.0000 J2.3200**	Clockwise circle interpolation to position (7.375,0.1950, -1.50). Arc center located at (0, 2.32) from tool start point
'	
'	
'	
N3470 G00 X6.4375 Y2.8750	Rapid move to position
N3480 G01 Z-2.0000	Linear cutting move into part
N3500 **G02** X7.3750 Y3.8125 **R0.9375**	Clockwise circle interpolation to position (7.375, 3.8125, -2.00) using a radius of 0.9375
N3510 G00 Z-1.7400	Rapid out of material
N3520 G00 X6.4375 Y2.8750	Move to start of next cut
N3530 G01 Z-2.2500	Linear cutting move into part
N3550 **G02** X7.3750 Y3.8125 **R0.9375**	Clockwise circle interpolation to position (7.375, 3.8125, -2.25) using a radius of 0.9375
N3560 G00 Z-1.7400	Rapid out of material
'	
'	
'	
N7120 G00 X0 Y0 Z1	Rapid to home position
N7130 M05	Shutdown spindle
N7140 M30	End Program

Figure 4-16 G02 sample code

Command blocks N3470 to N3560 of Figure 4-16 list a portion of the code to mill the circular pocket shown in Figure 4-18. The circle interpolation codes in these command blocks use the second formatting option, which is possible because these arcs are less than 90 degrees.

Figure 4-17 G02 milled pocket format

Figure 4-18 G02 format 2 milled pocket

G03 – Circle Interpolation: Counterclockwise Direction

Function

The G03 code is used to remove material from a workpiece in a circular path. Like G01 and G02, G03 is a cutting move. It directs the tool along an arc at a specified feed rate in the counterclockwise (CCW) direction. The tool path is limited to a fixed plane. For milling, the default plane is the (x,y)-plane. For turning, circle interpolation is limited to the (x,z)-plane. For milling operations, the plane in which the cut occurs can be changed by G-codes G17, G18, or G19. The G03 code is modal. The two input formats for arcs depend on the arc length specified.

Input Format

The following format can be used for any length of arc:

N___ G03 X___ Y___ Z___ I___ J___ K___ F___

The following format can be used only for arcs of 90 degrees or less:

N___ G03 X___ Y___ Z___ R___ F___

Terminology

N___Sequence number

G03 G Code for a circle interpolation move in the CCW direction

X___x-coordinate of the destination point of the tool

Y___y-coordinate of the destination point of the tool

Z___z-coordinate of the destination point of the tool

I___Relative distance in the x-direction from the start point of the arc to the center point of the radius

J___Relative distance in the y-direction from the start point of the arc to the center point of the radius

K___Relative distance in the z-direction from the start point of the arc to the center point of the radius

R___Radius of arc

F___Feed rate of the tool into the work in ipm.

Explanation

See previous explanation. Figure 4-19 shows a CCW circular tool path in the (x,y)-plane. The start point, end point, and center are specified along with their corresponding I and J directions. The I value is found by subtraction of the start point x-coordinate (1) from the center point x-coordinate (3.732):

center point X: 3.732

    start point X: −1

            I   = 2.732

A similar operation is performed to determine the J value:

center point Y: 2.75

start point Y: −2.982

J   = −0.232

Input Format

The code for this cutting move (Figure 4-19) would be:

N99 G03 X2.0 Y0.625 I2.732 J-0.232 F4

Figure 4-19 G03 CCW arc format 1

Since the cutting move is made in the (x,y)-plane, only the I and J parameters are specified. If the cut were to be made in the (y,z)-plane, only J and K would be specified. Correspondingly, for circle interpolation cuts in the (x,z)-plane, only I and K are used.

For circle interpolation moves of less than 90 degrees, the second, much simpler, format can be used. The only extra word that needs to be specified is the radius of the arc.

The code for the arc shown in Figure 4-20 would be:

N89 G03 X2.0 Y0.625 R2.742 F4

Recall that the feed word is modal. Therefore, if the F word were specified in a prior cutting move, it would not have to be specified again if the same feed rate were to be used.

Figure 4-20 G03 CCW arc format 2

Example Program

Figure 4-21 shows a portion of the program code required to mill the circular pocket shown in Figure 4-22. Command block N4990 uses the CCW circle interpolation command G03 to cut a 180-degree arc. Command block N5040 cuts the same arc at a different depth. In this case, the cutter is removing 0.25 inch of material per cut. Two passes are needed to reach a desired pocket depth of 0.50 inch. The center of the arc is specified by the I and J words in the command block. For command block N4990, the center of the arc is located 2.00 inches above the start point of the arc.

Code	Code Description
N10 G90 G70	Absolute coordinates in inch units
N20 T10 M06	Request Tool change with Tool # 10 selected
N30 G00 X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 F4 S2000 M03	Feed rate 4 ipm and turn on spindle, clockwise at 2000
N50 G00 Z.1	Rapid move to just above work (position (0, 0, 0.1))
'	
'	
'	
N4970 G00 X2.8750 Y4.8750	Rapid move to position
N4980 G01 Z-0.2500	Linear cutting move into part
N4990 **G03** X2.8750 Y0.8750 **I0.0000 J-2.0000**	Counter clockwise circle interpolation to position (2.875, 0.875, -0.25). Arc center located at (0, -2) from tool start
N5000 G00 Z0.1000	Rapid out of material
N5010 G00 X2.8750 Y4.8750	Move to start of next cut
N5020 G01 Z-0.5000	Linear cutting move into part
N5040 **G03** X2.8750 Y0.8750 **I0.0000 J-2.0000**	Counter clockwise circle interpolation to position (2.875, 0.875, -0.50). Arc center located at (0, -2) from tool start
N5050 G00 Z0.1000	Rapid out of material
'	
'	
'	
N5920 G00 X2.8750 Y3.8125	Rapid move to position
N5930 G01 Z-1.0000	Linear cutting move into part
N5950 **G03** X1.9375 Y2.8750 **R0.9375**	Counter clockwise circle interpolation to position (1.9375, 2.875, -1.00) using a radius of 0.9375
N5960 G00 Z-0.7400	Rapid out of material
N5970 G00 X2.8750 Y3.8125	Move to start of next cut
N5980 G01 Z-1.2500	Linear cutting move into part
N6000 **G03** X1.9375 Y2.8750 **R0.9375**	Counter clockwise circle interpolation to position (1.9375, 2.875, -1.25) using a radius of 0.9375
N6010 G00 Z-0.7400	Rapid out of material
'	
'	
'	
N7120 G00 X0 Y0 Z1	Rapid to home position
N7130 M05	Shutdown spindle
N7140 M30	End Program

Figure 4-21 G03 sample code

146   Chapter 4

Figure 4-22 G03 milled pocket format 1

Figure 4-23 G03 milled pocket format 2

Command blocks N5920 to N6010 list a portion of the code to mill the circular pocket shown in Figure 4-23. The circle interpolation codes in these command blocks use the second formatting option. This is possible because these arcs are less than 90 degrees.

G04 – Dwell

Function

The G04 code is used to stop axis movement for a specified amount of time. It is most often used after a cutting move to allow chips to be removed by the cutter. During a dwell command all axis movement is stopped while the cutter continues to rotate at the previously set spindle speed. The G04 code is nonmodal.

Input Format

N___ G04 P___

Terminology

N___Sequence number

G04 G Code for initiating a dwell

P___Time in seconds to delay before executing next command block.

Explanation

The dwell command stops cutter movement for a specific amount of time. It is commonly used after a plunging cutter move to allow chips to clear the cutter prior to initiation of a contouring cut. During the dwell the spindle continues to rotate. Caution must be exercised when dwells are used because excessive use of dwell commands can substantially prolong the machining cycle time.

Example Program

Figure 4-24 shows a portion of the program code required to mill two holes to a depth of 2.875 inches. Command block N4940 demonstrates the dwell command. This command block initiates a dwell for 2 seconds. This is to allow the chips created by command block N4930 to clear the cutter. Command block N7100 issues a 1-second delay.

Code	Code Description
N10 G90 G70	Absolute coordinates in inch units
N20 T10 M06	Request Tool change with Tool # 10 selected
N30 G00 X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 F4 S2000 M03	Feed rate 4 ipm and turn on spindle, clockwise at 2000
N50 G00 Z.1	Rapid move to just above work (position (0, 0, 0.1))
'	
'	
'	
N4920 G00 X7.375 Y2.875	Rapid move to cutting position
N4930 G01 Z-2.875	Plunge into work to a depth of -2.875
N4940 **G04 P2**	Dwell for 2 seconds
N4950 G00 Z.1	Rapid out of hole
'	
'	
N7080 G00 X2.875 Y2.875	Rapid move to cutting position
N7090 G01 Z-2.875	Plunge into work to a depth of -2.875
N7100 **G04 P1**	Dwell for 1 second
N7110 G00 Z.1	Rapid out of hole
N7120 G00 X0 Y0 Z1	Rapid to home position
N7130 M05	Shutdown spindle
N7140 M30	End Program

Figure 4-24 G04 sample code

G70 − Inch Units

   Function

   The G70 code informs the machine controller that tool path coordinates are in inch units. Note that the G70 code is specified by the ANSI/EIA RS274-D standard. However, the G20 code is often used by many controllers to indicate inch units. Both the G20 and G70 codes are modal.

   Input Format

   N___ G70

   Terminology

   N___Sequence number

   G70 G Code specifying inch units.

   Explanation

   Numerical control programs can be developed in units of either inches or millimeters. Hence, when inch units are desired a G70 code is issued. This code is typically at the beginning of the program, in the program setup section. However, if the operator desires, both systems of units can be used within a program. Thus, a G70 code may be used

elsewhere in the program if the units are to be switched from inches to millimeters, then back to inches.

Example Program

All of the example programs presented thus far have a G70 code listed at the very beginning of the program. Good programming practice dictates that the G70 code be one of the first codes specified in the program setup section. Examples are shown in Figures 4-21 and 4-24.

G71 − Metric Units
Function

The G71 code informs the machine controller that coordinates are in metric units. Specifically, the unit of measure when the G71 code is issued is millimeters. Note that the G71 code is specified by the ANSI/EIA RS274-D standard. However, the G21 code is often used by many controllers to indicate metric units. Both the G21 and G71 codes are modal.

Input Format
N___ G71
Terminology
N___Sequence number
G71 G Code specifying metric (millimeter) units.

Explanation

As in the previous explanation, numerical control programs may specify units of either millimeters or inches. Millimeter units require a G71 code be issued, typically at the beginning, in the program setup section. However, if desired, both systems of units can be used within the same program. Thus, a G71 code may be used elsewhere in the program if the units are to be switched from millimeters to inches and then back to millimeters.

Example Program

Refer to G70 program examples.

G90 − Absolute Coordinates
Function

The G90 code informs the machine controller that tool path coordinates are measured relative to a fixed origin of the workpiece, the program reference zero (PRZ). The G90 code is modal.

Input Format
N___ G90

Terminology

N___Sequence number

G90 G Code specifying tool path coordinates are measured relative to a fixed origin
point.

Explanation

Recall from Section 4.3.3 that tool path coordinates can be specified in either
absolute or incremental coordinates. Absolute coordinates specify the coordinates relative
to a fixed reference point on the workpiece, or the PRZ. Incremental coordinates, on the
other hand, specify the coordinates relative to the last point along the tool path. The G90
code dictates the use of absolute coordinates. The G90 code, along with the G70/G71
codes, is located at the very beginning of the program setup section. Note that most
machine controllers allow both absolute and relative coordinate use within a single
program. Thus, G90 may appear in the body of the program as well.

Example Program

Figure 4-25 shows the code to machine the part shown in Figure 4-26. The G90 is
issued at the beginning of the program in command block N10. This program utilizes
absolute coordinates exclusively. Command blocks N60 to N290 mill the slot in four
passes. This is necessary because the depth of cut for each pass is limited to 0.250 inch.

Code	Code Description
N10 **G90** G70	Absolute coordinates in inch units
N20 T22 M06	Request Tool change with Tool # 22 selected
N30 G00 X0 Y0 Z1.0	Rapid move to position (0, 0, 1)
N40 F4 S2000 M03	Feed rate 4 ipm and turn on spindle, clockwise at 2000 rpm
N45 G00 Z0.1	Rapid move to just above work (position (0, 0, 0.1))
N50 G00 X1.5 Y1.25	Rapid move to 1st hole
N60 G01 Z-1.0	Linear move to drill 1st hole
N70 G01 Z.1	Rapid move out of material
N80 G00 X2.5 Y2.75	Rapid move to location of 2nd hole
N90 G01 Z-1	Linear move to drill 2nd hole
N100 G00 Z0.1000	Rapid move out of material
N110 G00 X3.2500 Y3.0000	Rapid move to start of slot
N120 G01 Z-0.250	Linear move into material
N130 G01 X3.2500 Y0.7500	Linear move to mill 1st pass
N140 G00 Z0.1000	Rapid move out of material
N150 G00 X3.2500 Y3.0000	Rapid move to start of slot for 2nd pass
N160 G01 Z-0.500	Linear move into material
N180 G01 X3.2500 Y0.7500	Linear move to mill 2nd pass
N190 G00 Z0.1000	Rapid move out of material
N200 G00 X3.2500 Y3.0000	Rapid move to start of slot for 3rd pass
N210 G01 Z-0.750	Linear move into material
N230 G01 X3.2500 Y0.7500	Linear move to mill 3rd pass
N240 G00 Z0.1000	Rapid move out of material
N250 G00 X3.2500 Y3.0000	Rapid move to start of slot for 4th pass
N260 G01 Z-1.000	Linear move into material
N280 G01 X3.2500 Y0.7500	Linear move to mill 4th pass
N290 G00 Z0.1000	Rapid move out of material
N300 G00 X0 Y0 Z1	Rapid to home position
N310 M05	Shutdown spindle
N320 M30	End Program

Figure 4-25 G90 sample code

Figure 4-26 Part produced from Figure 4-25 code

G91 − Relative Coordinates
Function
The G91 code informs the machine controller that tool path coordinates are measured incrementally, relative to last point on the tool path. The G91 code is modal.

Input Format
N___ G91
Terminology
N___Sequence number

G91 G Code specifying tool path coordinates are measured incrementally, relative to the last point on the tool path

Explanation
The explanation for the G91 code is the same as the G-90 code explanation.

Example Program
Figure 4-27 lists alternative code to machine the part shown in Figure 4-26. This is a simplified example of how a combination of absolute and incremental coordinates can be used in a program. The code accomplishes the same task as that in Figure 4-25. Again, a G90 code is issued at the beginning of the program specifying absolute coordinates. The first and second holes are created with absolute coordinates. However, milling of the slot is accomplished with both absolute and incremental coordinates. Command block N110 locates the start of the slot and N120 cuts into the material using absolute coordinates. Command block N130, however, uses incremental coordinates to mill the slot. Command block N140 switches back to absolute coordinates to withdraw the tool and move to the start of the block for the second pass. This procedure is repeated a total of four times to mill the slot to the correct depth. This is a typical example of how incremental and absolute coordinates can be used in combination to produce a part, with absolute coordinates used to locate the feature to be created, and then incremental coordinates to create the feature. This technique is often used to produce distinct bolt patterns distributed over a workpiece.

Code	Code Description
N10 **G90** G70	Absolute coordinates in inch units
N20 T22 M06	Request Tool change with Tool # 22 selected
N30 G00 X0 Y0 Z1	Rapid move to position (0, 0, 1)
N40 F4 S2000 M03	Feed rate 4 ipm and turn on spindle, clockwise at 2000 rpm
N45 G00 Z0.1	Rapid move to just above work (position (0, 0, 0.1))
N50 G00 X1.5 Y1.25	Rapid move to 1st hole
N60 G01 Z-1.0	Linear move to drill 1st hole
N70 G01 Z.1	Rapid move out of material
N80 G00 X2.5 Y2.75	Rapid move to location of 2nd hole
N90 G01 Z-1	Linear move to drill 2nd hole
N100 G00 Z0.1000	Rapid move out of material
N110 G00 X3.2500 Y3.0000	Rapid move to start of slot
N120 G01 Z-0.250	Linear move into material
N130 **G91** G01 X0 Y-2.25	Switch to relative coordinates, Linear move to mill 1st pass
N140 **G90** G00 Z0.1000	Switch back to absolute coordinates, Rapid move out of material
N150 G00 X3.2500 Y3.0000	Rapid move to start of slot for 2nd pass
N160 G01 Z-0.500	Linear move into material
N180 **G91** G01 X0 Y-2.25	Switch to relative coordinates, Linear move to mill 2nd pass
N190 **G90** G00 Z0.1000	Switch back to absolute coordinates, Rapid move out of material
N200 G00 X3.2500 Y3.0000	Rapid move to start of slot for 3rd pass
N210 G01 Z-0.750	Linear move into material
N230 **G91** G01 X0 Y-2.25	Switch to relative coordinates, Linear move to mill 3rd pass
N240 **G90** G00 Z0.1000	Switch back to absolute coordinates, Rapid move out of material
N250 G00 X3.2500 Y3.0000	Rapid move to start of slot for 4th pass
N260 G01 Z-1.000	Linear move into material
N280 **G91** G01 X0 Y-2.25	Switch to relative coordinates, Linear move to mill 4th pass
N290 **G90** G00 Z0.1000	Switch back to absolute coordinates, Rapid move out of material
N300 G00 X0 Y0 Z1	Rapid to home position
N310 M05	Shutdown spindle
N320 M30	End Program

Figure 4-27 G91 sample code

4.2.6 Miscellaneous Functions

Miscellaneous functions (alternately, auxiliary functions) are specified as "M-codes." We have already seen them in many of the input formats. These codes are used to control all machine functions not associated with an actual (continuous) tool move. Thus, turning a spindle on, changing to a different tool, and unclamping a workpiece are all examples of machine functions that can be controlled with M-codes. M-codes consist of the letter "M" followed by a two-digit code. The digits indicate the specifics of the miscellaneous function. Figure 4-28 is a listing of M-codes specified by the ANSI/EIA RS-274-D standard. The figure provides a brief description of the code, its modality, and the point at which the function is executed or started relative to any preparatory functions contained in the same command block. If both boxes do not contain an X, then the point at which the function starts is dependent on the particular machine.

M-codes are normally the last word address of the command block. Generally, some controllers will allow variation from this standard. However, only one M-code is allowed per command block.

M Code	Description	When function starts relative to motion in block		Modal
		With	After	
M00	Stops the program and correspondingly the machine		X	
M01	Optional method to stop the program and machien		X	
M02	Indicates end of program		X	
M03	Turn on spindle in the clockwise (CW) direction	X		X
M04	Turn on spindle in the counter clockwise (CCW) direction	X		X
M05	Turn off the spindle		X	X
M06	Initiates a tool change			
M07	Turns on coolant number 1	X		X
M08	Turns on coolant number 2	X		X
M09	Turns coolant off		X	X
M10	Turns on or initiates automatic clamping			X
M11	Turns off or initiates unclamping			X
M12	Code used to synchronize multiple axes		X	
M13	Turns on the spindle in the clockwise (CW) direction AND turns on coolant	X		X
M14	Turn on spindle in the counter clockwise (CCW) direction AND turns on coolant	X		X
M15	Rapid traverse in the positive direction	X	X	X
M16	Rapid traverse in the negative direction	X	X	X
M17-M18	Unassigned	X	X	X
M19	Stops spindle in a specific angular position			X
M20-M29	Permanently unassigned			
M30	Indicates end of program and resets the machine		X	
M31	Interlock bypass	X		X
M32-M35	Unassigned			
M36-M39	Permanently unassigned			
M40-M46	Used to initiate gear changes when used, otherwise unassigned	X		X
M47	Returns to start of program to execute program again			
M48	Cancels M49	X		X
M49	Deactivates a manual spindle or feed override and returns to the value specified in the program	X		X
M50-M57	Unassigned			
M58	Cancels M59	X		X
M59	Holds RPM of spindle constant	X		X
M60-M89	Unassigned			X
M90-M99	Resereved for user			

Figure 4-28 M-codes

Detailed explanations of some of the basic miscellaneous codes follow. Many of the other M-codes listed in Figure 4-28 will be discussed in subsequent chapters. The codes that follow are explained as they relate to the three-axis milling process.

M03 − Spindle On: Clockwise
Function
The M03 code instructs the machine controller to turn the machine spindle on, in the CW direction. The M03 code is modal.

Input Format
N___ S___ M03
Terminology
N___Sequence number
S___Spindle speed
M03 M Code turning spindle on in the CW direction.

Explanation
The M03 code turns the spindle on in the CW direction at the speed given by the S word. This code is issued at the beginning of the program setup section. It is good practice to move the tool to a "safe" or "home" position away from any obstacles prior to turning the spindle on. If the command block in which the M03 is issued contains move commands, the controller will execute the moves first, then turn on the spindle.

Example Program
All of the programs listed thus far have examples of the M03 code. Command block N40 of Figure 4-27 is a typical example. Observe that prior to the issuing of the M03 code, the instruction the cutter gives in command block N30 rapid moves the tool to a safe position (0,0,1). Turning the spindle on in this position ensures that it will be fully up to speed prior to beginning the material removal process.

M05 − Spindle Stop
Function
The M05 code instructs the machine controller to stop the machine spindle. The M05 code is modal.
Input Format
N___ M05
Terminology
N___Sequence number
M05 M Code turning off and stopping the spindle.

Explanation

The M05 code turns the spindle off. This code normally appears at the end of the program in the program shutdown section. If the command block in which the M05 is issued contains move commands, the controller will execute the moves first, then stop the spindle.

Example Program

All the programs listed thus far have examples of the M05 code. Command block N310 of Figure 4-27 is a typical example.

M06 – Tool Change

Function

The M06 code instructs the machine controller to stop all machine operations and change the tool. The M06 code is not modal.

Input Format

N___ T___ M06

Terminology

N___Sequence number

T___Tool number

M06 M-Code instruction requesting a tool change.

Explanation

When an M06 code is encountered in the program, the machine stops all machine operations and changes the tool to the one specified by the T word. For example, in milling, the controller will stop spindle rotation and table movement, move the spindle to the tool changing position, change the tool and return to the state (location, spindle on, feed rate, etc.) of the machine prior to the tool change. The code not only appears at the beginning of the program in the program startup section, but can be issued anywhere within the program.

Example Program

All of the programs listed thus far show examples of the M06 code. Command block N20 of Figure 4-27 is a typical example. In this case the controller changes to tool number 22.

M30 - Program End, Reset Machine

Function

The M06 code instructs the machine controller that the program is completed and to reset the machine to system defaults. The M30 code is not modal.

Input Format

N___ M30

Terminology

N___Sequence number

M30 M-Code instruction telling the machine controller that the program is complete and to reset the machine.

Explanation

The M30 command is always located in the program shutdown section and is typically the last command issued. It is used to inform the controller that no other data are coming and that the machine should reset programming parameters (feed rates, spindle speeds, etc.) back to the system default.

Example Program

All the programs listed thus far show examples of the M30 code. Command block N320 of Figure 4-27 is a typical example.

4.2.7 Format Classification Sheet

All of the code discussed thus far is in compliance with ANSI/EIA RS-274-D and ISO 6983 standards. However, as mentioned, not all machine controllers follow the standards exactly and may use codes in ways not discussed in the previous section. Additionally, different machine controllers have various capabilities such as point-to-point positioning control, contouring control, or a combination of both. Therefore, when developing code for a specific machine it is important one know beforehand its control capabilities, available codes, and code format for the program. The machine *format classification sheet* was specifically developed as a quick reference for this information. It provides all the format requirements of a control system for a specific machine. It specifies type of machine, format classification shorthand, format detail, available G-codes, available M-codes, a range of values for the codes available, any code not covered in the standards, and any other unique aspect of the control system. It is typically found in the machine's user manual. Figure 4-29 is an abbreviated format classification sheet for a Fadal vertical machining center.

Format classification shorthand, listed in the format classification sheet, is a 9- or 10-digit code established by the ANSI/EIA RS-274-D and ISO 6983 standards that provides detailed information about a machine. Each character, with its location, has a specific meaning as shown below, per the ANSI/EIA RS-274-D standard:

- First character: Designates the type of control system, as follows:

    P – variable block format positioning control system

    C – variable block format contouring control system

    D – variable block format positioning and contouring control system.

- Second character: A digit specifying the number of motion dimension words.

- Third character(s): A single or double digit specifying the number of other words available in addition to the motion dimension words. This character is followed by a decimal point.
- Fourth character: A digit designating the type of dimensional data used by the control system as follows:
    - 1 – signifies absolute dimensions, no operational signs
    - 2 – signifies absolute dimension using + and – signs
    - 3 – signifies incremental dimensions using + and – signs
    - 4 – signifies absolute or incremental dimensions using + or – signs with incremental dimensions only
    - 5 – signifies absolute or incremental dimensions using + or – signs with both types of coordinates.
- Fifth character: A digit specifying the number of digits to the left of the decimal point in a dimension word for the longest axis of the machine.
- Sixth character: A digit specifying the number of digits to the right of the decimal point in a dimension word for the longest axis of the machine. This character is followed by a decimal point.
- Seventh character: A digit specifying the number of motion control channels.
- Eighth character: A digit specifying the number of numerically controlled machine axes.
- Ninth character: A digit designating type of numerical data as follows:
    - 1 – signifies decimal point programming only
    - 2 – signifies full field programming (leading and trailing zeros)
    - 3 – signifies leading zeros required
    - 4 – signifies trailing zeros required
    - 5 – signifies a combination of the above. Refer to the format classification sheet for detailed information.

A good example of format classification shorthand is shown in Figure 4-29. The shorthand is listed for the machine as D617.524.665. The interpretation of each shorthand character is shown in the figure.

The format detail specifies the words, the length of the words, and the order in which they appear in a block. The format detail shown in Figure 4-29 for the inch mode increment system can be interpreted as follows:

N5.4 – This indicates the sequence number can have a total of 9 digits, 5 digits immediately following the N character and 4 additional digits after a decimal point.

G2.1 – This signifies that preparatory functions consist of 2 digits immediately following the G character and 1 additional digit following a decimal point. Many machine controllers allow this format to obtain more codes than the amount specified by the standard.

Fadal   **User Manual**

    c. The program contains a duplicate O word: the new program becomes active; the old program is deleted.

Format Classification Sheet

Reference: Conforming to ANSI/EIA RS-274-D standard.

Machine   Vertical Machining Center (VMC).

Format Classification Shorthand   D617.524.665

D variable block format contouring/positioning system

6 motion dimension words (X, Y, Z, A, B, C)

17 other words (E, D, O, N, M, F, G, S, R, H, L, P, Q, T, I, J, K)

5 absolute or incremental data, depending on mode of operation

2 digits to left of decimal point in longest axis (3 metric)

4 digits to the right of the decimal point in longest axis (3 metric)

6 motion control channels (X, Y, Z, A, B, C)

6 numerically controlled machine axes (X, Y, Z, A, B, C)

5 decimal point programming: if no decimal point, defaults assumed

Format Detail   Inches Mode Increment System

N5.4
G2.1
X+3.4 Y+3.4 Z+3.4 I+3.4 J+3.4 K+3.4 B+3.4 R+3.4 Q+3.4
A+4.3
C+5.1
M2.1 H2 T2 D2
F4.2
S5.1, O4
L4 P4
MILLIMETERS MODE
N5.4
G2.1
X+3.3 Y+3.3 Z+3.3 I+3.3 J+3.3 K+3.3 B+3.3 R+3.3 Q+3.3

Figure 4-29 Fadal format classification sheet

Fadal   User Manual

A+4.3
C+5.1
M2.1 H2 T2 D2
F4.2 S5.1 L4 P4 O4

G Function Codes   0, 1, 2, 3, 4, 5, 8, 9, 10, 15, 16, 17, 17.1, 17.2, 18, 19, 20, 21, 28, 28.1, 29, 31, 31.1, 40, 41, 42, 43, 44, 45-48, 49, 50, 50.1, 51, 51.1, 51.2, 51.3, 52, 52.1, 53, 54-59, 66-71, 73-76, 80-89, 90, 91.1, 91.2, 92-94, 98, 99

M Function Codes   0, 1, 2, 3, 3.1, 3.2, 4, 4.1, 4.2, 5, 6, 7, 7.1, 8, 8.1, 9-16, 17-20, 30, 31, 32, 32.1, 33, 33.1, 41-43, 45-47, 48, 48.1, 48.2, 48.3, 49, 49.1, 49.2, 49.3, 60-69, 80, 81, 90-93, 94, 94.1, 95, 95.1, 96, 97, 98, 99

- 2 digit BCD output (standard)
- 2 decades of relay output (optional)
- The use of a minus sign (M-60) will perform the function to be accomplished after motion. This usage applies to M60 through M69 only.

F Function Range   The F word is used to define the feed rate. It is modal and remains in effect for G1, G2, and G3 moves until another F word is used in the program or in the MDI mode. See G93 and G94 in the index for more information.

- 1 to 150 percent feed rate override
- .01 to 375 inches per minute
- 1 to 3810 millimeters per minute
- .6 to 9000 degrees per minute (72 to 1)
- .6 to 7992 degrees per minute (90 to 1)
- .6 to 3960 degrees per minute (180 to 1)
- .6 to 1980 degrees per minute (360 to 1)

S Function   The S word represents the PRM to be used when the spindle is turned on with the M3, M4, or SPINDLE ON/OFF with the shift button combination. The lower belt range RPM amounts can be used from the upper belt range by using a .2 at the end of the interger. For example, S1000.2 would result in 1000 would result in a belt range to the lower range.

WARNING: The S word is modal and will remain in effect until another S word is used in auto or the MDI mode.

VMC 7.5 HP   (Manual Belt)

75 to 3750 Top belt range

Figure 4-29 continued

Fadal   User Manual

75 to 7500 Bottom range

VMC 15 HP

40 to 2500 Top belt range

150 to 10000 Bottom range

EXAMPLE: VMC 15 HP (Auto High/Low)

75 to 2500 Top belt range, S.1 used to override belt to Top belt range

2501 to 10000 Bottom range, S.2 used to override belt to Bottom belt range

VMC High Torque (Auto Hi/low)

40 to 2500 Top belt range, S.1 used to override belt to Top belt range

2501 to 10000 Bottom range, S.2 used to override belt to Bottom belt range

VMC High Speed Head (Single Range)

300 to 15000 Single range

T Function Code   The T word specifies turret location selection. The number will range from 1 through 30 depending on the available turret locations in the tool changer. The T word is usually used in conjunction with the M6 tool change M function. It would appear as an M6T# on a line by itself (See M6 for details). However the T word is modal and can be used on any line prior to the M6 code.

Note: The use of a minus sign with the T word (T-5) will rotate the turret until the pocket is located directly opposite from the spindle. This might be used to rotate long tools in the turret to some location to avoid hitting a part during program execution. At the next tool change the turret will rotate automatically back to its original position.

Note: Do not use the T-# with an M6.

D Function Code   The D word specifies which diameter or radius offset to use from the tool table for cutter radius compensation. It ranges from 1 through 99. This code is not necessary in Format 1, but may be used for cutter diameter override.

H Function Code   Programming Format 1:

Figure 4-29 continued

X+3.4 Y+3.4 Z+3.4 I+3.4 J+3.4 K+3.4 B+3.4 R+3.4 Q+3.4 A+4.3 C+5.1 – The details shown here specify the dimension word addresses used and the corresponding numerical format. The + following many of the dimension words specifies that + (or –) signs are used. The 3.4 signifies 3 digits to the right of the decimal point and 4 digits to the left.

M2.1 H2 T2 D2 F4.2 S5.1 O4 L4 P4 – These details show the non-dimension words and the corresponding numerical format.

Also shown in Figure 4-29 are the G-codes, M-codes, and additional information for the F, S, T, and D function codes. All of this information is vital to have during the development of the program code; one should always refer to the format classification sheet and the user manual prior to developing any program code for an unfamiliar machine.

4.3 Cutting Parameters

The rate at which the selected cutting tool can remove material from the workpiece is determined by the cutting parameters of the process. The cutting parameters of milling and turning operations include depth of cut, cutting speed, and cutting feed rate. These parameters, in conjunction with the workpiece material, dictate the machine power required and the life of the tool.

4.3.1 Depth of Cut

The depth of cut is the maximum amount of material the cutter can remove with each pass. Its value depends on the machine power available, the rigidity of the workpiece fixturing, and the rigidity of the cutting tool.

For a milling program, depth of cut is expressed as the z-coordinate of a linear move. For example, again consider the G-code shown in Figure 4-25 that yields the product in Figure 4-26. The slot is produced in four passes of the cutter with a depth of cut of 0.250 inch for each pass. This is shown in command blocks N110 to N290. Recall that the surface of the workpiece has a z-coordinate value of zero. Thus, whenever a negative z-coordinate appears, the cutter is below the surface of the workpiece. In command block N120, the cutter is linearly moved into the workpiece to a depth of –0.25 inch. Command block N130 then mills the first pass of the slot. Therefore, the depth of cut of the first pass is 0.25 inch. This process is then repeated in the second pass listed in command blocks N150 to N190. The depth of cut of this pass can be determined by subtracting the z-coordinate of the prior pass (–0.25) from the z-coordinate of the current pass (–0.50). So, the depth of cut for the second pass is:

$$\begin{array}{lr} \text{second pass } z\text{-coordinate:} & -0.50 \\ \text{minus first pass } z\text{-coordinate:} & -\underline{(-0.25)} \\ & -0.25 \end{array}$$

This process is repeated a four times until the desired 1-inch slot depth is reached.

4.3.2 Cutting Speed

Cutting speed (*V*) is the velocity at which the tool moves past the workpiece, expressed in feet per minute (fpm or ft/min) or meters/minute (m/min). However, the cutting speed must be converted into rotational speed of the cutter for milling operations or rotational speed of the workpiece in the case of turning operations. In either case the formula used is

$$N = V / \pi D,$$

where, for inches, N = rotational speed (rev/min or rpm), V = cutting speed (ft/min or m/min), and D = diameter of the tool for milling or diameter of the workpiece for turning (ft or m). Caution: Milling cutter diameters D are conventionally listed in inch (or millimeter) units, so they must be converted to feet (or meters) to keep units consistent with those of cutting speed *V*.

The cutting speed is dependent on the workpiece material and hardness, the type of operation, and the material of the cutter. Figure 4-30 shows cutting speeds and tooth feeds for some common materials. For perhaps the most comprehensive tabulation available refer to *Machinery's Handbook,* 28th ed.

Material	Cutting Speed (V) feet/minute	Feed per Tooth (f_t) inches/tooth
Plain Low Carbon Steels	85 - 125	0.004
Plain Medium Carbon Steels	70 - 100	0.004
Plain High Carbon Steels	70 - 85	0.004
Aluminum	200 - 300	0.005
Brass	200-250	0.005
Bronze	200-250	0.005
Stainless Steel Free Machining	60 - 80	0.003
Thermoset Polymers	150 - 300	*
Thermoplastic Polymers	300 - 500	*

* - Utilize as high a feed as possible without melting or degrading material

Figure 4-30 Cutting speeds and tooth feeds

In the CNC program the cutting speed parameter is entered by the S word as the spindle rotation as determined from the above equation. In the program code the letter S is immediately followed by the desired spindle speed in units of revolutions per minute (rpm). The S word is used in conjunction with miscellaneous function commands M03, which turns the spindle on. The S word is modal. Examples of the S word in use are evident in all program code figures listed in this chapter.

4.3.3 Feed Rate

Feed rate (f_r) is the speed at which the cutter moves into the workpiece during the material removal process. Many factors should be considered in determination of feed rate; these include the type of cutting tool, cutting tool material, cutting tool rigidity, depth of cut, hardness and material of the workpiece, rigidity of the workpiece, desired surface finish, and available machine power. For milling operations and occasionally turning operations, feed rate is expressed as inches per minute (ipm) or millimeters per minute. However, turning operations most often express feed rate in inches per revolution (ipr). To convert from ipm to ipr, use the equation:

$$f_{r(\text{ipr})} = f_{r(\text{ipm})} / N.$$

where N = rotational speed of the spindle (rpm).

To calculate the required feed rate for a particular milling operation, the following formula is used:

$$f_r = f_t\, n_t\, N$$

where, for inch units, f_r = feed rate (ipm), f_t = feed per tooth (in./tooth or ipt), n_t = number of teeth on the cutter, and N = rotational speed of the spindle (rpm). Values for feed per tooth depend on workpiece material, workpiece hardness, type of operation, and cutting tool material. These values are well-documented in table form in many texts. For perhaps the most comprehensive tabulation available, refer to *Machinery's Handbook*, 28th ed. Figure 4-30 also provides a sample listing of feed per tooth values for high-speed steel (HSS) tools and some common materials.

The feed rate, as determined by the above equation, is entered into the CNC program with the F word. In the program code the letter F is immediately followed by the desired feet rate in inches per minute. The F code is required whenever any type of interpolation move is issued (G01, G02, G04, etc.). The F word is modal. If no value is issued, the machine controller will utilize a system default value. Examples of the F word in use are evident in all program code figures listed in this chapter.

4.3.4 Cutting Parameter Calculations

The following examples demonstrate how spindle speeds and feed rates are calculated.

Example 4.1

A 4-flute end mill made of high-speed steel is to be used to mill a slot into aluminum. The end mill has a 0.25-in. diameter. Using the feed and speed data listed in Figure 4-30, calculate the spindle speed and feed rate settings for the milling machine.

Solution

Since the feed rate calculation requires the spindle speed, we will calculate the spindle rotational speed first.

From Figure 4-30, the cutting speed for aluminum is 200 to 300 feet per minute. For this application we will use an average cutting speed of 250 feet/min as a starting point. The diameter of the cutter is 0.25 inch. Note, for consistency of units the diameter of the cutter will be converted into feet. Therefore:

$$V = 250 \text{ ft/min}$$

$$D = (0.25 \text{ in})(1 \text{ ft/12 in}) = 0.0208 \text{ ft}$$

Once the cutting speed is known and the diameter of the cutter's units are converted into feet, the spindle RPM can be calculated from the formula given in Section 4.3.2:

$$N = V/\pi D = 250 \text{ ft/min} / 0.0208\pi \text{ ft} = 3826 \text{ rpm (revolutions per minute)}.$$

To find the feed rate, the feed per tooth for aluminum is used; it is given in Figure 4-30 as 0.005 inches per tooth(ipt). For end mills the number of teeth on the cutter corresponds to the number of flutes. Thus, for a 4-flute end mill the number of teeth is four, and so

$$f_t = 0.005 \text{ ipt}, \ n_t = 4 \text{ teeth}, \ N = 3826 \text{ rpm}$$

and

$$f_r = f_t \, n_t \, N = (0.005 \text{ ipt})(4 \text{ teeth})(3826 \text{ rev/min}) = 76.5 \text{ ipm}$$

Example 4.2

A 4-flute end mill made of high-speed steel is to be used to mill a slot into plain medium carbon steel. The end mill has a 0.25-in. diameter. Using the feed and speed data

listed in Figure 4-30, calculate the spindle speed and feed rate settings for the milling machine.

Solution

From Figure 4-30, the cutting speed for plain medium carbon steel is 70 to 100 fpm. For this application we will again use an average cutting speed, 85 fpm as a starting point. The diameter of the cutter is 0.25 inch. Using the same calculation method as the previous example:

$$V = 85 \text{ ft/min}$$
$$D = (0.25 \text{ in.})(1 \text{ ft/12 in.}) = 0.0208 \text{ ft.}$$

Once the cutting speed is known and the diameter of the cutter's units are converted into feet, the spindle speed (rpm) can be calculated from the formula given in Section 4.3.2

$$N = V / \ \pi D = 85 \text{ ft/min} / 0.0208\pi \text{ ft} = 1301 \text{ rpm.}$$

For the feed rate, the feed per tooth for this steel is given in Figure 4-30 as 0.004 ipt. For end mills the number of teeth on the cutter corresponds with the number of flutes. Thus, for a 4-flute end mill the number of teeth is four, and so

$$f_t = 0.004 \text{ ipt, } n_t = 4 \text{ teeth, } N = 1301 \text{ rpm}$$

and

$$f_r = f_t \, n_t \, N = (0.004 \text{ ipt})(4 \text{ teeth})(1301 \text{ rpm}) = 20.8 \text{ ipm.}$$

In addition to demonstrating how to calculate speeds and feeds for a milling operation, these two examples highlight how the material being cut significantly influences the *cutting parameters*. Note how the aluminum material can be cut significantly faster than the plain medium carbon steel, utilizing the same type of cutter. Also, these calculations yield the starting point for speeds and feeds. The machine power, rigidity of the system, and the desired tool life must also be considered. Adjustment during the trial running of the program should be anticipated. Maximizing feed rate and depth of cut to speed of material removal is always desirable.

4.4 Program Organization

In Section 4.2 the basic preparatory and miscellaneous function codes were discussed in great detail. In this section, the way these codes are organized in the program is addressed. Recall that the code is organized into three sections: *program setup*, *material removal*, and *system shutdown*. Section 4.1.2 discussed the corresponding content of each of the sections. With that section as a guide, we organize the preparatory and miscellaneous codes into the appropriate program sections.

4.4.1 Program Setup Codes

The preparatory codes that will typically be listed in this section of the program include:

- **G90/G91** – These codes are used to establish the type of coordinates to be used in the program initially.
- **G70/G71** – These codes specify the coordinate units (inches or millimeters) to be used in the program.
- **G00** – Rapid moves are used to position the tool and spindle at specific locations prior to turning the spindle on or removing material.
- **Other G-Codes** – Examples of other G-codes that may appear in this section are listed below.
    - G17–G18, circle interpolation plane selection
    - G32–G35, lead cutting information
    - G40, cutter compensation cancel
    - G54–G59, datum shifts
    - G80, cancel fixed cycles
    - G93–G96, feed rate unit information
    - G97, spindle speed unit information

The miscellaneous and other non-dimension codes that will normally appear in this section of the program include:

- M06 – Used to request a tool change so that the appropriate tool is loaded.
- T word – Used in conjunction with M06 to specify the location of the tool to be used. The T word is a number that indicates the location of the tool in the machine's tool turret. Not all machines have tool turrets, thus, both M06 and the T word may not be required. Note the T word is modal so it may appear before an M06 is issued.
- M03/M04 – Used to turn the spindle to on position. The M03 and M04 commands usually follow a rapid move (G00), which will position the spindle at a safe location to turn the spindle on.
- S word – Used in conjunction with M03/M04 to set the spindle rotational speed in revolutions per minute. Note the S word is modal so it may appear before an M03 is issued.
- F word – Specifies the feed rate for interpolation moves. Even though interpolation moves (G01, G02, and G03) will not appear in this section of the program, the initial feed rate can be listed. This is permissible because the F word is modal.
- M07 and M08 – Codes used to turn coolant flow on.
- M10 – Code to initiate automatic clamping (when available).

Other information that may appear in this section includes the program number and/or some descriptive explanation of the program. For some controllers the O word is used to identify the program number. Typical formatting is the letter O followed by a four-digit number. An example would be "O0011." This code would be listed on the first command block and would indicate that the next program to come is program number 11. The format classification sheet of the machine for which the program is being written should be consulted for available options.

The program setup section is always the first section of the program. However, mini-setup sections may appear in the body of the program. An example of when this would be required is if a tool change is needed in the middle of the material removal section.

4.4.2 Material Removal Codes

Material removal codes consist primarily of rapid and interpolation moves along with the corresponding dimension words. These codes instruct the machine controller to move the tool along a specific tool path to make the product. The G-codes that appear in the material removal section of the program include:

- G00 – Rapid moves to position the tool for the next cutting pass.
- G01 – Linear interpolation moves for straight line cuts.
- G02/G03 – Circle interpolation moves for cutting arcs and circles.
- Other interpolation moves such as helical and parabolic.
- G90/G91 – For switching between absolute and incremental coordinates as needed.

These are the primary codes that will appear. Note that other more advanced G-codes may also appear in this section of the program. In general, few if any M-codes will appear in this section. Where they do appear they are usually part of another mini-program setup section preparing the machine to execute the next material removal section.

4.4.3 System Shutdown Codes

The system shutdown section of the program is the smallest section. All that is needed in this section are rapid moves to position the tool in a safe position to allow convenient access to the workpiece and miscellaneous codes to turn off functions initiated in the program setup section. Also, an M-code to reset the machine for the next program is typically used. Therefore, the codes used in this section include:

- G00 – Rapid moves to position the tool in a safe position, out of the way of the workpiece.
- M05, M09, and M11 – Miscellaneous codes to turn off the spindle, coolant flow, and automatic clamping, respectively.
- M30 – End of program and reset the machine. This is the last command issued.

These are the primary codes that appear in system shutdown. Different machines may use slightly different code. The format detail sheet should always be consulted for detailed information about a specific machine.

4.5 Programming Process

Section 4.1.1 reviewed the CNC programming process steps. The ten steps listed encompass the tasks required to completely develop a CNC program. The following three sections focus on steps 3, 4, and 5: *developing the program tool path, determining program coordinates,* and *writing the program of instructions*. These are perhaps the three most critical steps of the process. They involve interpreting product geometry listed on the product drawing and translating that information into the program of instructions. As with any translation of one language to another, accuracy of the translation is critically important. Therefore, in the following sections, detailed instructions on how to perform this translation accurately will be addressed.

4.5.1 Tool Path/Process Flow Development

Process flow can be defined as the sequence or flow of the process steps to produce a product. For the CNC machining process this sequence is analogous to the path the tool must follow to make the product. Hence, this step is often called *tool path development*. The development of the tool path occurs after the initial evaluation of the product drawing, after the machine tooling and fixturing has been determined, and after the cutting parameters have been selected. It begins with a return to the product drawing.

Consider Figure 4-31. This figure shows a product with two holes and slot that will be produced on a milling machine utilizing a 4-flute, 0.25-in. diameter end mill cutter. Figure 4-32 shows how the part will be located in the machine and the layout of the workpiece coordinate system. Note the location of the PRZ. Recall that this is the origin of the workpiece coordinate system.

The first step in developing the tool path involves visualizing how the cutting tool will move to produce each of the product features. For the workpiece shown in Figure 4-31 the tool moves to the first hole, drills it, moves to the second hole, drills it, moves to the beginning of the slot, moves into the work, and mills the slot. Finally, the tool can be returned to its safe position, clear of the workpiece. This tool path can then be sketched on the drawing as shown in Figure 4-33.

Figure 4-31 Programming example drawing

Figure 4-32 Programming example PRZ and coordinate system

The second step in developing the tool path is to label the location or starting and ending point of each of the workpiece features along the tool path. Note that it is only necessary to label the point on the top surface of the workpiece. Also, as is often the case, a feature on the part may actually consist of multiple simpler features. For this example the slot actually consists of two features, a straight slot and a curved slot. Hence, Figure 4-34 shows the workpiece with the appropriate labeling. The completed, labeled tool path is shown in Figure 4-35.

Figure 4-34 Programming example position labeling

Figure 4-35 Programming example tool path

4.5.2 Developing Program Coordinates

Upon completion of the development of the tool path, the program coordinates can be determined. Developing the program coordinates involves identifying the coordinates, in the workpiece coordinate system, of each of labeled points on the tool path. These data go into a program coordinate sheet. The program coordinate sheet for the tool path listed in Figure 4-35 is shown in Figure 4-36. Note that the program coordinate sheet shows only the x- and y-coordinates of the point. This is because the z-axis coordinate is a separate tool move from the (x,y)-plane moves.

4.5.3 Program of Instructions Development

The development of the program of instructions is the point at which the actual translation of the process flow to a G-code program occurs. The tool path/process flow is broken down into the individual steps the CNC machine will make and written out in English on a program sheet. The moves are then translated from English into the appropriate word address format codes. A blank program sheet is shown in Figure 4-37.

Program Coordinate Sheet		
Point	X	Y
0	0	0
1	0.5	0.5
2	0.5	1.5
3	1.5	1
4	2.03	1.53
5	2.03	0.47

Figure 4-36 Programming example (Figure 4-35) program coordinate sheet

Figure 4-38 shows the completed program sheet for workpiece of Figure 4-31. Note how the process flow/tool path description is listed on the left side of the program sheet. Either immediately after each step of the process or once the flow is completely written out, the English phrases can be translated into G-code. As noted in Figure 4-38, the programming convention of separating the program into three sections was followed.

Note that all of the tool moves in the material removal section separate the *x*- and *y*-axis moves from the *z*-axis moves. This is the preferred convention unless the feature geometry dictates that all three axes be specified at the same time. An example of this exception is the slot shown in Figure 4-11. Rapid moves in the program setup and system shutdown section often occur in three dimensions, simultaneously. This is permissible as long as there is no danger of striking clamps or fixturing that is used to locate and hold the workpiece.

To provide clarity for beginning programmers, modality was not strictly followed. For example, in command blocks N50 and N60, repeating the G00 codes is not necessary. The G00 issued in command block N30 would stay active until the command block N70, when a G01 is issued. (How else might the program code change if modality were strictly adhered to?)

Program Sheet													
Program Number:		Part Name:									Sheet No.:		
Programmer:											Date:		
Process Flow/Tool Path Description	N	G - Code	Axis Coordinate			Center of Arc			Radius of Arc	F	S	T	M-Code
			X	Y	Z	I	J	K	R				

Figure 4-37 Blank program sheet

Program Sheet

| Program Number: | 1001 | | Part Name: | | Block | | | | Sheet No.: | 1 |
| Programmer: | XYZ | | | | | | | | Date: | - |

Process Flow/Tool Path Description	N	G - Code		Axis Coordinate			Center of Arc			Radius of Arc				
				X	Y	Z	I	J	K	R	F	S	T	M-Code
Set absolute coordinates and inch units	N10	G90	G70											
Change the tool	N20												T10	M06
Move to a safe position, 1 inch above work piece	N30	G00 *PROGRAM SETUP SECTION*		X0	Y0	Z1								
Turn on the spindle CCW and set the RPM's and feed rate	N40										F76	S3800		M03
Move to a position directly above the work piece	N50	G00				Z0.1								
Move to position 1, just above work piece	N60	G00		X.5	Y.5									
Drill 1st hole or linear move into work piece	N70	G01				Z-.25								
Rapid move out of work piece	N80	G00				Z0.1								
Move to Position 2, just above work piece	N90	G00			Y1.5									
Drill 2nd hole or linear move into work piece	N100	G01 *MATERIAL REMOVAL SECTION*				Z-.25								
Rapid move out of work piece	N110	G00				Z0.1								
Move to position 3, just above work piece	N120	G00		X1.5	Y1.0									
Move into workpiece depth of slot	N130	G01				Z-.25								
Linear move to position 4	N140	G01		X2.03	Y1.53									
Circle interpolate arc in CW direction	N150	G02		X2.03	Y.47					R.75				
Rapid out of work	N160	G00				Z0.1								
Move to safe position	N170	G00		X0	Y0	Z1								
Shutdown spindle	N180	*SYSTEM SHUTDOWN SECTION*												M05
End Program	N190													M30

Figure 4-38 Completed program sheet

In the material removal section, command block N150 issues a circle interpolate command to cut the curved slot. Since the arc is less than 90 degrees, the second formatting option discussed in Section 4.2.5 is used where only the radius of the arc is required. (What would the I and J coordinates be if the first formatting method were used?)

4.6 Turning Programs

Thus far the discussion of word address format programming has focused on developing code for the milling process. However, developing code for turning or lathe processes is just as important. G-code in terms of milling processes was discussed initially because it is the author's experience that students often find it easier to develop G-code programs for milling applications. It appears to be more intuitive. That being said, it is now time to turn our attention to the turning process.

Developing G-code programs for turning applications is identical to developing the code for milling applications. All the programming steps discussed in Section 4.1.1 are executed in the same manner. However, the workpiece fixturing, the tooling, the coordinate system and the PRZ are vastly different. Figure 4-39 highlights these differences.

	Milling	Turning
Work Piece Fixturing	Vise common. Work piece stationary	3 jaw chuck common. Work piece rotates
Cutting Tooling	Multi-point tools: endmills, face mills and drill bits. Cutting tool rotates	Single point tools, reamers and drill bits.
Coordinate System	X-Y-Z standard with fourth and fifth axes often added.	X-Z
PRZ	Lower right hand corner on top of work piece.	Right face on the center of the part.

Figure 4-39 Comparison of milling and turning from a programming standpoint

Consider the first row of Figure 4-39, workpiece fixturing. The fact that the workpiece rotates introduces some challenges with spindle speeds and feed rates. Recall that, for turning, spindle speed is a function of the diameter of the workpiece. As the outside diameter is machined, it gets smaller, thereby affecting the cutting speed. Hence, spindle speed may need to be adjusted as more cuts are taken on the outside diameter. Additionally, changes to spindle speeds affect feed rates, which may also have to be adjusted.

Row 2 of Figure 4-39 indicates that lathes use single point tools. There are many different types of single point tools designed to perform specific operations on only certain sides of the workpiece (facing versus profiling). Hence, in some aspects, tooling is slightly more complicated than during milling.

The coordinate system and the PRZ are perhaps the most significant differences. Since lathes use the x,z-coordinate system, circle interpolation codes are affected. Thus, when one specifyies the arc center for a G02 or G03 code, word addresses I and K are required. Additionally, the side from which the tool approaches the workpiece influences whether CW or CCW code is used. This is demonstrated in Figures 4-40 and 4-41. In Figure 4-40, the tool approaches from the bottom, thus, a CCW circle interpolation code (G03) is used to produce the fillet. In Figure 4-41, the tool approaches from the top, thus a CW circle interpolation code (GO2) is used to produce the same fillet. Both figures show the x,z-coordinate system and the location of the PRZ on the turned workpiece. Note that when the tool is cutting the workpiece, z will be negative. Also, when the tool approaches from above as in Figure 4-41, x will be moving in the negative x direction, thus getting smaller. Therefore, as is the convention, negative directional moves indicate the tool is moving into the workpiece.

Figure 4-40 Turning example—tool approaching from bottom

Another important concept to grasp is that of *roughing* versus *finishing cuts*. Turning programs typically require roughing cuts to remove the bulk of the material. This type of cut is so named because the object of the cut is to remove as much material as possible with minimal concern for the surface finish. Thus, after the cut, the surface will be in a "rough" condition, hence the name. However, for the finished product, the surface texture

needs to be in the state specified on the product drawing. Therefore, a finishing cut will bring the workpiece to the desired profile and surface texture. The part will be "finished" after this cut. The general procedure in turning is to take a series of roughing cuts to bring the workpiece down to within approximately 0.010 in. of its finished size as quickly as possible, after which a finishing cut is performed to bring the workpiece to its final size.

Now that some of the differences between turning programs and milling programs have been discussed, a simple turning example can be presented.

Figure 4-41 Turning example—tool approaching from top

4.6.1 Turning Example

Figure 4-42 Turning example

Consider the workpiece in Figure 4-42. It is a relatively simple turned part made of aluminum. Note that the dimensions are in millimeters. The important features are the 30-degree tapered face and a 25-mm fillet at the transition of the taper and the 50-mm diameter. For this example it is assumed that:

- The product drawing has been thoroughly evaluated; the part will be turned from a 100-mm diameter aluminum bar.
- The specific lathe on which the part is to be produced has been selected.
- A 3-jaw chuck will be used to hold the workpiece.
- The tool will be a right-hand tool approaching the workpiece from above.

The cutting parameters have been calculated and are as follows:

- Depth of cut
- Roughing ~ 3 mm
- Finishing ~ 1 mm
- Speed – 1520 rpm
- Feed – 180 mm/min.

The next step, step 4 of the CNC programming process, is to develop the tool path. As already mentioned, for most machining operations the tool path can be broken down into roughing and finishing cuts. Using the approach outlined in Section 4.5.1, the tool path was developed and is shown in Figure 4-43.

Figure 4-43 Turning example tool path

The dashed lines of Figure 4-43 show the starting size of the bar. The dashed arrows represent rapid moves; the phantom arrows represent linear moves for the roughing passes; the solid arrows represent linear moves for the finish pass. The roughing cuts consist of three passes with a depth of cut of 3 mm each and one pass with a depth of cut of only 2.5 mm. The last pass is only 2.5 mm so that 1 mm of stock is left for the finishing pass. The labeled points of the roughing passes, starting with the letter R, represent tool point destinations. The points for the finishing pass are also shown and are labeled with an F.

The next step, step 5 of the CNC programming process, is to determine the program coordinates. Figure 4-44 lists the program coordinates of the tool destination points.

Program Coordinate Sheet		
Point	X	Z
R0	30	12.00
R1	25	1
R2	22	1
R3	22	-42.80
R4	25	-48
R5	19	1
R6	19	-37.61
R7	16	1
R8	16	-32.41
R9	13.5	1
R10	13.5	-24.87
R11	15.11	-30.86
F1	12.5	0
F2	12.5	-25
F3	14.175	-31.25
F4	25	-50

Figure 4-44 Turning example program coordinates

Program Sheet

Program Number:	1				Part Name:						Sheet No.:		1	
Programmer:			Kandray		Lathe Example5.6.1						Date:		xx/xx/xx	

Process Flow/Tool Path Description	N	G - Code		Axis Coordinate			Center of Arc			Radius of Arc	F	S	T	M-Code
				X	Y	Z	I	J	K	R				
Set absolute coordinates and metric units	N10	G90	G71											
Change the tool	N20												T1	M06
Rapid move to a safe position, R0	N30	G00		X60		Z24								
Turn on the spindle CW and set the RPM's and feed rate	N40										F180	S1520		M02
Rapid move to a position slightly off the part (R1)	N50	G00		X50		Z1								
Linear Move into work for 1st cut (R2)	N60	G01		X44										
Linear move to R3	N70					Z-85.6								
Linear move to R4	N80					Z-96								
Rapid back to R2 for the next cut	N90	G00		X44		Z1								
Linear into work - R5	N100	G01		X38										
Linear move to R6	N110					Z-75.22								
Linear move to R3	N120			X44		Z-85.6								
Rapid back to R5 for the next cut	N130	G00		X38		Z1								
Linear into work - R7	N140	G01		X32										
Linear move to R8	N150					Z-64.82								
Linear move to R6	N160			X38		Z-75.22								
Rapid back to R7 for the next cut	N170	G00		X32		Z1								
Linear into work - R9	N180	G01		X27										
Linear move to R10	N190					Z-49.74								
Linear move to R11	N200			X30.22		Z-61.72								
Linear move to R8	N210			X32		Z-64.82								
Rapid back to R9 for the start of the finish cut	N220	G00		X27		Z1								
Linear into work - F1	N230	G01		X25										
Linear move to F2	N240					Z-50								
Circle interpolate CW to F3	N250	G02		X28.35		Z-62.5				25				
Linear move to F4	N260	G01		X50		Z-100								
Rapid back to Safe Position - R0	N270	G00		X60		Z12								
Shutdown spindle	N280													M05
End Program	N290													M30

Figure 4-45 Turning example program sheet

Finally, the program of instructions can be created as discussed in Section 4.5.2. The completed program sheet is shown in Figure 4-45. Note how the program code makes good use of the modal aspect of many of the word addresses. For example, command block N60 issues a G01 code for a linear move. Command blocks N70 and N80 do not list a G-code. Therefore, the G01 code remains in effect. Additionally, command block N60 does not contain a z-coordinate, because there is no change from the previous command block. (Can you identify other command blocks that take advantage of modality?)

4.7 Summary

In the simplest of terms, CNC programming develops a program of instructions to produce a part and converts the instructions into a form the controller can understand. There are various methods available to input the program of instructions into the CNC machine, including manual data input (MDI), conversational programming, and manual part programming. Manual part programming is accomplished using the word address format standard. Word address format programming is typically called G-code programming because of the repeated appearance of the letter G in the programs.

The word address format, as dictated by the ANSI/EIA RS-274-D and ISO 6983 standards, makes use of letter addresses combined with numbers to form word addresses. Word addresses are commands that instruct the CNC controller to perform some action. Word address format programs consist of series of sequential command blocks. The command blocks are made up of ordered word addresses. Word addresses can be classified as either dimensional or non-dimensional. Dimension word addresses provide dimensional information about the tool path. Non-dimensional word addresses include preparatory and miscellaneous codes. Preparatory codes involve actual tool moves. Miscellaneous codes control machine functions not associated with tool moves. Some of the more important preparatory and miscellaneous word addresses are discussed in great detail throughout the chapter. To determine which word addresses are available for a particular machine the machine's format classification sheet should be consulted.

Cutting parameters determine the rate at which a cutting tool removes material from the workpiece during the machining process. Cutting parameters consist of depth of cut, cutting speed, and feed rate. These parameters are dependent on the workpiece material, cutting tool, and other process-dependent considerations. Once determined, these parameters are entered into the G-code program with the S, T, and F letter addresses.

There are ten steps to the programming process. Of them, developing the program tool path, determining program coordinates, and writing the program of instructions are often considered the three most critical. Developing the tool path involves envisioning the path the tool will take to machine the features of the workpiece and label the positions accordingly. From the labeled tool path positions, the program coordinates can be developed. Finally, the program of instructions are written out in English words and then translated into the word address format codes.

4.8 Key Words

ANSI/EIA RS-274-D standard

auxiliary functions

CNC programmer

command blocks

computer-assisted manufacturing (CAM)

conversational programming

cutting cycle

cutting move

cutting parameters

cutting speed

depth of cut

dimension words

feed function

feed rate

finishing cuts

format classification sheet

format classification shorthand

format detail

F word

G-code programming

interpolation parameters

letter addresses

M-code

manual data input (MDI)

manual part programming

material removal

miscellaneous function

modal

modality

non-cutting move

non-dimension words

postprocessor

preparatory functions

program coordinate sheet

program of instructions

program setup

program sheet

roughing cuts

sequence number

spindle speed

spindle speed function

S word

system shutdown

tool function

tool path

T word

word address format

4.9 Review Questions

1. Define a program of instructions as it relates to CNC machining.
2. Discuss the various methods available to format and input the program of instructions into a CNC machine.
3. What is the most common language used in manual part programming?
4. What standard(s) govern(s) the word address format?
5. What are the three sections of a word address format program?
6. Describe the components of a command block in terms of addresses.
7. How is a word address formed?
8. Explain the difference between dimension words and non-dimension words.
9. List the order of words in a block.
10. Describe the difference between a modal word address and a non-modal word address.
11. Describe the function of the G00 code.
12. Describe the function of the G01 code.
13. Write out the G01 codes for the linear cuts shown in Figure 4-46a and Figure 4-46b using absolute coordinates first and incremental coordinates second. Start at position A and assume a constant z depth of -0.25 in.
14. Using all the necessary word addresses, write out the command blocks to move the tool along the tool path shown in Figure 4-47. Note that rapid moves are shown as dashed lines and linear moves are shown as solid lines. Also, the tool path starts and ends at the same point.
15. Describe the function of the G02 and G03 codes.
16. Write out the word addresses to circle interpolate the arc shown in Figure 4-48 in the following directions:
    CCW (use format 1),
    CW (use format 1).

Figure 4-46 Question 13 linear cuts

Figure 4-47 Question 14 tool path

Figure 4-48 Question 16 arc tool path

17. Write out the word addresses to circle interpolate the arc shown in Figure 4-49 in the following directions:

CCW (use format 2),

CW (use format 2).

Figure 4-49 Question 17 arc tool path

18. Explain the difference between the G70 and G71 codes.
19. Explain the difference between the G90 and G91 codes.
20. How do miscellaneous codes differ from preparatory codes?
21. List the function of the M03, M06, and M30 miscellaneous codes.
22. What is the purpose of a format classification sheet?
23. List and describe the three cutting parameters.
24. A 3-flute end mill, made of high-speed steel, is to be used to mill a slot into free machining stainless steel. The end mill is 0.5-in. diameter. Using the feed and speed data listed in Figure 4-30, calculate the spindle speed and feed rate settings for the milling machine.
25. A single point tool, made of high-speed steel, is to be used to turn down a workpiece made of bronze. The workpiece finished diameter is 1.5 in. Using the

feed and speed data listed in Figure 4-30, calculate the spindle speed and feed rate (ipm) settings for the lathe.

26. What G-codes are typically listed in the program setup section of a G-code program? What M-codes are used?

27. What G-codes are typically listed in the material removal section of a G-code program? What M-codes are used?

28. What G-codes are typically listed in the system shutdown section of a G-code program? What M-codes are listed?

29. List and describe the three most critical steps of the programming process.

30. Create a coordinate sheet and fill out a program sheet to mill the two holes and round slot of aluminum workpiece shown in Figure 4-50. The features will be milled with an HSS, 3-flute, end mill, 0.25-in. diameter. Depth of cut is to be 0.25 in.

Figure 4-50 Question 30 drawing

4.10 Bibliography

1. *Machinery's Handbook,* 28th ed. (2008). Industrial Press, Inc., New York, New York.

2. Groover, M.P. (2001). *Automation, Production Systems and Computer-Integrated Manufacturing,* 2nd ed., Prentice Hall, Upper Saddle River, New Jersey.

3. Nanfara, F., Uccello, T., and Murphy, D. (2002) *The CNC Workshop,* Schroff Development Corporation, Mission, Kansas.
4. Gibbs, D., and Crandell, T.M. , 1991 *An Introduction to CNC Machining and Programming,* Industrial Press, Inc., New York, New York.
5. Stenerson, J., and Curran, 2007, K. *Computer Numerical Control*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey.
6. Chang, T.C., Wysk, R.A., and Wang, H.P. 2005. *Computer-Aided Manufacturing*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey.
7. Valentino, J.V. and Goldberg, J. , 2003. *Introduction to Computer Numerical Control (CNC),* 3rd ed., Prentice Hall, Upper Saddle River, New Jersey.

Chapter 5

CNC Simulation Software

Contents

Objective

The objective of this chapter is to introduce a method of simulating or verifying CNC programs. Also covered is a discussion of readily available simulation software package (how to obtain, install, set up, and use it).

5.1 Overview of CNC Simulation Software

As we saw in Chapter 4, developing a G-code program is a complex process. Maintaining accuracy while generating tool path coordinates and writing out program code can be a challenging task. Consequently, the writer of G-code is prone to making code errors. Identifying errors prior to execution of the program in a production environment is of paramount importance. A program released to production without prior verification can lead to expensive and sometimes dangerous conditions. If the program has errors, workpieces might be scrapped, tooling broken, and machinery damaged. Thus, some method of program code verification is always performed.

When CNC technology was in its infancy the only method available for debugging and verifying the program was to run it on the actual machine. This situation is sometimes called *manual prove-out*. The machine the program was written for would be set up for a trial run. The real workpiece material would be substituted with wood, wax, or plastic to minimize finished material waste. The program would then be executed, edited, and rerun until it was accurate and error-free. Then the real workpiece would be machined to optimize speeds and feeds. Finally, the code would be released to production. However, this process had a negative impact on machine productivity, manufacturing lead times, and product costs. Consequently, CNC simulation software, also known as *NC verification software,* was developed to counteract the drawbacks of the traditional manual prove-out process.

CNC simulation software provides a means to test, simulate, or verify a G-code program prior to running it on an actual machine. Simulating program code enables visual confirmation that it is performing as intended. Simulation serves as the primary debugging tool to catch programming, typing, and syntax errors. Additionally, the processing time can be measured and optimized. Thus, when the program is finally sent to the machine to be run, the programmer can have a high degree of confidence in the outcome.

A simulation program reads a G-code program from an electronic file and executes it in the same manner that the CNC controller does on the actual machine. However, instead of driving the axes of the machine, the software simulates the tool motion and material removal in the virtual world of a computer. At a minimum, most packages provide simulation of the workpiece, tooling, and tool path (Figure 5-0). Some higher-end packages enable simulation of the complete machining, tooling, and fixturing (Figure 5-1).

Figure 5-0 Basic simulation example

Figure 5-1 Simulation showing fixturing and machine

For the beginning programmer, CNC simulation software is an invaluable tool. It enables the student to see, firsthand, how the machine interprets each of the word address codes in a program. This leads to one's deeper understanding of both the codes and programming process. Additionally, verifications are performed in the safety of the virtual world. Hence, when program collisions occur, which they inevitably do with beginning programmers, risks to the student, tools, and machine are eliminated.

There are numerous CNC simulation software packages available for purchase. In most situations the level of software sophistication is directly proportional to the required investment. In some cases no investment is required. The focus of this chapter is to introduce the reader to one such simulation program. It is called *CncSimulator*®, published by Bulldog Digital Technologies. CncSimulator® is capable of simulating milling, turning, and gas cutting CNC operations. A screen shot of the software in the default milling configuration is shown in Figure 5-2. It has a continuous free renewable license that the publisher calls "Returnware." The publisher provides a free license for a specific amount of time, after which you "return" to their website to renew or "refuel" the license.

Figure 5-2 CncSimulator screenshot

Section 5.2 discusses how to acquire, install, and set up the software. Section 5.3 covers the *user interface* along with supported G- and M-codes. Simulation examples are presented in Section 5.4.

5.2 Installation and Setup of CncSimulator®

CncSimulator® is available for download from the CncSimulator® website at www.cncsimulator.com. The system recommendations listed on the website include:

- Windows 98® or Windows XP® operating system
- Intel Pentium® or 100% compatible microprocessor
- 32 MB of RAM minimum
- 20 MB minimum hard disk space
- 16 bit high color setting
- 800 x 600 screen resolution

It is important to note that the only operating system the author has used with this software is Windows XP®. Its operability with the Windows Vista® operating system is unknown by the author. The website can provide additional information.

5.2.1 Installation

Locate the CncSetup.exe file. The file is located on the included CD in the Chapter 5 CncSimulator® folder. It is also available for download from the CncSimulator® website (www.cncsimulator.com), download page. Copy the file to a desired folder onto your computer's hard drive.

Execute the CncSetup.exe file by double clicking on it. This will start the CncSimulator® setup process as shown in Figure 5-3. The first screen to appear is a warning page to exit any Windows programs before you continue with the setup process. Press Next when ready to continue.

The next screen to appear is the License Agreement (Figure 5-4). Review the license agreement and select the appropriate radio button. Click Next when ready to continue or cancel to terminate setup.

Figure 5-5 shows the next screen to appear. Specify the desired location to install CncSimulator®. The default location is C:\Program Files\MicroTech\CncSimulator. Click Next when ready to continue or cancel to terminate setup.

The next screen requests a folder name for shortcut icon placement. This is shown in Figure 5-6. Click Next when ready to continue or cancel to terminate setup.

Figure 5-3 CncSimulator® setup screen

Figure 5-4 License Agreement screen

Figure 5-5 Installation folder screen (File: Figure_6-5.eps)

Figure 5-6 Shortcut location screen

The last screen prior to commencement of installation is shown in Figure 5-7. It summarizes all information entered up to this point. Press Back to make any adjustments. When Next is pressed the software is installed. The screen will show installation progress.

The final screen of the setup process in shown in Figure 5-8. It confirms that CncSimulator® has been successfully installed.

Figure 5-7 Setup summary screen

Figure 5-8 Installation confirmation screen

5.2.2 Setup − Petrol File Concept

Because CncSimulator® is a type of Returnware software, it is free as long as the user returns to the website every three months to renew the license. Its publishers view the renewable license as analogous to gassing up one's automobile. A car runs as long as it has fuel. When it runs out, it must be refueled.

When CncSimulator® is first started, two events occur. A pop-up warning message is issued (Figure 5-9), and then the help file opens to the instruction page for downloading a "petrol" file. Refer to these help instructions for assistance on downloading and installing the petrol file. The process is summarized below:

Click the OK button on the pop-up warning message, as is shown in Figure 5-9. This will open the Out of fuel dialog box, shown in Figure 5-10.

Figure 5-9 Petrol file warning dialog box

Figure 5-10 Out of fuel dialog box

The fuel code listed is needed to establish or renew the license. Click the Copy to clipboard button to place this code on the clipboard. Later this code will be pasted into a field on the website.

- Click the www.cncsimulator.com button. This will open the default Internet

    browser and take it to the CncSimulator® web page.

- From the CncSimulator® home page, click the Petrol Station link on the left.
- A new page showing the gas pump image of Figure 5-11 will appear. Paste the fuel code into the field shown. Right click and select the Paste option from the pop-up menu.

Figure 5-11 Gas pump image

- Enter the appropriate email address in the email field. This is where the petrol file will be sent.
- Click the Full tank button to download the petrol file sent to the email address listed.
- At the bottom of this page is a link to download a new driver required for 2008. Click on the ptldriver2008.zip file and download it to the CncSimulator® folder (typically C:\Program Files\MicroTech\CncSimulator). Unzip the file in this folder and replace the old file. Note that subsequent releases of the software may not require this step.
- An email with a link to a site to download the petrol.bin file is sent almost immediately. Retrieve it and click on the link listed. Save the petrol.bin file to the CncSimulator® folder (typically C:\Program Files\MicroTech\CncSimulator).
- Close the Out of Fuel dialog box by clicking the Done button.
- Restart CncSimulator®. It will open to the default milling screen (Figure 5-2).

5.3 User Interface

The default *user interface* is shown in Figure 5-12. A user interface provides a means for the user to interact with the software. Note that this is the default-milling configuration. It will appear different for turning and gas cutting simulations. The interface has a menu and toolbar area across the top. The remainder of the screen is segmented into five sections or *panes*. Three panes are dedicated to showing the simulation from various viewpoints and display configurations. The other two panes of the interface are the *status pane* and the *code editor pane*. A *status bar* stretches across the bottom of the screen.

For milling, the three simulation panes can be altered to show different views and displays of the simulation. For milling and gas cutting there are six different configurations possible for each pane. For turning, only three distinct displays are possible. Figure 5-12 shows the view that is displayed in each *simulation pane*. Methods of how to adjust the display in the pane will be addressed in the next section.

The status pane, located in the lower middle section of the screen, provides information about the status of the machine. Speeds, feeds, tool number in use, tool position, tool depth, spindle rotation direction, coolant status, and machining time are all displayed. Additionally, simulation control is provided in this frame. The simulate button shown starts the simulation. This is discussed in more detail in a later section.

The G-code program is displayed and edited in the code editor pane. Note that the program can be loaded from a text file with an ".nc" file extension or typed directly into the editor. The code editor pane has all the standard Windows editing capabilities along with some additional simulation controls. These are accessed by right clicking in the pane and/or from the pull down menus.

Figure 5-12 CncSimulator® user interface

5.3.1 Screen Customization

The user can customize the interface in a number of ways. Toolbars can be moved; panes can be resized—even into full screen size if desired, and simulation views changed. Additionally, the status bar and standard toolbar can be removed to maximize the simulation, status, and code editor panes.

The toolbars shown in Figure 5-12 are in the "docked" position. The user can move a toolbar any place on the screen by right clicking on its border and dragging it to a new position, holding the right mouse button (or touch pad button) down while dragging. Releasing the button in the desired location drops the toolbar there. Figure 5-13 shows each toolbar in its "floating" state.

The panes can be resized by dragging the borders vertically or horizontally. Figure 5-14 identifies the borders, and Figure 5-15 shows the interface after the size of the panes has been changed. Whenever CncSimulator® is restarted the panes will return to their original size.

Figure 5-13 Floating toolbars

Figure 5-14 Pane borders

Figure 5-15 Resized panes

For milling and gas cutting simulations, six different views can be displayed in the simulation panes. There are three views possible for turning simulations. The views available for each type of simulation, along with their corresponding icons, are listed in Figure 5-16.

For milling, the three-dimensional wireframe view has some interesting capabilities. Most notably, the view is adjustable. There are scrollbars located at the right and bottom of the pane. By adjusting the scrollbars, the workpiece can be rotated about a horizontal axis, a vertical axis, or about the center of pane. The axes pass through the center of the workpieces. A fourth scrollbar adjusts the zoom or distance from the model. Note that the rotation about the center of the screen is about an imaginary axis normal to the screen. (Figure 5-17). The three-dimensional wireframe view also displays the tool and tool path. Rapid moves are shown with dotted lines. Interpolation moves are shown with solid lines. The solid lines are colored, corresponding with the depth of cut gradient shown in the status pane. See Figure 5-18.

Adjusting the view of a simulation pane is a very simple process. Left click anywhere in the simulation pane and select the icon of the desired view from the view toolbar. Note that the view name in the upper left corner of the simulation pane will highlight red when the pane has been properly selected. Figure 5-19 demonstrates the process of changing the view of the simulation pane, which is located on the top right.

Icon	Description	Milling/Gas Cutting	Turning
XY	Displays a plan view of the work piece, looking down the +Z axis onto the X-Y plane. Work piece is displayed as a solid model.	O	Ø
YZ	Right side view. Views Y-Z plane, looking down the +X axis. Work piece is displayed as a solid model.	O	Ø
XZ	Front view for milling simulations. Views the X-Z plane looking down the -Y axis. For turning operations, this view is also of the X-Z plane, but looking down the + Y axis from above. Work piece is displayed as a solid model.	O	O
	Plot View. Displays the tool path plotted out on a X-Y graph for milling and gas cutting. For turning, the tool tip location is plotted on an X-Z graph. Interpolation moves are shown as solid lines, rapid moves as dotted lines.	O	O
	3D Wireframe View. Displays an adjustable 3D wireframe view of the model. The view can be adjusted by scroll bars located on the bottom and right side of the screen.	O	Ø
	Block View, a 3D Solid View. Displays a 3D oblique view of the model.	O	O
Key	O - Available Ø - Not Available		

Figure 5-16 Simulation views

Figure 5-17 Three-dimensional view

Figure 5-18 Display of rapid and interpolation moves in three-dimensional view

Figure 5-19 Changing view in pane

Another nice feature is that any simulation pane can be expanded to fit the screen. Again, the user simply makes the pane active as described above and selects the Full Screen icon on the view toolbar (labeled in Figure 5-19). To return to the multipane default screen, the esc key is pressed. Note that when the Full Screen mode is in use, a floating status pane is also displayed. Figure 5-20 shows full screen mode, which is slightly different from the default view.

Figure 5-20 Full screen mode

5.3.2 Menus, Toolbars, and Dialog Boxes

The CncSimulator® is controlled through the following three interfaces:
- pull down menus
- toolbars
- simulation dialog box.

The pull down menus provide a means to open, close, and edit G-code programs, enter workpiece sizes, tooling information, enter type of simulation (milling, turning, etc.), control program settings, and access help. Additionally, CncSimulator® is an open source type of program. Thus, plug-ins can be created by the user and accessed from the pull down menu.

Figure 5-21 Help dialog box

The standard toolbar and the code editor toolbar duplicate some of the functions provided through the pull down menus. However, the view toolbar is the only interface available for control of the simulation pane views.

The menus and toolbars are well-documented in the Help dialog box, which the user accesses by selecting the Help pull down menu and clicking on Help. Figure 5-21 shows the Help dialog box.

The simulation dialog box provides exclusive control over the actual simulation. To activate the dialog, click on the Simulate button located in the status pane. Figure 5-22 displays how to open the simulation dialog box, which contains three buttons and a slider control as shown in Figure 5-23. The Automatic Simulation button simulates the G-code program from beginning to end at the speed set by the slider control. The Single Line Simulation button simulates one line of code at a time, effectively allowing the user to step through the simulation line by line. The Stop Simulation button cancels automatic simulation and closes the simulation dialog box.

Figure 5-22 Opening the simulation dialog box

Figure 5-23 Simulation dialog box

5.3.3 G- and M-Codes Supported by CncSimulator®

CncSimulator® utilizes an *interpreter* to convert the CNC program code into movement data for display on the screen. The default interpreter recognizes only standard ISO codes. The user can program customized interpreters for specific machines. The website's developer's page gives detailed information on this subject. For the purposes of this text the default interpreter suffices. The G-codes that the interpreter supports depend on the particular machining operation—codes for milling differ from those for turning operations. However, the M-codes are the same for both milling and turning; M-codes supported are shown in Figure 5-24.

The G-codes shown in Figure 5-25 are supported by the default interpreter for milling. Most of the common preparatory functions discussed in Chapter 4 are supported, with the exception of G04, G70, and G71. G04, for initiating dwells, is of little consequence in program simulation. However, G70 and G71 are of vital importance because these codes provide the programmer with control over units (inch or millimeter). Since CncSimulator® was designed specifically for metric applications, the program developers may have been deemed these codes unnecessary for inclusion. Yet, CncSimulator® can accommodate inch units, doing so through the pull down menus of the user interface. Simulation examples in Section 5.4.2 explain this.

M Code	Description
M00	Stops the program and correspondingly the machine
M01	Optional method to stop the program and machine
M02	Indicates end of program
M03	Turn on spindle in the clockwise (CW) direction
M04	Turn on spindle in the counter clockwise (CCW) direction
M05	Turn off or stop the spindle
M06	Initiates a tool change
M08	Turns on coolant
M09	Turns coolant off
M17	Return from subprogram
M30	Indicates end of program and resets the machine

Figure 5-24 CncSimulator®-supported M-codes

G Code	Description
G00	Rapid or point-to-point move.
G01	Linear interpolation
G02	Circle interpolation in the clockwise direction
G03	Circle interpolation in the counter-clockwise direction
G25	Subprogram or subroutine call
G26	Separate subprogram or subroutine
G40	Cancel cutter compensation/offset
G41	Cutter compensation left
G42	Cutter compensation right
G53	Datum shift cancel
G54	Datum Shift 1 - Moves zero point to new position 1
G55	Datum Shift 2 - Moves zero point to new position 2
G56	Datum Shift 3 - Moves zero point to new position 3
G57	Datum Shift 4 - Moves zero point to new position 4
G58	Datum Shift 5 - Moves zero point to new position 5
G59	Datum Shift 6 - Moves zero point to new position 6
G81	Drilling Cycle
G87	Rectangular Pocket Milling Cycle
G90	Absolute dimensions
G91	Incremental dimensions
G92	Preload registers to shift coordinate axes relative to the current tool position
G94	Feed rate in inches or millimeters per minute
G95	Feed rate in inches or millimeters per revolution

Figure 5-25 CncSimulator®-supported G-codes for milling

An important note: If a G-code *not* listed in Figure 5-25 *is* encountered in the simulation of a program, the interpreter will *ignore* it. Ignoring it will not result in errors or a crash in the program. Thus, if a program of instructions contains a G04 or G70, for example, the interpreter takes no action. G70 or G71 to signify inch or millimeter units, respectively, may still be used in programs simulated with CncSimulator®, but the interpreter will ignore the code.

The other unfamiliar G-codes listed in Figure 5-25 are advanced codes. For the most part advanced codes are shortcuts. They are designed to simplify the programming process or provide additional control. They are typically process-specific, developed specifically for milling or turning applications. Thus, the advanced G-codes supported by the interpreter for turning operations are different than the milling G-codes.

In addition to knowing the G- and M-codes supported by the interpreter, it is important that the user know the letter address formats supported as well. Figure 5-26 displays the letter addresses supported for milling operations.

Note that the standard interpreter can use an alternate format for arcs for G02 and G03 instead of the formats defined in Chapter 4, and these formats will still function as intended. Here the convention for circle interpolation defined in Chapter 4 is followed. The alternate format is not listed in Figure 5-26. CncSimulator®'s Help dialog provides additional information.

The interpreter-supported G-codes for turning are shown in Figure 5-27, and the supported letter addresses are listed in Figure 5-28. Again, note the use of advanced codes for threading and roughing.

5.3.4 Machine Code versus Simulator Code

CNC programs are written for production carried out on a specific machine, but not for the simulator, which is just a tool to verify or prove out the program. Some simulators come with interpreters that match the controller used on the actual machine. For other simulators, the CncSimulator® being one, the user can develop or purchase an interpreter to mimic the machine control. However, if the simulator does not fully support the exact same codes used by the actual machine controller, the simulation may not be valid, or it may provide misleading information. Therefore, it is important the user determine if the simulator fully supports the machine code prior to simulating the program.

If the simulator does not fully support the machine, but simulation is of vital importance, the simulation program can be tailored to function fully on both simulator and actual machine. How? The programmer might write the program using mostly *common* preparatory codes—those which the simulator *can* support—and a few *advanced* codes, those that the simulator also supports. Tailoring such a program (one that runs on simulator and actual machine) requires that differences between the codes be identified and documented.

Letter Address	Description
F	Primary or first axis feed function used to specify the feed rate for preparatory interpolation codes.
I	Used with circle interpolation to specify incremental location of arc center from start point in X direction
J	Used with circle interpolation to specify incremental location of arc center from start point in Y direction
K	Used with circle interpolation to specify incremental location of arc center from start point in Z direction
L	Subprogram block number used with subroutines
N	Identifies the command block and specifies the order in which command block is executed within the program
P	Special parameter that defines overlap in rectangular pocket milling used with G87 pocket milling code
R	Specifies radius of arc used in circle interpolation
S	Specifies the speed at which the spindle is to rotate
T	Specifies tool number to be selected on machines that have an automatic tool changer
U	Special parameter that specifies number of times subroutine is repeated
X	Specifies primary motion dimension in X direction
Y	Specifies primary motion dimension in Y direction
Z	Specifies primary motion dimension in Z direction

Figure 5-26 CncSimulator®-supported letter addresses for milling

Figure 5-29 and Figure 5-30 show such documentation, between supported codes of CncSimulator® and those of two mills located in the author's laboratory. Figure 5-29 has a listing and description of all M-codes available on CncSimulator®, Mill #1, and Mill #2. An X in a cell of the table indicates that the code to the left is supported by the device named at the top of the column. A blank cell means the particular device does not support the code. Figure 5-30 is the listing for G-codes. (A similar table for all the letter addresses should be developed as well.)

G Code	Description
G00	Rapid or point-to-point move.
G01	Linear interpolation
G02	Circle interpolation in the clockwise direction
G03	Circle interpolation in the counter-clockwise direction
G25	Subprogram or subroutine call
G26	Separate subprogram or subroutine
G53	Datum shift cancel
G54	Datum Shift 1 - Moves zero point to new position 1
G55	Datum Shift 2 - Moves zero point to new position 2
G56	Datum Shift 3 - Moves zero point to new position 3
G57	Datum Shift 4 - Moves zero point to new position 4
G58	Datum Shift 5 - Moves zero point to new position 5
G59	Datum Shift 6 - Moves zero point to new position 6
G83	Threading cycle
G84	Rough machining cycle
G90	Absolute dimensions
G91	Incremental dimensions
G92	Preload registers to shift coordinate axes relative to the current tool position
G94	Feed rate in inches or millimeters per minute
G95	Feed rate in inches or millimeters per revolution

Figure 5-27 CncSimulator®-supported G-codes for turning

Letter Address	Description
D	Special parameter for threading cycle, used to define the depth in mm/cut
F	Primary or first axis feed function used to specify the feed rate for preparatory interpolation codes.
H	Special parameter for threading cycle, used to define the total thread depth
I	Used with circle interpolation to specify incremental location of arc center from start point in X direction
K	Used with circle interpolation to specify incremental location of arc center from start point in Z direction
L	Special parameter for threading cycle, used to define the dia. or radius for point A
N	Identifies the command block and specifies the order in which command block is executed within the program
P	Special variable that defines overlap in rectangular pocket milling used with G87 pocket milling code
R	Specifies radius of arc used in circle interpolation
S	Specifies the speed at which the spindle is to rotate
T	Specifies tool number to be selected on machines that have an automatic tool changer
U	Special variable that specifies number of times subroutine is repeated
X	Specifies primary motion dimension in X direction
Z	Specifies primary motion dimension in Z direction

Figure 5-28 CncSimulator®-supported letter addresses for turning

If a program is written for Mill #1 and is to be verified on CncSimulator®, only codes that have an X in the CncSimulator® column and the Mill #1 column can be used. Thus, the applicable M-codes are M02, M03, and M05. The usable G-codes are G00, G01, G02, G03, G90, and G91. This is not an extensive set of codes, but is more than adequate for machining parts of relatively high complexity. Additionally, if other codes are needed for the program to function properly, but are not supported by both devices, they may still be used. Most simulators and controllers ignore unsupported codes.

M-word	Description	CncSimulator	Mill #1	Mill #2
M00	Program stop; used in middle of program. Operator must restart machine.	X		X
M01	Optional program stop; active only when optional stop button on control panel has been depressed.			X
M02	End of program. Machine stop.	X	X	X
M03	Start spindle in clockwise direction for milling machine (forward for turning machine).	X	X	X
M04	Start spindle in counterclockwise direction for milling machine.	X		X
M05	Spindle stop.	X	X	X
M06	Execute tool change, either manually or automatically. If manually, operator must restart machine.	X		X
M08	Turn cutting fluid on mist.	X		X
M09	Turn cutting fluid off.	X		X
M10	Automatic clamping of fixture, machine slides, etc.		X	
M11	Automatic unclamping.		X	
M17	Return from subroutine command	X		
M20	Chain. The program you are chaining to must be in the same directory as the program you are chaining from.		X	
M25	Set robot output #1 off Used for robot synchronization.		X	
M26	Set robot output #1 on. Used for robot synchronization.		X	
M30	End of program. Machine stop. Rewind tape (on tape-controlled machines).	X		X
M35	Set robot output #2 off Used for robot synchronization (see Appendix C).		X	
M36	Set robot output #2 on: Used for robot synchronization (see Appendix C).		X	
M38	Stepper Motors OFF		X	
M47	Rewind Restarts the currently running program; takes effect after all motion comes to a stop.		X	
M71	Puff Blowing ON			X
M72	Puff Blowing OFF			X
M98	Call subroutine command			X
M99	Return from subroutine command			X

Figure 5-29 M-code comparison chart example

218   Chapter 5

G-word	Description	CncSimulator	Mill #1	Mill #2
G00	Point-to-point movement (rapid traverse) between previous point and endpoint defined incurrent block.	X	X	X
G01	Linear interpolation movement.	X	X	X
G02	Circular interpolation, clockwise.	X	X	X
G03	Circular interpolation, counterclockwise.	X	X	X
G04	Dwell for a specified time.		X	X
G05	Pause for Operator Intervention - Press Return to resume machining		X	
G09	Exact Stop			X
G15	End Polar Coordinate Interpolation			X
G16	Begin Polar Coordinate Interpolation			X
G17	Selection of x-y plane in milling.	X		X
G18	Selection of x-z plane in milling.	X		X
G19	Selection of y-z plane in milling.	X		X
G20/G70	Input values specified in inches.	X	X	X
G21/G71	Input values specified in millimeters.	X	X	X
G25	Wait for robot input #1 to be off		X	
G26	Wait for robot input #1 to be on		X	
G28	Return to reference point.			X
G29	Return from reference point.			
G33	Thread Cutting			X
G35	Wait for robot input #2 to be off		X	
G36	Wait for robot input #2 to be on		X	
G40	Cancel offset compensation for cutter radius (nose radius in turning).	X		X
G41	Cutter offset compensation, left of part surface.	X		X
G42	Cutter offset compensation, right of part surface.	X		X
G43	Tool Length Compensation Plus			X
G44	Tool Length Compensation Minus			X
G49	Tool Length Compensation Cancel			X
G50	Specify location of coordinate axis system origin relative to starting location of cutting tool.			X
G51	Scale Factor			X
G52	Local Coordinate System			X
G53	Machine Coordinate System	X		X
G54	Zero Offset 1/Workpiece Coordinate Settings	X		X
G55	Zero Offset 2	X		X
G56	Zero Offset 3	X		X
G57	Zero Offset 4	X		X
G58	Zero Offset 5	X		X
G59	Zero Offset 6	X		X
G61	Exact Stop Mode			X
G62	Automatic Corner Override			X
G64	Cutting mode			X
G73	High Speed Peck Drilling OR Chip Break Drilling Cycle (EMCO)			X
G74	Left Tapping Cycle			X
G76	Fine Drilling Cycle			X
G80	Cancel Canned Cycles			X
G81	Drilling Cycle Drilling	X		X
G82	Counter Boring Cycle OR Cycle with Dwell (EMCO)			X
G83	Deep Hole drilling cycle OR Withdrawal Drilling Cycle (EMCO)			X
G84	Tapping Cycle			X
G85	Reaming Cycle			X
G86	Drilling Cycle with Spindle Stop			X
G87	Back Pocket Drilling Cycle			X
G88	Drilling Cycle with Program Stop			X
G89	Reaming Cycle with Dwell			X
G90	Programming in absolute coordinates.	X	X	X
G91	Programming in incremental coordinates.	X	X	X
G92	Specify location of coordinate axis system origin relative to starting location of cutting tool.	X		X
G94	Specify feed per minute in milling and drilling.	X		X
G95	Specify feed per revolution in milling and drilling.	X		X
G97	Revolutions per Minute			X
G98	Specify feed per minute in turning OR Set initial plane default			X
G99	Specify feed per revolution in turning OR Return to retract (rapid) plane			X

Figure 5-30 G-code comparison chart example

Figure 5-32 Appearance at startup

From the Files pull down menu, select Open or select the Open icon from the standard toolbar (see Figure 5-32). The Open dialog box will appear as shown in Figure 5-33. Double click on the Mill folder, followed by the MM folder and finally the ISO folder.

Select the file _Sample2.nc and press the Open button as shown in Figure 5-34. The file will open in CncSimulator®. The interface will now appear as shown in Figure 5-35.

Figure 5-33 File open dialog box

Figure 5-34 Sample file selection

Figure 5-35 CncSimulator® interface after opening sample file

Note the size of the workpiece changed since the _Sample2.nc file was opened. The programmer specifies the workpiece size settings when the code is written or loaded for the first time (if written in a text editor). When the code is saved the workpiece size gets saved along with the program code. Thus, when the _Sample2.nc file was opened, so was the workpiece.

To determine the size of the workpiece, select Simulate from the pull down menu and click on the Detail Settings option, as is shown in Figure 5-36. This will open the milling options dialog box shown in Figure 5-37. Observe that the workpiece is 200 mm long by 150 mm wide by 20 mm thick. The workpiece will always be a rectangular prism (box), unless the Choose drawing option is used. This will open a Windows® standard file Open dialog box, enabling the user to select a .rit type CAD file. Since most CAD systems do not use this file extension, there is a free converter program available for download from the CncSimulator® website. The converter, called dxf2rit.exe, will

convert DXF R13 or R14 files into the .rit file extension. Complete instructions are included with the download.

Figure 5-36 Opening milling options dialog box

Referring to the graphics window of the dialog box, observe the location of the PRZ. Its location is set by the values specified in the NullpointX and NullpointY fields. The Scale factor field scales the tool motion relative to the workpiece. Setting a scale less than 1 will cause tool motion to decrease proportional to the scale factor. If the scale is set to a value greater than 1, tool motion increases proportional to the scale factor.

At the very bottom of the Milling options dialog box is the Milling depth button. This enables the user to set different colors to correspond to depth of cut. These colors will be displayed on the workpiece as material is removed during the simulation.

Note that several options are grayed out in the dialog box. This indicates that the options are not available with milling. These particular options are used for turning operations. To close the dialog box, click the OK button.

Refer to Figure 5-38, which highlights the code editor pane. Notice that the code consists of only rapid and linear interpolation moves. Recall that these codes are typically found in the material removal section. Additionally, there are no G90, M03, or M06 codes. Hence, it can be concluded that this particular program does not need a program setup section. This is sometimes the case when the machine for which the program is written is to be set up manually. Thus, the operator would load the correct tool, turn the spindle on, and set the speed manually.

Figure 5-37 Milling options dialog box

Scroll down to the bottom of the program by using the scroll bar on the right of the pane. Observe that the System Shutdown section, not shown in Figure 5-38, has only three lines: two rapid moves to reposition the tool and an M30 to end the program.

The sequence numbers (N10, N20, …) for the command blocks are also noticeably absent. Again, this is because the machine for which the code was written does not require sequence numbers. Note, however, the convention of the present text is to include sequence numbers in examples.

Figure 5-38 Code editor pane

To simulate the program:

Open the simulation dialog box by pressing the Simulate button located in the status pane. Refer to Figure 5-22.

Start the simulation by pressing the Automatic Simulation button. Control the speed of the simulation by adjusting the slider control. Refer to Figure 5-23.

Figure 5-39 is a snapshot of the user interface during program simulation. Observe that each command block is highlighted as it is being simulated. In the three-dimensional simulation pane, note how rapid moves are distinguished from interpolation moves. Notice in the status pane that the tool depth is identified by a variable length vertical bar. The position of this bar relative to the milling depth color code sets the color displayed in the workpiece during simulation. Recall that this color code can be set from the milling options dialog box.

Figure 5-39 Program simulation snapshot

Experiment with the simulation dialog box controls by stopping, restarting, and slowing down the simulation. Also, try stepping through a few lines of code using the single line simulation button. Do not exit CncSimulator®. Continue on to the next example.

Simulation Sample File 2

In this example a much more complicated program is simulated to demonstrate additional features of CncSimulator®. Using the procedure described in the last example, open the file _Sample6.nc from the ISO folder. Note how the program text in the code editor pane and the workpiece size change after the program is loaded.

Whenever a file is loaded it is stored in CncSimulator®'s *text buffer*. The text buffer is capable of holding several CNC programs simultaneously. Controls for the text buffer are located to the right of the code editor pane (Figure 5-40).

Text Buffer Controls

Text buffer list

Previous text buffer

Next text buffer

Close text buffer

Figure 5-40 Code editor pane text buffer controls

Click on the Text buffer list button. This opens the Buffer list dialog box. Assuming that this example is performed immediately following Simulation Sample File 1 the buffer list will appear as shown in Figure 5-41. Note how the file used in Simulation Sample 1 (_Sample2.nc) and the _Sample6.nc file both appear in the list. The Nameless1 buffer is an empty buffer available for entering a new program. It is possible to switch between the buffers by simply clicking on the file name in the Buffer list dialog box and clicking on the OK button.

A buffer can also be deleted (similar to closing a file in other programs) from the buffer list by selecting it and then clicking on the Delete button. If the program in the

buffer was not previously saved, the user is prompted to save the program before it is deleted.

Figure 5-41 Buffer list dialog box

The other buffer control buttons, shown in Figure 5-40, provide additional means of switching between and closing buffers. Experiment with these controls before continuing. Be sure to make the _Sample6.nc buffer the current buffer before proceeding.

Verify that the size of the workpiece is 150 mm long by 150 mm wide and 50 mm thick. Recall that this verification is accomplished by opening the milling options dialog box. The dialog box is opened by selecting Detail settings from the Simulate pull down menu.

Scroll through the program and note that this example program file uses five different tools: T1, T2, T4, T5, and T8. For milling, tools are contained in a text file with a .tol file extension. The user can alter this file and create additional separate tool files for each milling machine available. When CncSimulator® is opened a default tooling file is loaded. To see the content of the tooling file, select Edit tools from the Simulate pull down menu. This will open the milling tool dialog box as shown in Figure 5-42. Observe the sizes of the tools used in this program.

Figure 5-42 Milling tool dialog box

Figure 5-43 Completed simulation

Figure 5-44 Selecting turning screen

The tools can be edited directly from this dialog box and then saved in the same or a different file. Click OK to close the dialog box. To save changes to the tool file, select Save tools from the Simulate pull down menu. To open another tooling file, select Open tools from the Simulate pull down menu.

To simulate the program, select the Simulate button in the status pane followed by the Automatic simulation button in the simulation dialog box. Adjust the simulation speed as needed. Observe how the program makes good use of roughing passes followed by finishing passes. Also, note how the coolant is turned on and off during the simulation. The completed simulation is shown in Figure 5-43.

Simulation Sample File 3

In this example a turning program is simulated. Switch to the turning simulator by selecting Turning from the Screen pull down menu as is shown in Figure 5-44. Open the file _Sample4.nc from the Turn/MM folder. The screen should appear as indicated in Figure 5-45.

Observe the use of the PLOT view in the lower left hand pane. Since turning tools move in the two dimensional (x,y)-plane, the PLOT view provides essentially the same information as the three-dimensional view. Hence, the three-dimensional view is not available. Two other differences between a turning and milling simulation are seen in the tool and detail setting dialogs.

Figure 5-45 Turning screen

Lathe or turning tools, aside from drills, are very different from milling tools. Turning tools have a distinct shape corresponding to the operation to be performed. Therefore the tooling dialog for turning simulations was adjusted accordingly. Instead of one list of tools, as with milling, each tool is stored in an individual text file and displayed separately in a dialog box. Open this dialog box by selecting Edit tools from the Simulate pull down menu. The turning tool dialog box is shown in Figure 5-46.

The buttons along the bottom of the tool dialog box enable the user to:
- Navigate through the tool files contained in the t_tools folder
- Change the visible or current tool's orientation (MirrorX and MirrorY buttons)
- Delete the current tool
- Copy and paste the current tool to another file.

Scroll through the tools listed to become familiar with all the tools available in the "t_tools" folder.

Figure 5-46 Turning tool dialog box

The detail settings dialog for the turning interface is only slightly different than the milling dialog. Open this dialog box by selecting Detail settings from the Simulate pull down menu. Note that the fields for inputting the workpiece size are relabeled to accommodate round lathe workpieces. Additionally the NullpointX and NullpointY fields are grayed out. Conversely, the options that were grayed out in the milling options dialog box are now available in the lathe options dialog box. See Figure 5-47. The Origin at end face check box enables the user to move the origin to the right face of the part. The radius coordinates and diameter coordinates check boxes allow the user to indicate whether moves in the *x* direction are radius or diameter moves. Make no changes to the default settings and close the lathe options dialog box by clicking on the Cancel button.

Simulate the program by clicking on the Simulate button on the status pane followed by the Automatic Simulation button on the simulation dialog box. Figure 5-48 is a snapshot of the simulation in progress. Observe the appearance of the tool in the XZ view simulation pane. Note how rapid moves are distinguished from interpolation moves in the PLOT view simulation pane. Also, notice that the BLOCK view shows a three-dimensional view with a segment cut away. This is intended to provide better visualization of the finished workpiece.

Figure 5-47 Lathe option dialog box

5.4.2 CncSimulator® and Inch Units

As explained, CncSimulator® was developed specifically for millimeter units. However, programs written in inches can still be simulated. Simulation of inch milling programs requires only minor adjustments. Turning simulation programs require slightly more effort, including downloading an inch tool file from the CncSimulator® website.

To perform simulations of milling programs written in inch units, perform the following two tasks:

Figure 5-48 Turning simulation snapshot

Create appropriate inch-sized tools.

Open the Milling Tool dialog box by selecting Edit tools from the Simulate pull down menu. Scroll down to find unused fields. Enter a new tool number followed by the appropriate tool size information. Refer to Figure 5-49. Recall that this tool list can be saved by selecting Save tools from the Simulate pull down menu.

Specify the workpiece size in inch units.

- Select Detail settings from the Simulate pull down menu to open the Milling options dialog box. Enter the size of the workpiece in inch units.
- Prior to simulating the program, be sure the program calls the correct tool from the milling tool dialog box.
- Simulation of milling programs that are written in inch units is demonstrated in the Section 5.4.3. Simulating turning programs is somewhat more involved. To simulate turning programs written in inch units, complete the following steps:

Click "OK" when complete

Scroll down

Enter tool number

Enter tool size and length

Provide a tool description

Figure 5-49 Milling tool dialog box—creating an inch tool

- Download the Turn tool directory file from the CncSimulator® website.
- Go to the CncSimulator website by navigating to www.cncsimulator.com. At the bottom of the home page, select the Using inch link. Click on the Turn tool directory link to download the file t_tools_inch.zip. Download the file in a new folder in the C:\Program Files\MicroTech\CncSimulator directory titled t_tools_inch.
- Open the zipped file and extract the tool files to the new directory.
- Set the correct path to the inch lathe tools.
- Select Lathe tool path from the Simulate pull down menu as shown in Figure 5-50.

To verify that the inch tools are working correctly, simulate the inch.nc file located in the t_tools_inch folder. Simulating turning programs that are written in inch units is demonstrated in Section 5.4.4.

Figure 5-50 Selecting the inch lathe tool path

5.4.3 Milling Simulation Example

In this section the process of simulating a previously written milling program in inch units will be demonstrated. This process of simulating a milling program in CncSimulator® can be broken down into the following steps:

- Set the workpiece size and PRZ location in the milling options dialog box.
- Select Detail settings from the Simulate pull down menu.
- Enter the size of the workpiece in the appropriate data fields.
- Relocate the PRZ, if necessary, by entering new values in the Nullpoint X and Nullpoint Y fields.
- Click OK when complete.

- Locate or enter the appropriate tooling in the milling tool dialog box.
- Select Edit tools from the Simulate pull down menu.
- Edit existing tools or enter new tools as required.
- Click OK when complete.
- Type the program in the Code editor pane.

From the Edit pull down menu select Automatic line number. Enter the desired start number and increase value in the Automatic line numbers dialog box. Refer to Figure 5-51.

Type the program into the code editor.

- Check the program code for syntax errors.
- Select Check code from the Edit pull down menu.
- Fix any code syntax errors accordingly.
- Simulate the program.
- Open the Simulation dialog box by pressing the Simulate button located in the status pane. Refer to Figure 5-22.
- Start the simulation by pressing the Automatic Simulation button. Control the speed of the simulation by adjusting the slider control. Refer to Figure 5-23.
- Note it is also possible to step through the program, command block-by-command block, by selecting the Single Line Simulation button.
- Additional simulation options are available by right clicking in the Code editor pane next to any line of code.
- Save the program code and workpiece options.
- Select Save from the Files pull down menu.
- When prompted, select Yes to save the workpiece options.

Figure 5-51 Setting automatic line numbering

In Section 4.5 of the previous chapter the CNC programming process was discussed in detail and a milling program was developed. The product drawing is shown in Figure 5-52 and the completed program sheet is shown in Figure 5-53. Figure 5-52 shows a product with two holes and slot that will be produced on a milling machine utilizing a 4-flute, 0.25-inch diameter end mill cutter. Figure 5-54 shows how the part will be located in the machine and the layout of the workpiece coordinate system. Using the simulation process steps listed above, simulate this program.

Figure 5-52 Example product drawing

Program Sheet

Program Number: 1001											Sheet No.: 1	
Programmer:			Part Name:			Block					Date: -	

Process Flow/Tool Path Description	N	G - Code		Axis Coordinate			Center of Arc			Radius of Arc R	F	S	T	M-Code
				X	Y	Z	I	J	K					
Set absolute coordinates and inch units	N10	G90	G70											
Change the tool	N20												T17	M06
Move to a safe position, 1 inch above work piece	N30		G00	X0	Y0	Z1								
Turn on the spindle CCW and set the RPM's and feed rate	N40										F76	S3800		M03
Move to a position directly above the work piece	N50		G00			Z0.1								
Move to position 1, just above work piece	N60		G00	X.5	Y.5									
Drill 1st hole or linear move into work piece	N70		G01			Z-.25								
Rapid move out of work piece	N80		G00			Z0.1								
Move to Position 2, just above work piece	N90		G00		Y1.5									
Drill 2nd hole or linear move into work piece	N100		G01			Z-.25								
Rapid move out of work piece	N110		G00			Z0.1								
Move to position 3, just above work piece	N120		G00	X1.5	Y1.0									
Move into workpiece depth of slot	N130		G01			Z-.25								
Linear move to position 4	N140		G01	X2.03	Y1.53									
Circle interpolate arc in CW direction	N150		G02	X2.03	Y.47					R.75				
Rapid out of work	N160		G00			Z0.1								
Move to safe position	N170		G00	X0	Y0	Z1								
Shutdown spindle	N180													M05
End Program	N190													M30

Figure 5-53 Example completed program sheet

Figure 5-54 Example workpiece coordinate system

Note the following:

Figure 5-55 shows the settings for the milling options dialog box, which establishes the size and PRZ of the workpiece. As the workpiece size is changed, the part shown initially in the dialog box graphics area (the black part of the dialog box) disappears. This is caused by the lack of automatic scaling in this graphics area. Because CncSimulator® was developed for millimeter units, a workpiece with inch units is very small relative to the default workpiece, and thus disappears from view. This only occurs in this dialog box when the workpiece is small. Also, the PRZ does not change; therefore Nullpoint X and Nullpoint Y remain zero. Click OK to close the dialog box. The workpiece shown in the three simulation panes adjusts in size accordingly.

The default Milling tool dialog box does not contain a 0.25-inch diameter cutting tool. Therefore, one will be created. Figure 5-56 displays the settings in the Mill tool dialog box for the new tool to be created for this simulation. After entering the new tool information and clicking the OK button, the user is prompted to save the revised tool file. Click OK again to save it to the existing file or select the Other name button to save it to a different name. For this example, save it to the existing file.

Figure 5-55 Milling options dialog box settings

NR	Toolnr	Dia.	Length	Feed	Spindle	Name
10	10	1.000	30.000	0.000	0.000	Edgemill d. 1
11	11	10....	30.000	0.000	0.000	Drill diam. 10
12	12	12....	30.000	0.000	0.000	Drill diam. 12
13	13	14....	30.000	0.000	0.000	Drill diam. 14
14	14	16....	30.000	0.000	0.000	Drill diam. 16
15	15	18....	30.000	0.000	0.000	Drill diam. 18
16	16	20....	30.000	0.000	0.000	Drill diam. 20
17	17	0.25	1.0	0.000	0.000	1/4" end mill
18	0					
19	0					
20	0					

OK · Cancel · Prompt to save modified tool file

New Tool Number · New Tool Diameter · New Tool Length · New Tool Description

Save as FTOOLS.TXT? Yes No Other name

Figure 5-56 Milling tool dialog box settings

When entering program code in the **Code editor** pane, hit Enter at the end of each command block to move to the next line. Be sure to enter the correct tool number in the program code. If a different tool number (other than 17) identifies the ¼-inch endmill in the user's tool file, use the number so indicated.

If a syntax error is found during the code check, a warning dialog box will appear. After clicking **OK** in the warning box, the error will be highlighted in the **Code editor** pane. Figure 5-57 displays an example of a syntax error in which the letter **O** was inadvertently used instead of the number zero. This is a fairly common occurrence among students due to the close proximity of zero and letter **O** on the computer keyboard, or because they may think the characters zero and uppercase letter O are interchangeable. Fix any syntax errors and repeat the check code procedure until the warning dialog box no longer appears.

When the simulation is complete the screen should look similar to Figure 5-58. Be sure to save the program and workpiece options.

Figure 5-57 Syntax error demonstration

Figure 5-58 Completed simulation

5.4.4 Turning Simulation Example

In this section, the process of simulating a previously written turning program will be demonstrated. This process is very similar to simulation of a milling program, with a few minor exceptions:

Start CncSimulator.

- Start CncSimulator® by clicking on the Start button and selecting Programs>MicroTech CncSimulator>CncSimulator.
- If necessary, set the screen to Turning.
- Select Turning from the Screen pull down menu.
- Set the workpiece size, move the PRZ (origin) to the end of the workpiece face and specify Radius or Diameter coordinates in the lathe options dialog box.
- Select Detail settings from the Simulate pull down menu.
- Enter the size of the workpiece in the appropriate data fields.
- Relocate the PRZ to the end face by checking the Origin at end face option.
- Specify radius coordinates or diameter coordinates by clicking on the appropriate radio button. Note that this text uses only Radius coordinates.
- Click OK when complete.
- Specify the path to the correct lathe tools (inch or millimeter) to be used in the simulation.
- Select Lathe tool path from the Simulate pull down menu.
- When you use the installation procedures listed in this chapter, the millimeter lathe tools will have the path: C:\Program Files\Micro Tech\CncSimulator\t_tools.
- When you use the installation procedures listed in this chapter, the inch lathe tools will have the path: C:\Program Files\Micro Tech\CncSimulator\ t_tools_inch\ t_tools.
- Review the lathe tools and identify the tool number(s) of the appropriate tool(s) for the program to be simulated.
- Select Edit tools from the Simulate pull down menu.
- Browse through the tools by selecting the arrow buttons (<<<) or (>>>) located on the two bottom corners of the dialog box.
- When the desired tool is identified, make note of the tool number.
- Click OK when complete.
- Type the program in the Code editor pane.

From the **Edit** pull down menu select **Automatic line number**. Enter the desired start number and increase value in the **Automatic line numbers** dialog box. Refer to Figure 5-51.

- Type the program into the code editor.
- Check the program code for syntax errors.
- Select **Check code** from the **Edit** pull down menu.
- Fix any code syntax errors accordingly.
- Simulate the program.

Open the simulation dialog box by pressing the **Simulate** button located in the status pane. Refer to Figure 5-22.

Start the simulation by pressing the **Automatic Simulation** button. Control the speed of the simulation by adjusting the slider control. Refer to Figure 5-23.

Note it is also possible to step through the program, command block-by-command block, by selecting the **Single Line Simulation** button.

Additional simulation options are available by right clicking in the **Code editor** pane next to any line of code.

- Save the program code and workpiece options.
- Select **Save** from the **Files** pull down menu.
- When prompted, select **Yes** to save the workpiece options.

In Section 4.6.1 a CNC turning program was developed (in millimeter units) to produce the workpiece shown in Figure 5-59. The completed program sheet is shown in Figure 5-60. Note that the tool will be a right hand tool approaching the workpiece from above. Using the simulation process steps listed above, simulate this program.

Figure 5-59 Turning example product drawing

Program Sheet

Program Number:	1			Part Name:			Lathe Example5.6.1					Sheet No.:	1		
Programmer:		Kandray										Date:	xx/xx/xx		

Process Flow/Tool Path Description	N	G - Code		Axis Coordinate			Center of Arc			Radius of Arc	F	S	T	M-Code
				X	Y	Z	I	J	K	R				
Set absolute coordinates and metric units	N10	G90	G71											
Change the tool	N20												T1	M06
Rapid move to a safe position, R0	N30	G00		X60		Z24								
Turn on the spindle CW and set the RPM's and feed rate	N40										F180	S1520		M02
Rapid move to a position slightly off the part (R1)	N50	G00		X50		Z1								
Linear Move into work for 1st cut (R2)	N60	G01		X44		Z1								
Linear move to R3	N70					Z-85.6								
Linear move to R4	N80			X50		Z-96								
Rapid back to R2 for the next cut	N90	G00		X44		Z1								
Linear into work - R5	N100	G01		X38										
Linear move to R6	N110					Z-75.22								
Linear move to R3	N120			X44		Z-85.6								
Rapid back to R5 for the next cut	N130	G00		X38		Z1								
Linear into work - R7	N140	G01		X32										
Linear move to R8	N150					Z-64.82								
Linear move to R6	N160			X38		Z-75.22								
Rapid back to R7 for the next cut	N170	G00		X32		Z1								
Linear into work - R9	N180	G01		X27										
Linear move to R10	N190					Z-49.74								
Linear move to R11	N200			X30.22		Z-61.72								
Linear move to R8	N210			X32		Z-64.82								
Rapid back to R9 for the start of the finish cut	N220	G00		X27		Z1								
Linear into work - F1	N230	G01		X25										
Linear move to F2	N240					Z-50								
Circle interpolate CW to F3	N250	G02		X28.35		Z-62.5				25				
Linear move to F4	N260	G01		X50		Z-100								
Rapid back to Safe Position - R0	N270	G00		X60		Z12								
Shutdown spindle	N280													M05
End Program	N290													M30

Figure 5-60 Turning example completed program sheet

Note the following:

If this simulation is performed immediately after the milling simulation performed in the last section, the milling program will need to be closed. This is accomplished by selecting the Close Buffer button located in the Code editor pane.

Figure 5-61 shows the settings for the lathe options dialog box. Make sure that the Origin at end face option is selected. Note how this moves the PRZ symbol to the right end of the workpiece in the graphics area. All examples in this text, by convention, place the origin at the end face. Additionally, select the radius coordinates radio button. Radius coordinates dictate that values specified in the *x* direction represent the radius of the workpiece, whereas with diameter coordinates, values specified in the *x* direction represent the diameter of the workpiece. Consider Figure 5-62. This figure shows the same program simulated first with radius coordinates, Figure 5-62 (a), and then diameter coordinates, Figure 5-62 (b). Note that the workpiece simulated with radius coordinates is *twice* as large as the diameter coordinate workpiece. This is because the diameter is twice the radius. Thus, when X30 is specified using radius coordinates, the actual diameter is 60, whereas an X30 specification using diameter coordinates will yield an actual diameter of 30.

A right hand tool approaching from above is Tool 1.

When entering program code in the Code editor pane, hit Enter at the end of each command block to move to the next line. Be sure to enter the correct tool number in the program code.

Fix any syntax errors and repeat the check code procedure until the warning dialog box no longer appears.

When the simulation is complete, the screen should look similar to Figure 5-63. Be sure to save the program and workpiece options.

Figure 5-61 Lathe options dialog box settings

Figure 5-62 (a) Radius coordinates (b) Diameter coordinates

5.5 Summary

The process of creating G-code programs is a complex, challenging process with many opportunities for error. Error-ridden programs executed on production equipment can cause damage to workpieces, tooling, and machinery, and they can jeopardize operator safety. Consequently, CNC simulation software was developed to verify that G-code programs function as intended prior to execution on an actual machine. This is the modern method of program code verification.

CncSimulator® is a free, Returnware software program that provides very good simulation of milling, turning, and gas cutting G-code programs. The user interface consists of menus and toolbars for entering commands along with three user selectable simulation windows or panes. Additionally there is a code editor pane for program manipulation and a status pane for simulation process control.

CncSimulator® utilizes an interpreter to convert the CNC program code into movement data for display on the screen. The default interpreter recognizes only standard ISO codes. In general, program code not recognized by the interpreter is ignored. Thus, programs written for a specific machine with codes not recognized by the interpreter may have to be tailored to run in CncSimulator®. In order to tailor a program to run on both the simulator and the actual machine, the user must identify and document differences between the codes.

Simulation is easily accomplished if the user sets the screen to the appropriate process, specifying workpiece and PRZ settings, setting up the appropriate tools, entering the program into the simulator, checking for errors, and finally simulating the program. Simulation and editing of the program are repeated until an error-free program is produced, ready for execution on the actual machine.

Figure 5-63 Completed turning simulation

5.6 Key Words

automatic simulation	program code verification
CNC simulation software	radius coordinates n
CncSimulator®	Returnware
code editor pane	simulation dialog box
diameter coordinates	single line simulation
interpreter	standard toolbar
lathe options dialog box	status bar
manual prove-out	status pane
milling options dialog box	text buffer
milling tool dialog box	user interface
nullpointX	
nullpointY	

5.7 Review Questions

1. Define program code verification as it relates to CNC machining.
2. List and describe the two methods available to verify a G-code program. Which one is used most often and why?
3. What are the benefits of using CNC simulation software?
4. Which processes does CncSimulator® provide for program code verification?
5. Define user interface and describe the main elements of CncSimulator®'s user interface.
6. List and describe the six different simulation views available for milling and gas cutting simulations.
7. In which pane is the G-code displayed and edited?
8. Describe the difference between a docked and floating toolbar.
9. How many simulation pane views are available for turning simulations?
10. Which simulation view, available only for milling, is adjustable?
11. Define the function of CncSimulator®'s interpreter.
12. List the letter addresses, G- and M-codes, supported by the default interpreter for the milling process.
13. List the letter addresses, G- and M-codes, supported by the default interpreter for the turning process.
14. What action does the interpreter take when it encounters an unsupported G-code during a simulation?
15. What information is provided about a G-code program when Program Statistics is selected from the Edit pull down menu?
16. List the information contained in the milling options dialog box.
17. What is the purpose of the text buffer?
18. What simulation options are available from the simulation dialog box?
19. For milling simulations, how is the PRZ moved to different locations on the workpiece?
20. Explain the difference between radius and diameter coordinates.
21. List the steps necessary to simulate and verify a CNC milling program in CncSimulator.
22. List the steps necessary to simulate and verify a CNC turning program in CncSimulator®.
23. Figure 5-64 is the program sheet for milling the part shown in Figure 5-65. The tool to be used is a ¼-inch end mill. Enter the program into CncSimulator® as listed in Figure 5-64 and simulate it.

Program Sheet

Program Number: 1003		Part Name:	Review Question 23						Sheet No.: 1			
Programmer:		XYZ							Date: -			

N	Process Flow/Tool Path Description	G - Code	X	Y	Z	I	J	K	R (Radius of Arc)	F	S	T	M-Code
N110	Set absolute coordinates and inch units	G90 / G70											
N120	Change the tool											T17	M06
N130	Move to a safe position, 1 inch above work piece	G00	X0	Y0	Z1								
N140	Turn on the spindle CCW and set the RPM's and feed rate				Z0.1					F90	S4000		M03
N150	Move to a position directly above the work piece	G00	X1.5	Y1.0									
N160	Move to center of pocket, just above work piece	G00			Z0.1								
N170	Start Pocket	G01			Z-.375								
N180	Linear move to start of 1st circle interpolation	G01	X1.6725	Y1.0									
N190	1st circle interpolation	G02	X1.6725	Y1.0		I-.1725	J0						
N200	Linear move to start of 2nd circle interpolation	G01	X1.845	Y1.0									
N210	2nd circle interpolation	G02	X1.845	Y1.0		I-3.45	J0						
N220	Linear move to start finish cut of Pocket	G01	X2.0303	Y1.0									
N230	1st linear pocket cut	G01	X1.5	Y1.5303									
N240	2nd linear pocket cut	G01	X.9697	Y1.0									
N250	3rd linear pocket cut	G01	X1.5	Y.4697									
N260	4th linear pocket cut	G01	X2.0303	Y1.0									
N270	Rapid out of work	G00			Z0.1								
N280	Move to center of 1st Hole	G00	X.5	Y.25									
N290	Drill 1st Hole	G01			Z-.375								
N300	Rapid out of work	G00			Z0.1								
N310	Move to center of 2nd Hole	G00		Y1.75									
N320	Drill 2nd Hole	G01			Z-.375								
N330	Rapid out of work	G00			Z0.1								
N340	Move to center of 3rd Hole	G00	X2.5										
N350	Drill 3rd Hole	G01			Z-.375								
N360	Rapid out of work	G00			Z0.1								
N370	Move to center of 4th Hole	G00		Y.25									
N380	Drill 4th Hole	G01			Z-.375								
N390	Rapid out of work	G00			Z0.1								
N400	Move to safe position	G00	X0	Y0	Z1								
N410	Shutdown spindle												M05
N420	End Program												M30

Figure 5-64 Question 23 program sheet

Figure 5-65 Question 23 product drawing

24. Figure 5-66 is the program sheet for turning the part shown in Figure 5-67. The tool to be used is lathe tool number 4, which is a profiling tool. Enter the program into CncSimulator® and simulate it. Be sure to specify radius coordinates and move the PRZ to the end face.

Program Sheet

| Program Number: 1 | | | | | | | | Part Name: | | | Review Question 24 | | Sheet No.: 1 |
| Programmer: | | Kandray | | | | | | | | | | Date: | xx/xx/xx |

N	G - Code	X	Y	Z	I	J	K	R (Radius of Arc)	F	S	T	M-Code	Process Flow/Tool Path Description
N10	G90 G71												Set absolute coordinates and metric units
N20	G00										T1	M06	Change the tool
N30	G00	X40		Z3									Rapid move to a safe position, R0
N40	G00								F180	S1520		M03	Turn on the spindle CW and set the RPM's and feed rate
N50	G00	X40		Z-21.4142									Rapid move to a position slightly off the part (A2)
N60	G01	X37.5											Linear Move into work for 1st cut (A2)
N70		X34.5		Z-24.4142									Linear move to A5
N80				Z-110.5858									Linear move to A6
N90		X37.5		Z-113.5858									Linear move to A3
N100	G00	X34.5		Z-24.4142									Rapid back to above A5 for the next cut
N110	G01	X31.5		Z-27.4142									Linear move to next cut (A5)
N120				Z-107.5858									Linear move to A7
N130				Z-110.5858									Linear move to A8
N140		X34.5		Z-27.4142									Linear move to A6
N150	G00	X31.5		Z-27.4142									Rapid back to above A7 for the next cut
N160	G01	X28.5		Z-30.4142									Linear move to next cut (A7)
N170				Z-104.5858									Linear move to A9
N180				Z-107.5858									Linear move to A10
N190		X31.5		Z-30.4142									Linear move to A8
N200	G00	X28.5		Z-30.4142									Rapid back to above A9 for the next cut
N210	G01	X26		Z-32.9142									Linear move to A9
N220				Z-102.0858									Linear move to A11
N230				Z-113.5858									Linear move to A12
N240		X37.5											Linear move to A3
N250	G00	X38.5		Z-20				25					Rapid back to above A1 for the Finish cut
N260	G01	X25		Z-32.5									Linear move to A13
N270				Z-102.5									Linear move to A14
N280		X37.5		Z-115									Linear move to A4
N290	G00	X40		Z5									Rapid back to Safe Position
N300												M05	Shutdown spindle
N310												M30	End Program

Figure 5-66 Question 24 program sheet

Figure 5-67 Question 24 product drawing

5.8 Bibliography

1. Cnc Simulator Software, Help Screens, release 4.44f, published by Bulldog Digital Technologies.

Chapter 6

Introduction to Robotics Technology

Contents

Objective

The objective of this chapter is to provide a thorough understanding of the terminology and basic operating concepts of industrial robots.

6.1 Industrial Robotics

The term "robot" has its origins in the Czech language. It first appeared in a play in 1921. It was used to describe artificial people produced in a factory to serve humans. It is derived from the word *robota* which means "serf labor." In eighteenth century Europe a serf was a person bound to an estate and forced to work the land as so directed by the lord of the estate. Serfs were viewed as the lowest social class, essentially slaves. As such, their feelings toward the work they performed were of little concern to their overlords, who had them perform the most undesirable and difficult work. Hence, "robot" came to mean a machine with human characteristics, but the performed work unsuitable for the typical human—essentially a machine in servitude. A standard dictionary definition of a robot names it "a machine that resembles a human and does mechanical, routine tasks on command" (www.dictionary.reference.com).

Figure 6-0 Human attributes of an industrial robot arm

Industrial robots are capable of performing a variety of tasks as directed by a program. For industrial applications, the a robot's humanlike characteristics include a mechanical arm, as is shown in Figure 6-0, the ability to make decisions based on sensory input, and the capability of communicating with its environment. Robots were first introduced, and are still often used, to replace human workers in the performance of mundane, difficult, or dangerous tasks. Initially, they were developed to perform at least as well as a human laborer. However, as we shall see, their modern capabilities go far beyond those of even the best laborer in terms of speed, accuracy, repeatability, and reliability. Their programmability and reprogrammability place robots in category of

programmable automation. Accordingly, the International Standards Organization (ISO) standard ISO/TR/8373-2.3 defines an industrial robot as:

…an automatically controlled, reprogrammable, multipurpose, manipulative machine with several reprogrammable axes, which may be either fixed in place or mobile for use in industrial automation applications.

Although at first robots were used mainly in stand-alone applications to replace human laborers in material handling applications, robot technology rapidly matured. Now robots are cheaper, more reliable, and much more capable; they can be used in ways that go far beyond stand-alone applications. In fact, the majority of new applications for robots involve integration into automation cells, often working in conjunction with part feeders, conveyors, CNC machines, and programmable logic controllers (PLCs). A typical automation cell is shown in Figure 6-1.

Figure 6-1 Automation cell

The robot in this cell quickly removes two different sizes of molded parts from the shuttle plate and places them on the staging plate. As the shuttle moves back into the press to mold the next heat of parts, the robot lifts the parts off of the staging plate, routes their inside and outside diameters on the router, then places them on the scale for weighing. Thus, the robot performs material handling, machining, decision-making, and communication tasks. However, the robot does not coordinate the action of the cell; this is accomplished through a PLC, with which the robot corresponds. Additionally, the robot has to make many decisions, such as when to move a part to a specific position, when to open or close the gripper, detecting the size of the part, when to unload the shuttle, and when to route parts—to name a few. This example indicates the tremendous potential of robots to aid in productivity improvement.

In this chapter we cover robot capability and the best use of industrial robots to improve productivity in factories. What follows is a thorough discussion of robot terminology, applications, hardware, configurations, and selection considerations. Robot programming is covered in Chapter 7.

Figure 6-2 Robot hardware

6.2 Robot Hardware

Robot technology terminology is loosely divided into *physical makeup*, also known as "robot hardware," and *performance capability*. Figure 6-2 depicts the basic hardware of any modern industrial robot; these include a *mechanical arm*, an *end effector*, a *power source*, a *robot controller*, and a *teach pendant*.

6.2.1 Mechanical Robot Arm

The mechanical arm consists of rigid *links* connected via mechanical *joints*. The joint allows relative movement between the input link and output link. There are five types of joints, as shown in Figure 6-3. A *linear joint* allows translational motion between the input and output links. Additionally, the axis of the input link is parallel to the axis of the output link. An *orthogonal joint* also enables translational motion between the input and output links; however, the axis of the input link is perpendicular to the output link. Rotational motion between the input link and output link can be achieved with a *rotational joint*, *revolving joint*, or *twisting joint*. The type of joint used to achieve the rotational motion is dependent on the orientation of the rotational axis relative to the input and output link axes. For instance, for a joint to be classified as a rotational joint, the axis of rotation must be perpendicular to the axis of the links. For a revolving joint, the axis of the input link is parallel to the axis of rotation, but the axis of the output link is perpendicular to the axis of rotation. Finally, for a twisting joint the axis of rotation is parallel to the axes of both the input and output links.

Each joint axis of the robot arm gives one *degree of freedom* of movement. Combining multiple links and joints yields multiple degrees of freedom. The types of joints that are combined dictate robot configuration and corresponding range of arm motion. In the robot arm shown in Figure 6-4, each of the joints has been labeled and the joint rotational axis identified. This information allows us to identify the type of joints used in this particular robot configuration. It is an industry convention that joints be identified starting at the base of the robot and proceeding through the links and joints to the end effector. ("End effector" is a general term for the tooling connected to the end of the robot arm.) The first joint we consider is the *waist joint*.

The waist joint connects the robot base with what can be considered the robot's *torso link*. The torso link is represented in Figure 6.4 by the structure and motors between the waist joint and the shoulder joint. The torso link axis runs from the waist joint to the shoulder joint or perpendicular to the waist joint. Thus, the axis of rotation is parallel to the input link, but the output link axis is perpendicular to it. So, this would be considered a *revolving joint*.

The next joint is the *shoulder joint*. It is a rotational joint because the axis of rotation is perpendicular to both the input link (torso) and the output link. Note that the elbow joint is also a *rotation joint*. The forearm link axis is parallel to the forearm rotational axis and the wrist link. Accordingly, this joint is a *twisting joint*. Wrist joint 1 is considered a

rotational joint and wrist joint 2 a twisting joint, an identification that is based on the input and output link axes' relationship to the joint rotation axes of the preceding two joints.

Figure 6-3 Robot arm joint types

Figure 6-4 Robot joint example

The first three joints of the robot of Figure 6-4—the waist, shoulder, and elbow—are considered *position axes* because they move the end effector into position to do work. These axes make up the body and arm assembly of the robot. The next three joints—the forearm, wrist 1, and wrist 2—make up the wrist assembly; they are *orientation axes* because they orient the end effector to perform the desired task. The mechanical robot arm of Figure 6-4 is an articulated robot arm with six degrees of freedom (six joints); each joint has rotational motion between its input and output link. This is typical of this kind of configuration. However, other types of rotational motion joints in various combinations can also be used in a construction of an articulated arm robot.

Even more distinct configurations can be constructed with linear joints exclusively or linear joints in combination with rotational motion joints. Such configurations include a *polar* robot configuration, a *cylindrical* robot configuration, a *Cartesian coordinate* robot configuration, and a special type of articulated robot arm called a *selective compliant assembly robot arm*, or *SCARA* for short. These configurations are shown in Figure 6-5.

Work
Envelopes

Cylindrical
Robot
Configuration

Cartesian Coordinate
Robot
Configuration

SCARA
Robot
Configuration

Figure 6-5 Robot configurations

A summary of robot configurations follows:
- The polar robot consists of a twisting joint, a rotational joint, and a linear joint.
- The cylindrical robot has a twisting joint and a linear and orthogonal joint.
- The Cartesian coordinate robot is made up of two orthogonal joints and one linear joint.

The SCARA robot has a twisting joint, a linear joint, a rotational joint, and another twisting joint, where the end effector is attached. SCARA robots are typically used in high speed assembly applications, where vertical stability is important.

Often, the defining feature of the different configurations is the *work envelope*. The work envelope is the space within which the end effector can be moved without limitations. It is essentially the distance within which the robot can reach and do work. This is often the primary consideration in the selection of a robot for a specific application.

The different robot configurations shown in Figure 6-5 each have distinct work envelopes. As can be seen in the figure and as the name implies, the polar robot has a polar- or spherical-shaped work envelope. The cylindrical robot has a cylindrical work envelope. Since the Cartesian coordinate robot has linear motion joints its work envelope is a rectangular cube. The SCARA has a somewhat cylindrical work envelope. An

articulated arm robot has an irregular work envelope dependent on the joints used and corresponding range of motion of each joint.

Note that the robot configurations seen in Figure 6-5 show only the joints that make up the body and arm assembly. The wrist assembly is attached to the end of the body and arm assembly and orients the end effector to the work. Wrist assemblies typically add two to three additional degrees of freedom. In the parlance of aircraft terminology, these additional axes are said to provide pitch, yaw, and roll capabilities to the end effector. Two standard wrist joint configurations are shown in Figure 6-6.

Figure 6-6 Wrist assemblies

The robot arm configurations discussed thus far are called *serial robots* because the joints and links are assembled in a serial fashion. As such, they are *open loop* mechanisms. There is another type of configuration, a *closed loop* mechanism, which is gaining popularity in some applications. This configuration has both passive and active (powered) joints and links. This type of robot is called a *parallel robot*. These robots can have several different configurations, a few of which are shown in Figure 6-7.

Figure 6-7 Parallel robot configurations

The end effector in a parallel robot configuration is mounted to the parallel plate, which is connected to the base through some combination of active and passive links and joints. In the configuration on the figure's left, the joints are all passive and the six links are active. The configuration on the right has four active joints combined with passive links and eight passive joints. Note that parallel configuration robots do not physically resemble a human arm but still satisfy the ISO definition of a robot. A cylindrical/cone shape defines the work envelope of parallel robots.

Although parallel robots offer some interesting capabilities in terms of position accuracy and high speed, they have a limited work envelope. Consequently, the open loop configurations remain the most popular, particularly for robots with articulated arm and for Cartesian coordinate robots.

6.2.2 End Effectors

The end effector is also referred to as *end-of-arm tooling*. Recall that a robot is a multipurpose machine. Thus, the application in which the robot is used dictates the type of end effector. Consequently, a wide variety of end effectors are available.

The most commonly recognized end effector is the *gripper*. Grippers are used in material handling applications to grasp and manipulate objects. The major types of grippers include *vacuum grippers*, *magnetic grippers*, *simple mechanical device grippers*, and *mechanically actuated grippers.*

Vacuum
Gripper

Magnetic
Gripper

Figure 6-8 Vacuum and magnetic grippers

Vacuum grippers are essentially vacuum-actuated suction cups. They are often used to pick up smooth fragile objects, such as sheets of glass, plastic sheets or bags, and thin metal sheets. Magnetic grippers use either electromagnets or permanent magnets with air release to pick up magnetic objects. Scoops or hooks would be considered simple mechanical devices. Examples of a vacuum and magnetic gripper are shown in Figure 6-8.

Mechanically actuated grippers are by far the most common type of gripper. They have fingers that are mechanically actuated to open or close to grasp objects. Gripper fingers are typically custom-machined so they match the object to be manipulated and then bolted to the gripper jaws. The gripper jaws are most often pneumatically actuated, but electrically actuated grippers are also available. The gripper's jaws can be actuated in either an angular or linear fashion. The number of jaws/fingers depends on the shape of the object to be manipulated. Two-, three-, and four-jaw grippers are common. Figure 6-9 shows a two-jaw gripper that is angularly actuated and a three-jaw linearly actuated gripper. Many other configurations are available.

Figure 6-9 Two- and three-finger grippers

Some material handling applications call for multiple grippers to be simultaneously mounted to the end of the robot arm. A *dual gripper* setup has two grippers on the end of the robot arm. This enables the loading and unloading of objects in one arm motion, thereby saving process cycle time. A dual gripper setup is shown in Figure 6-10.

Figure 6-10 Dual gripper setup

The robot manufacturer does not supply grippers; they are designed by the automation engineer for integrating the robot into the process under consideration. Thus, the engineer selects the appropriate gripper for the application from an independent supplier and designs fingers for optimum location and maximum grip of the object manipulated. The selection considerations include number of jaws, type of jaw actuation (angular or linear), mode of actuation (pneumatic or electric), available grip force, type of return, opening/closing speed, and, perhaps most important, gripper weight.

A gripper or any end effector's weight is of critical importance in the integration of a robot into a process because it lessens the robot arm's available *payload capacity*. Payload capacity is the maximum weight the robot arm can carry and still meet the robot's designed speed and acceleration/deceleration performance capabilities. Thus, if the application is using a robot with a payload capacity of 2 kg and the object being manipulated weighs 1 kg, the gripper assembly cannot weigh more than 1 kg.

Robots are often used in non-material handling applications. Thus, depending on the application, the end effector may be some type of tool. In this case, instead of manipulating an object, the robot manipulates the tool relative to a stationary or slowly moving object. Additionally, it controls the tool in terms of starting, stopping, and regulating its operation. Example end effector tools include welding guns, paint spray guns, rotating spindles, heating torches, and laser cutting tools.

6.2.3 Power Sources

A reliable and readily controllable power source is required to actuate a robot arm. The three types of power sources used to power robots include *hydraulic systems*, *pneumatic systems*, and *electrical systems*.

Hydraulic systems are closed loop systems in which electronically controlled valves direct the flow of pressurized hydraulic fluid to the appropriate linear or rotary actuator to achieve the desired robot arm motion. Figure 6-11 shows a typical closed loop hydraulic system. The pump, driven by an electric motor, draws the hydraulic fluid from the tank, pressurizes it, and forces it through the hydraulic lines. If the control valve is closed the pressure relief valve is opened by the pump-generated pressure and the hydraulic fluid flows back into the tank in a closed loop. If the control valve is open, as shown in the figure, the pressurized fluid flows through the control valve and into the actuator, thereby retracting the cylinder. As the cylinder retracts, the fluid to the left of the piston is forced through the control valve back to the tank. The cylinder can be extended by changing the position of the spool in the control valve. This is accomplished electronically through the robot's control system. For simplicity, the figure shows only one control valve and one actuator. In an actual hydraulic robot there would be at least one actuator per degree of freedom.

Figure 6-11 Hydraulic power source

Hydraulic fluid systems can be either oil over water or straight oil. In either case the fluid is highly incompressible, enabling good speed and positional control of the actuator.

Additionally, hydraulic systems provide a high power-to-size ratio with large payload capacities. Thus, when a robot is needed to carry a large load, a hydraulically actuated robot may be a wise choice. However, there are some serious limitations that need to be considered.

The support system consisting of the tank, motor, lines, and control valves substantially increases the cost and footprint of the robot. Also, hydraulic systems in general tend to be noisy and to leak. Thus, they are messy and require regular maintenance. Additionally, oil-based systems can be a fire hazard. Because of these concerns, hydraulic robots are not used as often as pneumatic and electromotive robots.

Pneumatic robots use many of the same components as hydraulic robots in terms of actuators and control valves. However, the two systems have very different performance capabilities. This is primarily due to the compressibility of the air in the pneumatic systems, which makes difficult the accurate control of position and speed—a major disadvantage of pneumatic robots. Air systems generally use adjustable hard stops for location control. Another difference between air and hydraulic systems is that air systems are open loop, versus the closed loop of hydraulic systems. This is an advantage, as piping is simpler in air systems because no return lines are needed.

Another—major—advantage of pneumatic robots is the readily available supply of air. Most manufacturing facilities have compressed air systems already installed. Thus, a separate support system is not required, resulting in substantial savings compared to what is possible for a hydraulic system. Also, leaks in pneumatic systems do not contaminate the surrounding environment.

Because they cannot achieve precise speed and position control, the use of pneumatic robots is limited, primarily, to pick and place type operations. However, pneumatic technology is well developed, yielding reliable and affordable robots. A typical pneumatic robot is shown in Figure 6-12. Cartesian coordinate robot configurations with two or three degrees of freedom are the most common. Pneumatic robots find many uses in the plastics industry primarily for sprue pulling and molded part removal.

Electrically actuated robots are by far the most common type of robots in industry today. They use electrical servomotors, enabling precise speed and position control. The rotational position is proportional to an electrical command signal. These robots are controlled by a closed loop control system allowing accurate and repeatable position control.

Figure 6-12 Pneumatic robot

The servomotors may be AC or DC. AC motors are more stable, light, and rugged. Additionally, they require less maintenance than DC motors, which require regular brush replacement. However, DC motors generate more torque, thereby justifying their use in high strength applications.

To power a robot arm joint, servomotors generally require some type of rotational motion speed reduction and corresponding torque amplification. This is accomplished with a *reduction drive*. Common types of reduction drives for rotary motion include pulleys and synchronous timing belts, gear trains—including spur, worm, and bevel, and harmonic drives. Linear motion is accomplished with ball screw mechanisms. Figure 6-13 shows examples of each. Most engineers and technologists are familiar with the range of speed reductions and torque increases possible with traditional synchronous timing belts, various gear trains, and ball screw mechanisms. However, the harmonic drive is not as common and offers some unique capabilities, which can be put to good use in robotic applications. Next we consider the harmonic reduction drive.

Synchronous Timing Belt

Spur Gears

Worm Gears

Bevel Gears

Circular Spline (Fixed)

Flexspline (Output)

Ball Bearings

Elliptical Wave Generator (Input)

Harmonic Drive

Ball Screw Mechanism

Figure 6-13 Reduction drives

Harmonic drives were invented in the 1950s. They provide high reduction ratios and corresponding large torque amplification in a condensed package. This drive's operating principle is relatively simple but somewhat difficult to visualize. A harmonic drive, as shown in Figure 6-14, consists of an elliptical wave generator, flexspline, ball bearings, and circular spline. The elliptical wave generator serves as the input to the mechanism and the flexspline is the output. The flexspline has gear teeth on its outside diameter that mesh in two locations 180 degrees apart with gear teeth on the inside diameter of the circular spline. Note that the circular spline is fixed and does not rotate. As the wave generator rotates, the ball bearings cause the flexspline to deform, thereby moving the mesh point around the inside of the circular spline. Because the flexspline has two less teeth than the circular spline, the flexspline rotates in the opposite direction of the elliptical wave

generator. This unique construction enables the harmonic generator to have zero backlash for very high positional accuracy. This, in turn, makes the drives ideally suited for electric drive robotic joint applications.

In some robotic cases no reduction drive is used and the motor drive shaft can be directly connected to the joint axis. This is called a *direct drive*. It yields a one-to-one relationship in motor-to-joint speed and torque. Obviously, this can only be used in rotational motion applications where speed is more important than torque.

Figure 6-14 Harmonic drive

The advantages of electric drive systems include the following: no danger of contamination of the work environment; no separate power source (hydraulic) requirement; and low noise level. Additionally, electric drive systems have the fastest arm motion with the quickest response. The primary disadvantage of the electric drive system is payload capacity, which is typically limited to 250 pounds.

6.2.4 Robot Controller and Teach Pendant

The robot controller is a special purpose computer that has three major functions: First and foremost, it controls the actuators that direct arm motion; second, it provides the means for the robot to interact with and control periphery equipment and end effectors; third, it serves as the user interface from which the robot can be manually controlled and programs can be entered, edited, and executed. These functions are depicted in Figure 6-15.

Figure 6-15 Robot controller functions

The performance capabilities of any robot are solely dependent on the type of arm motion control the robot controller executes. Consider a pneumatic robot used in a pick and place application versus an arc-welding robot. Each case requires a very different level of control over the robot arm. In a pick and place application the robot moves to a position to grasp an object, then to another location to release the object. The pneumatic robot controller sequences the actuation of each joint to achieve the desired motion. However, the actual stopping position of each joint is set with limits or mechanical stops. This type of control is called *limited sequence control* and is the most basic type of robot motion control. On the other end of the spectrum is *continuous path control*. Robots using continuous path control are capable of moving to precise locations, without the aid of mechanical stops, while following a specified path. Continuous path control robots not only simultaneously control the position of each robot axis but also the velocity and acceleration of each joint to achieve the programmed path. This type of controller utilizes interpolation routines similar to CNC control. Thus, the joint actuators are typically electrical servomotors. An arc-welding robot tasked to create a weld on a curved path would require continuous path control.

A limited sequence control robot controls the joint actuators with open loop control. Thus, as discussed in Chapter 3 on CNC control, there is no feedback system to verify that

the axis has reached the desired position. For pneumatic robots this necessitates the use of mechanical hard stops. An open loop control system for a pneumatic robot is shown in Figure 6-16. The signal to move the axis is sent to the actuator—in this case the control valve—and the controller assumes the axis has achieved the desired position at the desired speed. However, because of the lack of feedback this assumption cannot be verified. Although the lack of a feedback system does limit the capabilities of this type of robot, it also greatly simplifies the controller. That is why limited sequence control robots can often be controlled with a standard programmable logic controller (PLC). The capabilities of programmable logic controllers are discussed in more detail in Section 8.4.

Figure 6-16 Open loop controlled pneumatic robot axis

Continuous path control robots require closed loop feedback systems—one for each robot axis. The feedback system for the shoulder axis is shown in Figure 6-17.

Assume the robot is to follow the path shown in Figure 6-18. The robot program, which is stored in the controller memory, specifies these positions, along with the path and speed that the robot arm is to follow. In order to achieve the desired path, robot axes must move simultaneously in a coordinated fashion. Thus, an *executive controller* resolves the program information into individual axis movements, as shown in Figure 6-19. Each individual axis movement is then sent to the axis controller, which analyzes the signal and turns on and rotates the servomotor to obtain the desired joint position at the desired speed. The axis *encoder* informs, or feeds back, the position of the axis continuously to the axis controller, while the *tachometer* reports the rotational speed. The axis controller

continuously compares the desired position and speed to the actual position and speed, adjusting the servomotor accordingly. This is shown schematically in Figure 6-17.

Figure 6-17 Closed loop axis controller for the shoulder axis

Figure 6-18 Six-axis robot movement

Figure 6-19 Executive axis controller

The majority of modern electromotive industrial robots exercise continuous path control. Thus, they are capable of performing a wide variety of tasks requiring precise control of an end effector or tool. Examples include material handling, in which objects must be precisely grasped and placed, seam arc-welding, and even machining applications. In any of the aforementioned examples, control of the end effector in terms of opening and closing a gripper, or turning on or off a welding tool or spindle, also needs precise execution. The robot controller performs this function as well.

Control of end effectors and interfacing with auxiliary equipment within a robot manufacturing cell is called *peripheral equipment control*, which is a means for the robot to communicate with the surrounding environment through electrical connections. The robot controller monitors these connections and, depending on the status of the connection, takes some appropriate action as dictated by the robot program. For example, in the automation cell previously discussed and shown in Figure 6-1, the robot unloads parts from a mold, places them on a staging plate, routes the inside and outside diameters, and then places the parts on a scale for weighing. It could not perform these tasks unless it was aware of the status of the surrounding equipment. It has to know when the mold is in position for unloading, inform the press that the unloading is complete, and tell the scale when to weigh a part. Additionally, the gripper has to be opened and closed at precisely the right time. Such auxiliary equipment control is accomplished through *digital input/ output* (I/O) *interface boards*, located in the robot control cabinet, and also through an *end effector* (EE) *cable* routed through the robot arm and terminating near the end effector.

The I/O boards in the robot controller are hardwired to comparable boards located on the PLC, which coordinates all the action of the cell. Thus, when the mold is in position to be unloaded, the PLC turns on an output, which will show up as an input to the robot. The robot program is written in such a way that when the robot sees this input is on, it begins to unload the parts from the mold. Correspondingly, as the robot begins to unload the mold, it turns on an ouput, which is monitored by the PLC. The PLC programming is written so that it does not move the mold as the parts are unloaded. When the robot completes the unloading it turns off this output, which in turn informs the PLC that it is alright to move the mold plate. This is depicted in Figure 6-20.

Figure 6-20 Robot controller interfacing

The EE cable is terminated near the end effector in the form of a female connector. A male connector, hardwired to the end effector, is then plugged into the female connector. A typical connector setup is shown in Figure 6-21. The robot program contains instructions specifying when the gripper should be opened or closed. The robot controller turns these instructions into electrical signals, which are routed through the EE cable to actuate the end effector.

Figure 6-21 EE cable connectors

Figure 6-22 shows a robot control cabinet interface for a Fanuc RJ controller. Also visible in the figure is the teach pendant. A close-up view of the teach pendant can be seen in Figure 6-23. The control cabinet and teach pendant represent the operator interface. Through these devices the robot program is created, program files are manipulated and edited, and the robot is manually controlled. Specifically, the control cabinet provides a means to turn the robot on or off, to start or stop program execution, and/or to execute an emergency stop. In modern robots the teach pendant is the primary means through which programs are entered, edited, simulated, stored, and loaded for execution. Teach pendants typically consist of an LCD screen and pushbuttons or toggle switches for entering and executing commands. As will be explained in the next chapter, during the programming process the robot is manually moved to the positions to be programmed. Once the robot is in the desired position, the position information, often along with the means to achieve that position, is stored for later use in the program. These tasks are typically accomplished with the teach pendant. Specific programming techniques utilizing the teach pendant are discussed in the next chapter.

Figure 6-22 Robot controller

Figure 6-23 Robot teach pendant

6.3 Robot Applications

Robotic technology is highly valued in productivity improvement, as it is an affordable, reliable, and extremely capable in this respect. It has proven itself countless times in industry and is now finding applications in many nonindustrial applications, including the pharmaceutical and health care fields. Robotic technology should be considered whenever productivity improvements are sought. There are also specific work environments in which robotic technology tends to thrive. They include work environments that are hazardous or uncomfortable for workers, ones where tasks require a high degree of accuracy, and/or those that entail highly repetitious processes.

Hazardous or uncomfortable work environments are those where human workers may be exposed to high temperatures, dangerous atmospheric conditions, difficult work positions, such as kneeling or crouching, or even the handling of awkward objects—that is, any type of environment that causes significant stress on the worker. Specific examples include work that is done around die casting presses, forging presses, heat treating processes, and welding.

Work environments that require a high degree of accuracy or are highly repetitious have been proven to be good venues for robots. Modern robots are much more accurate and reliable than human workers. They do not become bored or distracted and are thus ideal for mundane repetitive work. Their application in this type of environment inevitably yields the added benefit of improved quality.

Robotic applications are categorized into three main areas:
1. Material handling,
2. Processing operations,
3. Assembly and inspection.

In material handling applications the robot moves a part from one location to another. In this type of application the end effector will be some type of gripper. In pick and place or material transfer applications the robot picks up an object from one location and moves or transfers it to another. In palletizing applications the robot places each subsequent part in a different location from the previous part. Machine loading and unloading are common material handling applications used for numerous types of processes, including die casting, machining, injection molding, pressworking, forging, and heat treating. Figure 6-24 shows some machine loading and unloading applications.

Figure 6-24 Robot machine loading and unloading

The most common processing operation applications for robots include spot-welding, arc-welding, and spray painting. In spot-welding processes the robot is typically a large six-axis, high payload robot that can easily manipulate the heavy spot-welding gun into tight spaces. Arc-welding processes are very appropriate for robotic technology because they are hazardous and very repetitive. This is also true of spray painting processes, another area in which robotic technology is heavily utilized. Machining is yet another processing operation that uses robots; in machining the robot uses a spindle for its end-of-arm tooling. Robots equipped in this manner can perform drilling, routing, and many other types of machining operations. Examples are shown in Figure 6-25.

Fig-6-25 Robot machining application examples

Assembly and inspection process are yet another area in which robot technology is often applied. SCARA and Cartesian coordinate robots are often used in these applications because they lend themselves well to assembly of components and have high vertical stability. Additionally, robots can be equipped with adhesive dispensers or soldering units for assembly of circuit boards and other electrical assemblies. In inspection applications the robot may transport either the part to the inspection device or the inspection device to the part. Some examples are shown in Figure 6-26.

Figure 6-26 Robot assembly application examples

6.4 Robot Safety

Worker safety is always of prime importance in an industrial setting, but perhaps even more so in robot installations. As discussed in the last section, a prime application for a robot is a work environment that is hazardous or uncomfortable for human workers. So, it would be ironic if the robot installation were more hazardous than the original work environment it replaced. Yet, this can easily occur if specific steps to ensure the safety of the worker, robot programmer, and maintenance personnel are not made.

During normal operation, a robot arm follows the path specified in the robot program, without regard for objects in its path. It is typically moving at a high rate of speed and often carrying a substantial payload. This combination of high speed and large payload results in considerable inertial loads. Additionally, the arm motion is often unpredictable to the casual observer. Thus, if an object, human or otherwise, inadvertently moves into the robot's work envelope during normal operation, a collision is likely and could very well be catastrophic. Even when not in normal operation, such as during programming or maintenance, the robot poses a risk to personnel. These scenarios often require personnel to enter the robot's work envelope to perform tasks. Thus, if the robot arm accidentally moves while the workers are within the work envelope, it is highly possible that the worker may collide with or become trapped between the arm and other objects. Therefore,

it is imperative that safeguards be in place to prevent personnel from entering a robot's work envelope during normal operation and that protect them while they are in the work envelope, performing necessary maintenance and programming tasks. We next discuss these measures.

6.4.1 Robot Safety Standards

The Occupational Safety and Health Administration (OSHA) is the federal agency tasked with enforcing worker safety. The Occupational Safety and Health Act issued by the federal government in 1970 is the most comprehensive U.S. law regarding worker safety. It authorizes the federal government to establish and enforce occupational safety standards for workers. Under this act employers are required to provide employees a safe place to work, free from dangers that cause or are likely to cause serious or fatal injuries.

OSHA publishes guidelines to aid employers in providing a safe workplace. Chapter 4 of Section IV of the *OSHA Technical Manual* deals specifically with robot safety. The chapter "Industrial Robots and Robot System Safety" provides a background on robots in general and addresses hazards, investigation guidelines, and control and safeguarding of personnel. The manual is available online at www.osha.gov.

The OSHA manual goes into considerable depth on the types of accidents that can occur and the sources of hazards. Types of worker injuries that are largely preventable through effective safeguarding can be summarized into three major categories:
1. Impact, collision, or blow from a robot arm.
2. Some body part being trapped or crushed between the robot arm and other peripheral equipment.
3. Impact, collision, or blow from released parts or end effector components due to mechanical failure.

It is important to note that these hazards exist not only for the production worker but also for programmers and maintenance personnel. Thus, safeguarding needs to include provisions for protecting everyone. In general, a combination of safeguarding measures is encouraged with emphasis on redundancy.

Throughout the OSHA manual reference is made to another standard developed jointly by the American National Standards Institute (ANSI) and the Robotics Industries Association (RIA). The standard, *ANSI/RIA R15.06 Industrial Robots and Robot Systems-Safety Requirements* is the de facto standard for robot safety. It is beyond the scope of this text to include an extensive detailed review of these standards. However, the next section addresses safeguarding considerations in a general sense. Thus, the reader is encouraged to review both the OSHA manual and ANSI/RIA R15.06 for additional detailed information on controlling and safeguarding personnel.

6.4.2 Safeguarding Considerations

Section V, Chapter 4, Section 4 of the *OSHA Technical Manual* offers controlling and safeguarding considerations. These considerations are summarized below.

1. A detailed review of the robot cell's operating characteristics and associated potential hazards should be performed and documented at each stage of robot cell development. This review is called a *risk assessment*. Based on this assessment the appropriate safeguarding to reduce or eliminate any risks can be applied.
2. Safeguarding devices should be used to restrict personnel access to the robot's work envelope. These devices include, but are not limited to, physical barriers, presence-sensing devices, E-stop devices, interlock devices, and safety control units.
3. Awareness devices can be used to make personnel aware that they are entering or approaching a hazardous area. Such devices include rope barriers and stanchions, flashing lights, signs, whistles, or horns. Awareness devices should be used, where applicable, with other safeguarding devices.
4. The robot programmer or "teacher" should be protected by limiting maximum robot arm speed to 250 mm/s or 10 in/s during programming.
5. When the robot is operating automatically all safeguarding should be in place with no opportunity for any part of the operator's body to enter the safeguarded workspace.
6. If personnel are required to be in or near the safeguarded workspace, the robot should be in teach mode, running at low speed, and said personnel should have full control of the robot arm.
7. Safeguarding maintenance and repair personnel is complicated because the tasks these workers perform are so varied. Thus, to ensure adequate protection refer to ANSI/RIA15.06 standard, Section 6.8.
8. Regular maintenance and inspection is critical for robot systems. It is needed to minimize hazards from component malfunctions and failures.
9. Extensive robot safety training is required for all personnel associated with the robot system operation, including but not limited to programmers, operators, and maintenance personnel.
10. The following sections of the ANSI/RIA R15.06 standard list additional requirements to be considered:
    - Section 6 – Safeguarding Personnel
    - Section 7 – Maintenance of Robots and Robot Systems
    - Section 8 – Testing and Start-up of Robots and Robot Systems
    - Section 9 – Safety Training of Personnel

Additionally, the robot system must comply with OSHA regulations OSHA 29 CFR 1910.333, Selection and Use of Work Practices, and OSHA 29 CFR Part 1910.147, The Control of Hazardous Energy (Lockout/Tagout).

6.4.3 Safeguarding Example

Figure 6-27 shows a top view of the robot cell discussed previously, in Section 6.1. Recall from that discussion that the robot removes parts from the mold shuttle, places them on the staging plate, routes the inside and outside diameter, and places the finished parts on the scale. Good parts are kicked out of cell and scrap parts are kicked into a scrap bin.

Figure 6-27 Robot safeguarding example

Note the robot's work envelope in the figure. As discussed in the last section, keeping personnel out of the robot's work envelope during normal operation is of prime importance. In this example this is accomplished with physical barriers and a presence-sensing device. Physical barriers can include almost any type of guarding or obstruction that may prevent a person or a person's body part from entering the robot's work

envelope, including other equipment in the cell. Therefore, in the figure the robot's control panel and the press serve as effective physical barriers. They are used in conjunction with other physical barriers such as wire mesh fencing.

The light curtain shown in the figure is a presence-sensing device. It is an electronic device that casts synchronized, parallel light beams to a receiver unit. If an opaque object interrupts the beam, the light curtain controller sends a stop signal to the robot cell controller. Thus, it prevents access to the work area when the robot is in motion, but allows easy access to the work area at other times. Other types of presence-sensing devices include pressure-sensing pads and ultrasonic or laser proximity-sensing devices.

Note that the removable physical barriers should be interlocked with the cell's controller or power supply. Thus, if a physical barrier is removed the robot or other equipment will not function.

Outside the physical barriers is a worker awareness zone. The intent of defining a worker awareness zone is to make personnel aware that they are near a potential hazard. The perimeter of this area could be marked with yellow caution paint in conjunction with flashing lights, physical barriers such as stanchions and ropes, and warning signs. Access to this area should be limited to cell operators, programmers, and maintenance personnel.

Note that this example is only intended to provide a cursory overview of safeguarding a robot system. For detailed information and requirements, refer to the OSHA manual and ANSI/RIA R15.06.

6.6 Robot Selection Considerations

Robots are incredibly capable machines. However, there is not one universal robot suitable for all applications. Thus, selecting the right robot for a particular application is critical to the success of a project.

The robot selection process starts with a thorough understanding of the application in which it is to be used. The application itself may indicate the type of robot to be considered. For example, an assembly application may suggest the use of a SCARA type robot for its vertical stability and high downward force. On the other hand, a welding application typically requires a robot with high dexterity and payload capacity to maneuver a heavy welding gun. Thus, a large six-axis robot may be indicated. Once the process requirements are fully understood, the following should be considered:

- robot arm geometry
- end effector requirements
- robot arm performance capabilities
- robot controller capabilities
- new versus used robot

Robot arm geometry is a prime consideration because it defines the robot's work envelope. The work envelope specifies the robot's range of arm motion. Thus, one must

ensure that the robot arm can reach all areas of the application's work area. Additionally, the robot geometry specifies the arm's dexterity. The robot arm must also be capable to maneuver the end effector through the range of motion specified by the application requirements.

The end effector type, size, and weight are significant contributors to the robot selection process. The type of end effector influences the robot controller requirements. For example, a welding robot application would require a special control module that would not be needed for a material handling application. The end effector size and weight dictate the payload capacity of the robot. If the type of end effector is a gripper, the weight of the part being manipulated must be added to the end effector weight to determine the total payload capacity required. The required payload capacity also plays a role in the type of power used for the robot. Extremely heavy payloads may require a hydraulically powered robot.

Robot arm performance capabilities refer to robot speed, accuracy, and repeatability. The robot selected must be able to satisfy the required application cycle times. If an application does not require high accuracy, such as an injection molding sprue-pulling application, an air-powered robot with mechanical hard stops may suffice.

Factors affecting robot controller capabilities include ease of programmability, familiarity of the robot program language, and ability to control or interface with peripheral equipment. The amount of equipment with which the robot must interface dictates the type and amount of input and output boards the controller will require.

Robots, like most types of automated devices, are expensive pieces of equipment. Some savings can be realized if a used robot is available that can meet the application performance requirements. Due to the large amount of offshoring in recent years, there are many used robots on the market that still have substantial life in them. Resellers often refurbish the robots as well. These robots can typically be purchased at a fraction of the cost of a new robot. Factors to consider when purchasing a used robot are age, hours of use, and availability of spare parts. A good rule of thumb regarding ease of acquiring spare parts for a used robot is to select a robot make and model that has been in widespread use. With many models in service, the probability of finding spare parts or even a spare robot arm greatly increases.

6.7 Summary

A modern robot is a machine that possesses humanlike characteristics and is capable of performing a wide variety of tasks. The ISO standard ISO/TR/8373-2.3 defines an industrial robot as an automatically controlled, reprogrammable, multipurpose, manipulative machine with several reprogrammable axes, which may be either fixed in place or mobile for use in industrial automation applications.

Initially, robots were used in stand-alone applications to replace human laborers in material handling applications. However, the majority of new applications for robots

involve integration into automation cells often working in conjunction with part feeders, conveyors, CNC machines, and PLCs. As a result, modern robots have tremendous potential to aid in productivity improvement.

Robot hardware includes a mechanical arm, end effector, power source, robot controller, and teach pendant. The mechanical arm consists of rigid links connected via mechanical joints. Different types of joints provide different types of motion. The five types of joints are linear, orthogonal, rotational, revolving, and twisting. Combining multiple links and joints yields a robot arm with multiple degrees of freedom. The combination of joint types dictates the robot configuration and corresponding range of motion possible by the arm. The five most common types of robot configurations include the polar robot, the cylindrical robot, the Cartesian robot, the articulated arm robot, and the SCARA robot. The defining feature of the different configurations is the work envelope.

"End effector" is the general term for describing the tooling connected to the end of the robot arm. As such, it is also referred to as end-of-arm tooling. Grippers are the most common type of end effector and are used in material handling applications to grasp and manipulate objects. In many applications the end effector is a tool. In this case, instead of manipulating an object, the robot manipulates the tool relative to a stationary or slowly moving object. Example end effector tools include welding guns, paint spray guns, rotating spindles, heating torches, and laser cutting tools.

The three types of power sources used to power robots include hydraulic systems, pneumatic systems, and electrical systems. Hydraulic power sources are used when the robot is required to carry a large payload. Pneumatically powered robots use a plant's readily available air supply to move the robot arm. However, because of the compressibility of air, position control is limited to mechanical hard stops. Thus, these robots have simpler control requirements. They are used primarily for simple pick and place applications. Electrically actuated robots are, by far, the most common robot type used in industry today. They use electrical servomotors whose rotational position is proportional to an electrical command signal. These enable precise speed and position control. Electrical servomotors are controlled by a closed loop control system, which allows accurate and repeatable position control. Electrically actuated robots have the fastest arm motion with the quickest response.

The robot controller is a special purpose computer that has three major functions: It controls the actuators that direct arm motion; it provides the means to interact with and control periphery equipment and end effectors; and it serves as the user interface from which the robot can be manually controlled and programs can be entered, edited, and executed.

The performance capabilities of a robot are solely dependent on the type of arm motion control the robot controller executes. Robots using continuous path control are capable of moving to precise locations, without the aid of mechanical stops. Limited sequence control robots are the most basic type of robot motion control. A limited

sequence control robot controls the joint actuators with open loop control. The majority of modern electromotive industrial robots exercise continuous path control.

Peripheral equipment control provides a means for the robot to communicate with the surrounding environment through electrical connections. The robot controller monitors these connections and, depending on the status of the connection, takes some appropriate action as dictated by the robot program.

The robot control cabinet and teach pendant represent the operator interface. Through these devices the robot program is created, program files are manipulated and edited, and the robot is manually controlled.

There are specific work environments in which robotic technology tends to thrive. They include hazardous or uncomfortable work environments, work environments in which a high degree of accuracy is required, and/or work environments that are venues of highly repetitive processes. Application areas include material handling, processing, and assembly and inspection.

Due to the high energy of a robot arm under normal operation, substantial safeguarding is required to protect operators, programmers, and maintenance personnel. Chapter 4 of Section IV of the *OSHA Technical Manual* titled "Industrial Robots and Robot System Safety" provides a background on robots in general and addresses hazards, investigation guidelines, and control and safeguarding of personnel. ANSI/RIA R15.06 Industrial Robots and Robot Systems-Safety Requirements is the de facto standard for robot safety; it is referenced extensively in the OSHA manual.

The robot selection process starts with a thorough understanding of application. Once the process requirements are fully understood then robot arm geometry, end effector requirements, robot arm performance capabilities, controller capabilities, and the choice of whether to use a new or used robot can be evaluated.

6.8 Key Words

continuous path control
degrees of freedom
digital input/output (I/O) interface boards
dual gripper
electrical power systems
encoder
end effector
end effector (EE) cable
end-of-arm tooling
executive processor
gripper
hydraulic power systems
limited sequence control

linear joint
link
magnetic grippers
mechanical arm
mechanically actuated grippers
orthogonal joint
payload capacity
peripheral equipment control
power source
pneumatic power systems
reduction drive
revolving joint
risk assessment
robot controller
rotational joint
simple mechanical device grippers
tachometer
teach pendant
twisting joint
vacuum grippers
work envelope

6.9 Review Questions

1. Discuss the origins of the term *robot*.
2. What is the definition of robot?
3. List and discuss the basic hardware of any modern robot system.
4. Explain the difference between a robot link and a robot joint.
5. List and describe the five types of robot joints.
6. Define a robot's work envelope.
7. What are the five basic robot configurations? Define the work envelope for each.
8. What are the major differences between a parallel robot configuration and the five basic robot configurations?
9. List and describe four types of robot grippers.
10. List and describe the three basic types of robot power sources. What are the advantages and disadvantages of each?
11. Explain the role of a reduction drive in an electrically actuated robot arm.
12. What are the three major functions of the robot controller? Describe the role each plays in an industrial robot.
13. Explain the difference between a limited sequence controller and a continuous path controller. Which type would be required in a spray painting application?

14. What role does the executive controller play in regard to controlling arm motion?
15. Define peripheral equipment control.
16. What device enables the creation, editing, and manipulation of the files of a robot program?
17. What are the three main robot application areas? Give an example of each.
18. List and discuss the two robot safety standards reviewed in the chapter.
19. What are the three types of injuries that workers should be safeguarded against?
20. List and discuss the five major considerations for selecting a robot for a particular application.

6.10 Bibliography

1. Groover, M.P.2001, *Automation, Production Systems and Computer-Integrated Manufacturing,* 2nd edition, Prentice Hall, Upper Saddle River, New Jersey.
2. Rehg, J.A.2003, *Introduction to Robotics in CIM Systems,* 5th edition Prentice-Hall, Upper Saddle River, New Jersey.
3. Colesock, H.2005 *Industrial Robotics*, McGraw Hill, New York.
4. Chang, T.C., Wysk, R.A., and Wang, H.P. 2005 *Computer-Aided Manufacturing*, 3rd edition, Prentice-Hall, Upper Saddle River, New Jersey.
5. http://www.powertransmission.com/issues/0706/harmonic.htm
6. www.OSHA.gov, *OSHA Technical Manual*, Section IV: Chapter 4 Industrial Robots and Robot System Safety.

Chapter 7

Robot Programming

Contents

Objective

The objective of this chapter is to present a methodology to create, organize, program, and implement a robot program for industrial application. Additionally, various programming methods, languages, and instructions will be discussed, and simulation of the robot program will be addressed.

7.1 Robot Programming Concepts

Figure 7-0 shows a typical material handling application in which a robot is tasked to move a part from the conveyor on the left to a processing station on the right.

Figure 7-0 Robotic material handling application

In order to execute this task the robot needs to know many things, some of which are:
- Point at which to pick up a part from the conveyor.
- Path the gripper should follow to get to the part.
- Points at which the gripper should open and close.
- Path the gripper should follow to get to the processing station.

The list is far from complete; the robot needs to answer numerous other questions if it is to adequately support the work cycle. However, the list shows that the major issues center around choice of path the arm should follow and when certain events should occur. It is the job of the robot program to address these issues. A *robot program* is defined as a set of program language instructions (or commands). These commands specify the path of the end effector (e.g., gripper, end of arm tooling), make logic decisions, and execute peripheral actions necessary to support a work cycle. In *robot programming* a robot program is entered and stored in the robot controller's memory.

The path that the robot arm needs to follow is specified in a robot program by a combination of stored *robot arm positions* and program *motion instructions*. Stored robot arm positions are named locations in space that the end effector must reach or pass near in order to execute a work cycle. Motion instructions dictate the stored position to which the

arm must move, as well as how it is to move. An arm may move the end effector in a straight line to the stored position or move each arm joint simultaneously to reach the position as quickly as possible. The arm may pause at the position or just move through it. It may have to achieve the position exactly or just pass near it. All information is conveyed to the robot controller by the motion instructions.

Again referring to Figure 7-0, shown are examples of *peripheral actions*—the opening and closing of the gripper and communication with the conveyor and processing station. These activities are specified by the robot language *input and output (I/O) instructions*, which examine the status of inputs to—and turn on outputs from—the robot controller. The peripheral equipment, in turn, reads these outputs and takes appropriate action.

Coordination of the execution of the motion instructions and execution peripheral action is accomplished with *logic instructions*, which make decisions within the robot program.

There is no truly universal robot programming language. Languages and robot programming methods are dependent on the robot supplier. However, each language has, at a minimum, logic instructions, methods of storing and recalling arm positions, motion instructions, and I/O instructions.

7.2 Programming Methods

The choice of robot programming method is largely based on the type of robot being processed. Simple robots, such as the pneumatic playback robot, will be set up with mechanical stops and limit switches. Work cycle steps are executed through some type of sequencing controller, such as a programmable logic controller. Servo robots, on the other hand, will have some method of storing robot arm positions, which are then used in conjunction with a higher level programming language. Regardless of robot type, two tasks must be accomplished: (1) Arm motion must be programmed; (2) Sequencing of work cycle program must be programmed. Because these two tasks are accomplished in various ways, different programming methods are needed.

Most modern industrial robots are servo driven with rather sophisticated controllers and, accordingly, excellent performance capabilities. Thus, our programming focus will be limited to these types of robots. The two programming methods available for robots of this type are *motion programming* and *robot language programming*.

The first of these, motion programming, moves the robot arm through a work cycle; appropriate instructions are entered at each step of the process. This is called *teaching the robot* the work cycle program. In this type of programming the storing of robot arm positions occurs simultaneously with the entering of *program instructions*. (Important: "Program instructions" are not the same as "program of instructions". The former is a list of instructions for the process; the latter are instructions in the actual robot program.) The general procedure for programming a robot using motion programming is as follows:

1. Enter an instruction to move the end effector to the start location or home position of the work cycle program. Store this location to the robot controller.
2. Enter the appropriate logic and/or I/O instructions to control the end effector and/or peripheral equipment as required by the application.
3. Move the end effector to the next position along the robot's path in the correct orientation. Enter the appropriate motion command to reach this location. This stores the location and sets the necessary parameters for the arm to reach the location during execution of the work cycle program.
4. Enter necessary logic and/or I/O instructions to control the end effector and/or peripheral equipment for this location.
5. Repeat steps 3 and 4 until the work cycle is fully programmed.

In this method the robot is dedicated to the *programming* task. *It is not performing any useful work during the programming process.* Because it takes time to fully program a robot, the time out of service can be significant. That is why some manufacturers go to the extent of purchasing two identical robots for a given application. One robot will be dedicated to doing the work and the other to developing new robot programs.

The second method, robot language programming, reduces some of the time the robot is actually out of service for programming. It separates the programming of motion, logic, and I/O instructions from teaching of robot arm positions and orientations. The only time the robot is out of service is when the robot arm positions are taught, named, and stored to the robot controller. The actual program is written on a separate computer terminal away from the robot. Once developed, it is stored for later uploading to the robot controller for verification. This is advantageous because the robot can be performing other production tasks while a new program is written. The only production time lost is that needed for programming arm path positions. Additionally, computer-based graphic simulation programs are often used to further minimize robot downtime, which they do by verifying the program through simulation before uploading it to the robot controller. This is discussed in more detail in subsequent sections.

What both methods have in common is that the robot must be taught the required arm positions to execute the work cycle program. This is discussed next.

7.2.1 Teaching Arm Positions

Teaching the robot the necessary arm positions to execute the work cycle program involves moving the end effector to the desired position and orienting it correctly. This phase of the programming is called the *teaching positions* step. Once correctly positioned each joint position is stored as a named location to the robot controller and/or a motion command is entered. In the past this was accomplished in a number of ways, including manually moving the arm into position, an action called *manual leadthrough*. In this the programmer would actually grasp the robot arm and physically move it into the desired position. If the robot arm were particularly large and heavy, a separate programming

device having the same arm configuration as the robot would be used. Now, modern servo robots use a technique called *powered leadthrough*. With this technique the robot is power-driven to the desired position with the use of a *teach pendant*. A teach pendant for a Fanuc ArcMate robot is shown in Figure 7-1.

Figure 7-1 Typical teach pendant

Teach pendants have buttons and switches that an operator pushes to move robot joints to position the end effector in the correct location. This is called *jogging* the robot arm into position. (The term "jogging" is only used during programming.) During programming, robot positions are being taught. This is the time when, using the teach pendant, the robot is jogged into the desired position. When the program is executed there is no jogging; the robot just moves to the position. For robots that utilize motion programming, the teach pendant also has the capability to enter all necessary programming instructions. The teach pendant is the primary user interface for the robot.

In order to more efficiently move the arm into position, robots have multiple *jogging coordinate systems*. These coordinate systems force the robot to move in a specified manner. They are: *joint coordinate system*, *world coordinate system*, and *tool coordinate system*.

The joint coordinate system allows each joint to be moved individually, one at a time. This is the freest form of jogging the robot because it gives the programmer maximum control of the robot arm. The other jogging coordinate systems restrict the robot arm's jog motion. Thus, this is the coordinate system of choice for quick and easy jogging

of the end effector to any position within the work envelope. Typically, it is used to move the arm over a large distance to approach the position of interest and to effectively orient the end effector. Once near the position and oriented correctly, the other jogging coordinate systems are used to more accurately jog the end effector into the final programmed position. Prior to discussing these other jogging coordinate systems we need to understand *tool centerpoint*.

All robot arms have a mounting plate on the robot's wrist for end effector mounting. This is the robot's *tool plate*, or *faceplate*. In its center is the *tool centerpoint*, which is the point of action for the end effector, as its position and orientation correspond with the end effector's position and orientation. The robot controller accurately controls the position and orientation of the tool centerpoint. The mounting configuration, by extension, controls the position and orientation of the end effector. Figure 7-2 shows a tooling plate and corresponding tool centerpoint for a typical robot arm.

Figure 7-2 Tooling plate and tool centerpoint

The other two jogging coordinate systems, the world coordinate system and the tool coordinate system, force the robot arm's joints to move in a synchronized manner, enabling the end effector to move relative to a specific coordinate system. Often these coordinates systems are called *reference frames*. These two systems are shown in Figure 7-3.

Figure 7-3 World coordinate system and tool coordinate system

World coordinate system jogging enables the end effector to move parallel to one of the coordinate axes (*x*, *y*, or *z*) of a world coordinate system located at the robot's base. As the end effector moves, the orientation of the tool centerpoint remains constant, enabling the end effector to maintain its orientation to the work while it is positioned.

The tool coordinate system enables the robot arm to move relative to a coordinate system located at the tool centerpoint. This is useful because it is often necessary to move the end effector away from the workpiece along the end effector's centerline. The tool coordinate system also maintains orientation of the tool centerpoint while it is in motion.

In a typical application, each of the coordinate systems may be used at different times. Consider Figure 7-4.

Figure 7-4 Teaching position example

Figure 7-4(a) shows an application in which a pyramid-shaped workpiece is sitting on a worktable in front of a 6-axis robot. Assume here that the robot is tasked with performing work on an inclined face of the workpiece. Initially the robot is jogged close to the desired work position using the joint coordinate system. In Figure 7-4(b) we see each joint moved individually to roughly position the end effector near the workpiece. The joint coordinate system continues to be used to jog the appropriate joint to correctly orient the end effector. In Figure 7-4(c) the world coordinate system is seen, used to jog the end effector into proper position. Finally, the tool coordinate system jogs the end effector to the final work position as shown in Figure 7-4 (d). Having achieved the final position with the proper end effector orientation the position can be stored as a named location or a motion instruction entered. Both of these actions are performed with the teach pendant.

7.2.2 Taught Positions and User Coordinate Systems

The *user coordinate system* is an important part of the step called "teaching robot positions". It is established by the user to aid in teaching positions and in programming particular robot applications. Consider Figure 7-5. This figure shows an application where a robot is tasked to engrave the word "ROBOT" on an inclined workpiece. Because the orientation of the workpiece is not parallel to the robot's world coordinate system, it would be beneficial to create a custom or user coordinate system coincidental with the workpiece. This is a common situation, so most robot suppliers provide a means for creating custom user coordinate systems.

Figure 7-5 User coordinate systems

For example, Fanuc robots call their user coordinate system a *user frame*. All positions taught and stored to the controller are stored in reference to this user frame. By default, the user frame is set to the robot's world coordinate system. However, the programmer can set the user frame to any orientation desired by following a simple procedure. Once the user frame is established the robot can be jogged parallel to it and the position stored accordingly. This often significantly reduces the effort required in teaching positions. Additionally, the user frame can be manipulated within the program code during program execution. Although manipulating user frames offers great programming flexibility, it is robot specific and goes beyond the scope of this text. Individual robot programming manuals should be consulted for additional information.

Now that we have addressed robot programming methods, including teaching arm positions, and user coordinate systems, we look, in depth, at robot programming languages.

7.3 Robot Programming Languages

Robot programming languages communicate to a robot controller which actions it needs to perform in order to execute a work cycle. The language chosen in each case instructs the manipulation of the robot arm, handles sensory data, provides intelligence in the form of logic decision-making, and processes data. There is no standard robot programming language (as there is in the other programmable automation technologies discussed in this text, namely CNC and PLC). Rather, most modern languages are typically higher-level, structured languages designed to simplify the amount of data the programmer must input. Higher-level languages have instructions that resemble standard English. These languages are termed "structured" because the resulting programs have a hierarchical flow structure with conditional branching and nested loops. In fact, some

robot programming language developers worked from an existing higher-level structured programming language, like BASIC or PASCAL, and added the robot arm control instructions. So, familiarity with languages of this sort allows one to transition to developing complex robot programs with relative ease.

As we have noted, languages are solely dependent on robot brand. Therefore, we must select a particular brand of robot in order to demonstrate the development of a robot program. We will use the Fanuc robot brand. Note, however, that its programming concepts are readily transferable to other robot programming languages. The author views Fanuc Robots as one of the robotics industry leaders. Fanuc has a large portion of the industrial robot market and a strong presence in the United States. One may commonly observe a bright yellow, hard-working Fanuc robot in action in a domestic manufacturing facility!

Fanuc Robots utilize a proprietary programming language called KAREL, which was developed from PASCAL. KAREL was named for the Czech writer Karel Capek, who supposedly derived the word "robot" from the Czech noun *robota* for "labor" or "serf labor"; see Chapter 1 and 6). (In fact, Capek himself credited his brother Josef for first coining the word).

A portion of a robot program written in KAREL is shown in Figure 7-6. Text on the right that is preceded by two hyphens (- -) are comments. Other comments appear sporadically throughout a robot program. All programs should be heavily commented so that future reviewers of the program may quickly understand the program. One can see that the instructions are given in plain English. Even a quick glance of the program allows a reader to quickly translate many of the instructions.

Figure 7-6 Karel program

This particular program was written for an A510 Fanuc Robot utilizing an RH controller in a material handling application. Robot controller type is important to

programming languages and techniques. The RH controller is a 1990s era robot controller. Programming was performed through a computer terminal built into the control cabinet or through a PC emulating a VT220 terminal connected to the controller with an RS-232 cable. A teach pendant manually controlled the robot and stored named arm positions used by the program. The robot was completely offline during the program process. With the purchase of additional software, one could develop the program offline and upload it to the robot when needed. There are still many RH controllers in use and available on the used robot market.

The newer generations of Fanuc robot controllers use, primarily, *teach pendant programming* (TPP) methods. TPP is motion programming that uses a specialized language to simplify robot programming for specific applications such as material handling, arc welding, and painting, to name a few. TPP methods are essentially built on top of KAREL to enable an end user lacking much expertise in robot language programming to quickly and easily develop robot programs. TPP is available only with the latest Fanuc controllers such as the RJ and R-30 series controllers. Note, however, that KAREL is still widely available and often used because it is very versatile, able to handle virtually any type of robot application. Figure 7-7 shows a program in the TPP language for a Fanuc ArcMate arc welding robot with an RJ controller. This particular program does not contain any arc welding instructions. It is just a simple routine to move the end effector out of the way so that the welding fixture can be reloaded with a new part. The truncation of the instructions as compared to the KAREL program of Figure 7-6 is to simplify programming through the teach pendant.

```
LBL[1:Main Logic Section]
RO[3] = OFF
J PR[1] 100   FINE
RO[4] = ON
IF RO[1] = ON JMP LBL[2]
JMP LBL[1]
LBL[2: Unloading Routine]
RO[4] = OFF
RO[2] = OFF
J P[1] 100   FINE
L P[2]  100  FINE
RO[3] = ON
WAIT  3
L P[2]   60  FINE
RO[1] = ON
J P[3] 100   FINE
WAIT RI[2] = ON forever
L P[4]  30  FINE
RO[1] = OFF
RO[3] = OFF
WAIT  3
L P[3]   100  FINE
RO[2] = ON
IF RO[1] = ON JMP LBL[2]
JMP LBL[1]
```

Figure 7-7 ArcTool program example

Since teach pendant programming languages are directed at specific applications, the KAREL language will be used exclusively in this text, in an effort to provide a generalized approach to robot programming. Note, however, that the methods and techniques presented with KAREL are readily adaptable to Fanuc TPP and, for that matter, to any high-level robot programming language.

7.4 Robot Program Development, Organization, and Structure

In this section we discuss developing a robot program for a particular application. The applications will be simple but will highlight the most important aspects of robot programming, including steps involved in developing the program, program organization, typical motion instructions, and logic and communication instructions. This approach should equip the reader with the knowledge necessary to attack much more difficult robot applications.

There are four major program development stages:
1. Writing the program of instructions
2. Robot programming
3. Program testing
4. Release to production.

The key aspect of stage 1, writing the program of instructions, is program planning; a well-planned program dramatically shortens the time it takes to develop a program, specifically in regards to program testing and modification. This saves precious production time. Conversely, a poorly planned program may require much iteration before it is successful, accruing substantial development costs due to lost production time and extra wages. This stage yields a program of instructions written in the appropriate robot language. Additionally, during this stage a simulation of the program should be performed. Program simulation will be discussed in more detail in a subsequent section.

Stage two is physically entering the program into the robot controller. As already discussed, programming is accomplished through either motion programming or robot language programming. In either case, arm positions have to be taught and stored (Section 7.2.1) and the program directly entered into the controller through the teach pendant or uploaded from a separate computer. Once the program is entered and stored in the controller it can be tested and modified as necessary.

In stage 3 the program is tested initially by a manual stepping through of the program line by line at very low speed. (All robots have the capability to verify a program by executing it line by line or step by step, in "slow motion.") This enables the programmer to verify that the program was entered correctly and that it functions as intended. Next, the program is run continuously at low speed. If it passes this test, additional runs will be performed in which the speed is gradually increased until the program is running at production speeds.

The decision to release the program to production (stage 4) depends on company procedures, but it generally involves short production runs intended to verify program integrity and functionality. The time needed to complete stages 3 and 4 is largely dependent on how well the program was planned, which we now turn to.

7.4.1 Writing the Program of Instructions

Writing a program of instructions entails four simple steps:
1. Create a sketch or drawing of the application
2. Develop the process flow of the application
3. Translate the process flow into the robots programming language
4. Simulate the program.

When sketching or drawing the application, be sure to include the relative positions, sizes, and locations of the application's robot(s), processing equipment, tooling, and end effectors. Note that multiple sketches may be required, each depicting the process at various stages of the work cycle. An example application sketch is shown in Figure 7-8.

Figure 7-8 Example application sketch

The figure shows a top view of an application in which a 4-axis Fanuc A-510 robot unloads two different sizes of wood disks from a mold, places them on a staging plate, picks them up again, machines the inside and outside diameters, and, finally, releases the parts in a specific location for other processing. Note that the sketch represents the relative location of the application equipment, namely, the press, staging plate, mold, robot, and machining stations (router). Also, observe how each of the robot arm positions are identified, including the mold locations (1A, 2A, 3A,…), the staging plate locations (1B, 2B, 3B,…), routing positions, home position, repair position, and a drop zone position. From this sketch we proceed to the next step of program planning: developing the program's process flow.

Developing the program's process flow may involve making a rough flow chart of the robot's role in the work cycle. From this rough flow chart one can further break down or isolate the robot's action into individual motion routines. This is demonstrated in Figure 7-9 for the application shown in Figure 7-8.

Figure 7-9 Rough process flow chart

From this rough process flow the logic and motion sections of the program can be identified. The next step is to actually translate the process flow into instructions the robot can understand. This involves writing the program in the robot's programming language. To aid in this process a programming sheet can be used. Figure 7-10 shows a completed program sheet for a Fanuc ArcMate teach pendant program.

Program Sheet

| Program Name: Demo Program | | Programmer: *Dan Kandray* | | | | | | | | Sheet No.: | | | Date: |

Process Flow/Work Cycle Description	Program Step	Motion Type	Position Number	Position Descritpion	Speed %	Speed mm/s	Speed in/min	Termination Type Fine	Termination Type CNT	Gripper Open/Close	Register Instr	Input/Output Instr	Position Register	Branching Instr.
Label Motion Routine	1												LBL[1]	
Open Gripper (RO[2])	2											RO[2] = off		
Turn Off part moved output (RO[1])	3											RO[1] = off		
Move Joint to perch position (PR[1])	4	J	PR[1]	Perch	100			FINE					J PR[1] 100 FINE	
Wait for Input 1 to start routine (RI[1])	5												WAIT RI[1] = ON	
Joint move to P[2]	6	J	P[2]	ABOVE LOAD	100			FINE					J P[2] 100 FINE	
Linear move to P[1]	7	L	P[1]	LOADING PT		30		FINE					L P[1] 30 FINE	
Close gripper (RO[2] on)	8											RO[2] = ON		
Wait for gripper to settle	9												WAIT 3	
Linear move to P[2]	10	L	P[2]	ABOVE LOAD		30		FINE					L P[2] 30 FINE	
Joint move to P[3] continuous	11	J	P[3]	CLEAR OBSTACLE	100				CNT 50				J P[3] 100 CNT 50	
Turn ON part moved output (RO[1])	12											RO[1] = ON		
Joint move to P[4]	13	J	P[4]	ABOVE UNLOAD	100			FINE					J P[4] 100 FINE	
Linear move to P[5]	14	L	P[5]	UNLOAD PT		30		FINE					L P[5] 30 FINE	
gripper open	15											RO[2] = OFF		
Wait	16												WAIT 3	
Linear move to P[4]	17	L	P[4]	ABOVE UNLOAD		30		FINE					L P[4] 30 FINE	
Move to Perch	18	J	PR[1]	PERCH POSITION	100			FINE					J PR[1] 100 FINE	
Jump to start of program	19												JMP LBL[1]	

Figure 7-10 Completed program sheeti

The robot program should be organized as shown in Figure 7-11. In the *main logic section* the robot communicates with the outside world. This section could also be called the *setup section* because it sets up the robot program. In this section the robot's outputs will be set to inform peripheral equipment and the work cell controller (typically a PLC) of the robot's status. Additionally, in this section of the program the robot will check the status of its inputs and determine the appropriate action. This decision-making ability is also called *program logic*. This section of the program is essentially the jumping off point for execution of robot motion routines.

Figure 7-11 Example program organization

In any application the robot is required to repeatedly perform a series of tasks. Each task typically involves moving the robot arm. Arm motion instructions associated with a given task are located in the *motion routines* sections of the program. These organize and group arm motion instructions that are routinely repeated. Typically there are many arm motion routines for each program. For the application of Figures 7-8 and 7-9, there would be several motion routines for the first block of the flow chart alone. One motion routine picks up the part at location 1A and moves it to position 1B. Another routine picks up the part at location 2A and moves it to position 2B, and so on. There would be even more routines for moving parts from the staging plate to the router and then to the drop zone position. Organizing the program with motion routines simplifies writing, makes the program easier to follow, and aids in debugging.

Program simulation, the last step of writing the program of instructions, is execution on a graphical computer program that simulates the robot arm motions. Program simulation can be used to verify program integrity, evaluate robot arm trajectories, and check for potential interferences. Additionally, simulation can aid in developing motion routines that identify the minimum number required stored positions. Program simulation is discussed further in Section 7.6.

7.4.1 Arm Motion and Motion Instructions

Arm motion is defined as physical arm movement from the time the robot arm starts moving until the time it stops. As the arm moves, it forces the tool centerpoint to follow a specific path that is called the *trajectory*. During arm motion the robot arm accelerates to a specified speed, maintains that speed along the trajectory, then decelerates to stop at the desired destination. To program arm motion one must specify:

- trajectory
- acceleration/deceleration
- speed
- termination.

In KAREL, this information is communicated to the robot with a combination of system variable settings, motion instructions and taught positions. The number of motion instructions and taught positions needed to program arm motion depends on the complexity of the motion. Consider Figure 7-12.

Figure 7-12 Robot arm motion

The figure shows a 6-axis robot in three different positions. Assume each position represents a taught position. We will consider several different scenarios of arm motion related to points A, B, and C. Initially, assume arm motion should start at position A and finish at position B (Figure 7-13).

Figure 7-13 Robot arm motion from A to B

This motion consists of one *motion interval*. A motion interval is motion generated by a single motion instruction. Position A is the start point of the motion interval and position B is the termination point. *Termination* means that the motion instruction used to generate the motion interval is complete and the robot motion controller can move to the next instruction.

In KAREL, as with most languages, there are numerous program motion instructions, depending on the type of motion desired. For this example, a suitable motion instruction would be the MOVE TO instruction. This instruction initiates motion of the tool centerpoint to a specified position. In the program the instruction it would appear as follows:

MOVE TO B

Note that this instruction only causes the execution of the motion interval. The details of the motion's termination, trajectory, speed, and acceleration/deceleration have yet to be specified. First consider the motion interval's termination. Most programming languages allow for multiple types of termination: It could be specified such that the arm stops precisely at the taught position, close to the taught position, or merely passes through the taught position. Here, the motion interval is to terminate precisely at position B.

In the KAREL language, the termination type is specified by the system variable $TERMTYPE. The possible values for $TERMTYPE are:

1. FINE
2. COARSE

       3. NOSETTLE
       4. NODECEL
       5. VARDECEL

FINE termination causes the robot to move to the taught position and stop before beginning the next motion interval. Likewise, COARSE termination causes the robot to move to a position and stop. The difference between the two is the tolerance applied on each of the axis encoders that determine if a desired position has been reached. For FINE the tolerance is small, causing the controller to achieve the taught position precisely. COARSE termination has a larger tolerance, allowing the arm to just move close to the taught position. NOSETTLE, NODECEL, and VARDECEL do not apply to this example (they are defined in the next section).

In order to stop arm motion precisely at position B, the FINE termination type is required. Thus, the program instructions would appear as follows:

    $TERMTYPE = FINE
    MOVE TO B

The type of *interpolation* method specified by the motion instruction will determine the trajectory of the motion. The three standard interpolation methods are:

- joint interpolation
- linear interpolation
- circular interpolation

Joint interpolation requires each axis (or joint) start and stop at the same time. The resulting trajectory of motion depends on the relative speed of each axis. Using the arm speed specified by the programmer (the *programmed speed*), the robot's motion controller calculates the time each axis needs to move from a current position to its final position. The longest time becomes the *interval time*. Accordingly, the other axes will move at lesser speeds so that each axis stops moving at the exact same moment. This means the trajectory is unpredictable—not a simple geometric shape. However, axes follow the same trajectory each time the motion interval is executed. Joint interpolation is demonstrated in Figure 7-14. The perspective is from above, looking down on the robot.

Figure 7-14 Joint interpolation

Linear interpolation (Figure 7-15) dictates that tool centerpoint move in a straight line from the initial to final position at the specified program speed. Additionally, the orientation of the tool is changed continuously.

Figure 7-15 Linear interpolation

Circular interpolation forces the tool centerpoint to follow a circular arc trajectory from starting to final position. However, this type of interpolation generally requires an additional intermediate taught position. This is demonstrated in Figure 7-16. The arm does

not stop at the intermediate position but passes through it. So, to define circular motion, three taught positions are required.

Figure 7-16 Circular interpolation

If "motion segment" is defined as arm motion between two taught positions, then a circular interpolation motion interval consists of two motion segments. Figure 7-17 shows this graphically.

Figure 7-17 Circular interpolation with motion segments identified

In KAREL, interpolation type is specified by the $MOTYPE (motion type) system variable. Accordingly, the three possible values of $MOTYPE are JOINT, LINEAR, and CIRCULAR. For circular interpolation the intermediate position used to define the circular arc must be specified in the program instructions. This is accomplished by adding the VIA clause to the MOVE TO instruction. Assume for the motion interval shown in Figure 7-11 that circular interpolation is desired. An intermediate position as shown in Figure 7-15 is taught and stored as position I. The program instructions would then be written as follows:

$MOTYPE = CIRCULAR
$TERMTYPE = FINE
MOVE TO B VIA I

During arm motion the robot accelerates the tool centerpoint up to the program speed, maintains that speed, and then decelerates to stop motion at the termination position. This is shown as a graph of the velocity profiles shown in Figure 7-18. A *velocity profile* is a plot of the tool centerpoint velocity versus time. Two profiles are shown in the figure. Note that the average translational speed (or velocity) is less than the program speed because of arm acceleration and deceleration.

The Fanuc robot motion controller holds constant the time the arm is allowed to accelerate or decelerate. As the programmed speed is increased the average acceleration will increase. This makes the average acceleration value proportional to the programmed speed. As the programmed speed increases so will the average acceleration/deceleration. Again, this is demonstrated by the two velocity profiles. So, through arm speed, the acceleration and deceleration are automatically determined.

Figure 7-18 Arm motion velocity profile

KAREL uses the system variable $SPEED to control the translational speed of all programmed motions. This value is expressed in mm/sec. Its maximum and minimum values are dependent on the particular robot model used. If it is desirable to run the robot at maximum speed during linear or circular interpolation then the $SPEED system variable is set as follows:

$SPEED = $SPEEDLIM

If this setting is specified for joint interpolation, the controller will convert this value to a fraction of the maximum joint speed: If maximum speed is desired with joint interpolation, then it is necessary to set the $SPEED variable to $SPEEDLIMJNT.

For the example shown in Figure 7-17 maximum speed was assumed to be the desired speed. Since circular interpolation is specified, the program instructions would be:

$SPEED = $SPEEDLIM
$MOTYPE = CIRCULAR
$TERMTYPE = FINE
MOVE TO B VIA I

The instructions for this motion interval example are now complete. The arm will move with precision to position B in a circular arc defined by intermediate position I at maximum translational speed. Since the arm stops at position B, the velocity profiles would be similar to one of the plots of Figure 7-16. The motion interval, defined by the single motion instruction MOVE TO B VIA C, terminates also. So, for this example both arm motion and motion interval finish at the same time. This is not always the case: sometimes the motion interval terminates while the robot arm continues to move. This often occurs because arm motion is complicated, requiring numerous taught positions and many motion segments. Additionally, it is usually not necessary or efficient to decelerate the arm to a stop at each taught position along the trajectory, but rather to let the robot arm continue to move as it passes near or through the taught positions. This scenario is called *multiple segment motion*.

7.4.2 Multiple Segment Motion and Program Motion Routines

Consider Figure 7-19, a simple robotic material handling application. A robot is tasked with moving a dowel pin from position A to position C, which represent taught positions of the robot's arm motion. The robot arm comes from some safe position, moves above position A, then to A, grasps the pin with a gripper, lifts it from the hole, moves over the obstacle to above C, slides the pin into the hole, opens the gripper to release the dowel, and moves out of the way back to the safe position. This task can be programmed in, essentially, three arm motions:

1. Moving the gripper into position to grasp the dowel pin.

2. Extracting the pin from the hole, moving it over the obstacle, inserting it into the
   other hole.
3. Moving the arm out of the way back to a safe position.

Because these three arm motions address a specific task and are likely to be repeated
often within the context of a larger program, the program instructions for these three
individual arm motions will be organized as a program motion routine (see end of this
section).

Figure 7-19 Multiple motion segment example

The second arm motion is the most complex because it involves actually performing
the work of moving the pin. The arm motion should be as smooth as possible from A to C.
Figure 7-20 shows the proposed path or trajectory of the dowel pin, along with additional
taught positions B, D, and E necessary to complete the arm motion.

Figure 7-20 Multiple motion segment pin positions

The movement of the pin from A to C represents one arm motion because the arm starts moving at A and does not stop until it gets to C. There are four motion segments: A to D, D to B, B to E, and E to C, as shown graphically in Figure 7-21. Motion segments 1 and 4 require linearly interpolated trajectories in order to pull the pin straight out of the hole and push it straight into the second hole. Motion segments 2 and 3 can utilize joint interpolation because the path does not require a specific geometric shape.

Figure 7-21 Motion segments and required interpolations

With multiple segment motion it is often not necessary for the robot arm either to stop at or to pass *through* each taught position; it may only need to pass *near* the position. KAREL utilizes termination type to control the arm's proximity to the taught positions. Consider Figure 7-22, which shows multiple segment motion from position A to B, then B to C. The lines drawn between the positions represent arm trajectory for different termination type. The KAREL instructions for this motion, without termination type specified, is written as follows:

```
$TERMTYPE = ?????
MOVE TO B
MOVE TO C
```

The motion controller processes motion instructions (or, more descriptively, *motion intervals*) sequentially. For the code above, it will not process the move to C until it completes *or terminates* the instruction to move to B. With FINE termination, the motion instruction is not terminated until the arm is essentially at the taught position. So, the arm will decelerate fully and slow to a stop as it reaches taught position B and then will immediately accelerate again as the move to C is processed. To the casual observer the

motion would appear somewhat jerky. COARSE termination yields a similar result without settling exactly on taught position B. NOSETTLE, NODECEL, and VARDEL termination types provide smoother motion segments.

NOSETTLE and NODECEL terminations enable the arm to keep moving without stopping at the taught position. With NOSETTLE the arm decelerates, but does not stop before starting the next motion interval. With NODECEL, on the other hand, the arm does not even decelerate as it passes near the taught position. VARDECEL allows the user to set the amount of rounding with another system variable called $DECELTOL. A value of 1 for $DECELTOL will yield rounding equivalent to NODECEL and a value of 99 yields rounding equivalent to NOSETTLE.

Figure 7-22 Effect of termination type on trajectory

For the motion shown in Figure 7-21, the termination for positions D, B, E, and C need to be specified. Consider motion segment 1. Since the pin should be pulled straight up with linear interpolation, the arm must achieve position D precisely. Consequently, FINE termination will be used for taught position D. For the next motion segment it is not necessary to pass through position B. However, it is important that the obstacle be cleared. Assuming that just passing near position B in Figure 7-21 would allow the pin to clear the obstacle, a NOSETTLE termination command would suffice. The next position, E, requires FINE termination to ensure the pin be located directly above the second hole. Finally, position C will require FINE termination as well. Since FINE termination will appear most often as the termination type, the $TERMTYPE system variable will be used to specify this type of termination as the default termination type. Following this same

logic, the default motion type will be specified as JOINT with the $MOTYPE system variable. The NOSETTLE termination type and the LINEAR motion type will then be specified using the WITH clause when needed. The WITH clause is used in a move statement to specify temporary values for system variables. Accordingly the KAREL instruction for this motion would appear as follows:

```
$SPEED = $SPEEDLIM
$MOTYPE = JOINT
$TERMTYPE = FINE
WITH $MOTYPE = LINEAR MOVE TO D
WITH $TERMTYPE = NOSETTLE MOVE TO B
MOVE TO E
WITH $MOTYPE = LINEAR MOVE TO C
```

The resulting motion trajectory is shown in Figure 7-23.

Figure 7-23 Pin trajectory

Note how the NOSETTLE termination setting for position B causes the trajectory to only move near to B. This completes one arm motion of the motion routine.

The other two arm motions necessary to complete this motion routine include moving the gripper into position to grasp the pin and moving the gripper out of the way after the pin has been moved to position C. These two motions are shown together in Figure 7-22. Whenever it approaches or leaves a desired position the end effector must not collide with

any obstacles, including the part to be manipulated. For this reason, the gripper in Figure 7-24 approaches the pin from above to ensure the gripper fingers do not collide with the pin. This can be programmed by taught positions, such as position D, or by a different type of motion instruction. The latter will be used to program the approach trajectory and departure trajectory.

Figure 7-24 Gripper approach and departure

Consider Figure 7-25. The figure shows the gripper above the pin (Figure 7-25(a)) and then just prior to grasping the pin (Figure 7-25(b)). Assume it is known, as shown in the figure, that the gripper must be approximately 75 mm above the pin to avoid striking it as it approaches.

Figure 7-25 Gripper approach

Because this scenario is so common in robot programming, many languages provide a specific instruction to minimize the need to teach additional positions. These can be termed *approach instructions* because they deal with the approach to a desired position. In KAREL the approach instruction is MOVE NEAR, which causes the tool centerpoint to approach a desired position by an offset distance. The offset distance is along the negative z-axis of the stored position. If the distance is specified as a positive value, the offset will be in the negative z-direction. A negative distance, conversely, will cause the offset to occur in the positive z-direction. Thus, for the example in Figure 7-25(a), the approach instruction would appear as follows:

MOVE NEAR A BY -75

It is very important one know the orientation of the coordinate system used to store the designated position. A wrong sign for the direction may result in the gripper crashing into the part. Recall from Section 7.2.2 that Fanuc stores positions relative to the user frame (UFRAME) coordinate system. So, the programmer has to know the correct orientation of the *z*-axis of the destination position when using the MOVE NEAR instruction. The Fanuc documentation gives additional information on user frames.

The instructions that allow the arm to approach and grasp the pin are shown below:

```
$SPEED = $SPEEDLIM
$MOTYPE = JOINT
$TERMTYPE = FINE
MOVE NEAR A BY -75
DELAY 500
WITH $MOTYPE = LINEAR MOVE TO A
DELAY 500
CLOSE HAND 1
DELAY 500
```

Observe the addition of some new commands. The DELAY command causes the program to pause for a specified number of milliseconds. In this example, the program delays for 500 milliseconds. When high accuracy is required, a slight delay between arm motions and gripper activation is usually desired. Accordingly, the CLOSE HAND 1 instruction is sandwiched between two DELAY statements. The CLOSE HAND 1 instruction causes the gripper to close and grasp the pin. The numeral 1 is needed in the instruction because robots can have more than one gripper attached at the same time.

The last arm motion to program is that for moving the gripper back to the safe position once the pin is moved to position C. The trajectory for this motion is shown in Figure 7-26. Figure 7-26(a) shows the gripper and pin position immediately following placement of the pin in position C. Figure 7-26(b) shows the arm trajectory after the pin has been released. Position B pulls the gripper in that direction.

(a)

(b)

Figure 7-26 Gripper departure trajectory

Using the same concept as in the approach trajectory, only in reverse order, the arm motion instructions are as follows:

    $SPEED = $SPEEDLIM
    $MOTYPE = JOINT
    $TERMTYPE = FINE

```
OPEN HAND 1
DELAY 500
WITH $MOTYPE = LINEAR MOVE NEAR C BY -75
DELAY 500
WITH $TERMTYPE = NOSETTLE MOVE TO B
MOVE TO SF
```

The OPEN HAND 1 command causes the gripper to open. The MOVE NEAR statement is used because, even though the gripper is already at position C, the MOVE NEAR instruction will cause the gripper to move above C again. The NOSETTLE termination type pulls the gripper near B without actually passing through it.

The three individual arm motions of the preceding example can now be organized into a single motion routine, as illustrated in Figure 7-27.

Figure 7-27 Motion routine for three arm motions

An important point: A robot must be in the same state at beginning and end of a motion routine. The arm must return to the start position and the gripper to the beginning

state (open or closed) of the routine. In fact, this starting position will be the same for all motion routines within the program. That is why the starting position is called the *safe position*: from this position it is safe to start and execute any motion routine. This is also called the *perch position* because in it the robot is similar to a bird perched in a tree, waiting to take off and execute a motion routine. The safe position should be an optimal position that minimizes the distance the robot arm must travel for each routine.

In order to minimize the number of instructions within the motion routine and to organize the routine in a standard logical manner, the format shown in Figure 7-28 is recommended.

Figure 7-28 Motion routine organization

The motion setup section will contain the instructions that set up or prepare the arm for motion. These instructions typically contain the default values for speed, acceleration/ deceleration, trajectory interpolation method, and termination. The next group of instructions provides actual arm motion commands. A motion routine concludes with some type of return instruction. Recall that the motion routine is called from the main logic section of the program, thus, controller focus must return to the main logic section. These instructions are logic instructions and will be addressed in more detail in the next section.

Using this format the motion routine instructions for the example shown in Figure 7-27 are organized as in Figure 7-29. Note that lowercase phrases preceded by a hyphen (-) are program comments. A well-commented program aids debugging.

```
Motion Setup    ⎧ – Setup section
    Section     ⎨ $SPEED  $SPEEDLIM
                ⎨ $MOTYPE = JOINT
                ⎩ $TERMTYPE = FINE
                ⎧ – Approach and Grasp Pin
                ⎪ MOVE NEAR A BY -75
                ⎪ DELAY 500
                ⎪ WITH $MOTYPE = LINEAR MOVE TO A
                ⎪ DELAY 500
                ⎪ CLOSE HAND 1
                ⎪ DELAY 500
                ⎪ – Move Pin to C
                ⎪ WITH $MOTYPE = LINEAR MOVE TO D
 Arm Motion     ⎪ WITH $STERMTYPE - NOSETTLE MOVE TO B
   Section      ⎨ MOVE TO E
                ⎪ DELAY 500
                ⎪ WITH $MOTYPE = LINEAR MOVE TO C
                ⎪ – Release Pin and go to Safe Position
                ⎪ OPEN HAND 1
                ⎪ DELAY 500
                ⎪ WITH $MOTYPE - LINEAR MOVE NEAR C BY -75
                ⎪ DELAY 500
                ⎪ WITH $TERMTYPE = NOSETTLE MOVE TO B
                ⎩ MOVE TO SF
  Return        ⎧ – Return to Main Logic Section of the Program
  Section       ⎨ GO TO MAIN
```

Figure 7-29 Motion routine instructions

Review the organization of the motion routine. The default speed, motion type and termination type are listed in the setup section. Each arm motion discussed previously is listed in the arm motion section. Note that a DELAY instruction was added after the instruction to move the pin to position E, to allow the arm to settle before pressing the pin in the hole. The routine concludes with a GO TO instruction. The GO TO instruction causes the *program focus* to move to a portion of the program labeled MAIN. Program focus refers to the part of the program code executed by the processor: The processor encounters a GOTO statement, jumps to a different part of the program, and executes the code located in that portion of the program. The processor thus shifts the program focus. This instruction, along with other logic instructions, will be discussed in the next section.

7.4.3 Communication and Logic Instructions

Modern industrial robots can acquire knowledge from the surrounding environment through electronic inputs. The robot program processes this information and directs the robot to act accordingly. The ability to acquire and apply knowledge toward a purposeful

goal is commonly known as *intelligence*. Thus, one can say a robot's intelligence is based on its ability to communicate (acquire information) and make logical decisions (apply knowledge). Accordingly, understanding how to gather information and make decisions within the context of the robot program is vitally important.

Communication instructions enable the robot to gather information by checking the status of equipment and sensors electronically connected to it. These electrical connections are called *inputs*. Additionally, the robot communicates its state or status to its surroundings through similar electronic connections called *outputs*. Communication instructions provide the interface between this input/output (I/O) system and the robot program. Based on the status of the inputs, the robot program will make decisions according to the program logic. *Logic instructions* establish the decision-making protocol of the program. The result of the decisions may be to execute a motion routine, turn on a light on the operator panel, request input from the operator through the teach pendant, or bring about any number of actions.

To see how communication instructions are used with the robot's I/O system and process sensors in robotic applications, consider Figure 7-30. This figure shows a robotic application in which a robot is used to move and load a part onto a turntable of a processing station. A sensor on the conveyor is activated or turned on when a part is in position, ready to be moved to the processing station. Another sensor is activated when the turntable is in position to be loaded.

Sensors measure process variables that provide information about the state of the robot's surroundings. They are electrical devices that are physically connected to the inputs of the robot. The simplest type of sensor is the switch. Switches measure discrete process variables. A discrete process variable has one of two values: on or off. There are numerous types of switches available, including pushbuttons, toggle switches, and limit switches. The two switches in Figure 7-30 are limit switches. (For a more detailed discussion of switches, discrete process control, and process variables refer to Chapter 8.)

As seen at the bottom of Figure 7-30, the robot controller enclosure houses input and output modules. For more detail the basic appearance of one module input module and one output module are shown in a larger size. Note how the part in the position sensor and the turntable in the position sensor are wired into the input module. Also, note the two lights to the left of the controller enclosure. These lights are to inform the operator of the status of the robot. They are outputs.

The application workflow is as follows. A part comes down the conveyor and is moved into the pickup position. This activates (turns ON) the part in position sensor. When the turntable in position sensor is also activated, the robot executes a motion routine to move the part to the turntable. Assume that the motion routine developed in the last section was developed for this application. While the robot is executing the motion routine, it will turn ON the robot busy light and turn OFF the robot ready light. When the motion routine is complete, the robot will turn OFF the robot busy light and turn ON the robot ready light.

Figure 7-30 Robot application example

Communication instructions enable one to control the I/O system of the robot. There are two major categories of inputs and outputs available with any robot: *peripheral devices* and *end effector tooling*.

The peripheral devices shown in Figure 7-30 include the part in position sensor, the turntable in position sensor, the robot ready light and the robot busy light. These devices will be hardwired by the end user into the I/O modules as is shown in the figure. The end user defines peripheral I/O as is required by the application. Thus, in KAREL this category is referred to as *user-defined I/O signals*.

Fanuc's KAREL language accesses input signals with the communication instruction DIN[n], where n is the signal number. This sort of instruction is called a *proram logic instruction*. Assume for the application shown in Figure 7-30 that the part in position sensor is connected to input terminal 1 and the turntable in position sensor is wired to

input terminal 5. Note that the program language treats I/O data as either ON or TRUE, which means the switch is active or ON; or as OFF or FALSE, meaning the switch is inactive or OFF. When the part-in-position switch detects a part, it is made active and DIN[1] = ON. Correspondingly, if DIN[5] = ON, then the turntable is in position.

KAREL assigns output values to the output terminals using the DOUT[n] instruction, where n is the signal number. Assume the robot ready light is wired to output terminal 1 and the robot busy light is wired to output terminal 3 for the application shown in Figure 7-30. Thus, to turn the robot ready light ON and the robot busy light OFF the following communication instructions would be issued:

DOUT[1] = ON
DOUT[3] = OFF

Recall in the last section the use of the OPEN HAND 1 and CLOSE HAND 1 in the KAREL motion routine for moving the pin. These instructions are communication instructions for the end effector tooling. Because robots are so often used in material handling applications, most—if not all—robot programming languages have dedicated end effector or gripper instructions. The end effector is typically electronically connected to the robot controller through the EE (end effector) cable. This cable passes through the robot arm, terminating near the end effector. This enables a short wiring distance from the end effector to the EE cable connection (Figure 7-31).

Figure 7-31 End effector electrical connection

In the case of an air actuated end effector, an air control valve would be connected to the EE cable. Compressed air, piped through the robot arm (in similar fashion as the EE cable) would also be connected to the valve and the end effector. The OPEN HAND or CLOSE HAND signal would activate the valve allowing compressed air to pass and thereby actuate the end effector.

Some end effectors, such as welding guns, engravers, and spindles may require additional outputs in order to operate. Additionally, some may have sensors attached that are important to program logic. Examples include gripper status (opened or closed) or

sensing, if a part is in the gripper. Accordingly, the EE cable is often a multiple pin connector capable of providing numerous input/output signals. In KAREL this I/O is called *robot digital input and output signals*. KAREL communication instructions for robot digital I/O are RDI[n] and RDO[n], respectively. These instructions function the same as the user-defined I/O signals discussed previously.

Some robot languages give the programmer access to operator panel I/O signals and/or teach pendant I/O signals. These signals enable the operator to interface with the program during program execution. This I/O is evaluated and manipulated much the same way that user-defined and robot-defined I/O signals are evaluated and manipulated. Specific programming language manuals provide additional information.

As has been mentioned, communication instructions gather knowledge and logic instructions apply that knowledge to a purposeful end. The next several paragraphs address how this is accomplished within the context of a robot program.

Recall from the section on robot motion routines how robot program instructions are normally executed sequentially line by line from top to bottom. When one instruction terminates or finishes, the controller executes the next instruction. When decision-making is involved instructions cannot be executed sequentially. Depending on the decision, the robot controller may need to move or jump to instructions elsewhere within the program. Accordingly, logic instructions provide a means to make decisions, jump to specific instructions (like motion routines), and repeat instructions until a condition is met (a process called *looping*, which we will return to).

The typical logic decision-making instruction is the IF statement. An IF statement evaluates a Boolean expression's truth value (i.e., if it is true or false). (A *Boolean expression* evaluates the status of a variable; for example, DIN[1] = ON.) If the expression is true the controller will execute one sequence of instructions; if it is false an alternative sequence of instructions will be executed. In KAREL, the IF statement has the format:

IF (Boolean expression)
THEN (true statement)
ELSE (false statement)
ENDIF

Meaning, in standard English,

If the Boolean expression is true, then execute the true statement;
if it is false, then execute the false statement.

Following the execution of either the true statement or the false statement, the controller will then execute the next instruction following the ENDIF. If ELSE is not used in the IF statement format, and the Boolean expression is false, the controller will execute the next instruction following the ENDIF.

For example, the following KAREL Boolean expression would be used to determine whether the digital input signal 1 is true, or ON:

DIN[1] = ON

The equal sign is called the *Boolean operator*. Other operators include greater than (>), less than (<), and not equal (<>). Two or more variables can be evaluated within a single Boolean expression by placing AND between the variables, as we will soon demonstrate.

True and false statements can either be executable KAREL instructions or a multiple instruction sequence. Often the true or false statements are *jump instructions*: They cause the controller to move, or "jump out of," the IF statement to another section of the program. They can be used anywhere in the program, not just in an IF statement.

In KAREL, the jump instruction is the GO TO instruction. The format is:

GO TO (statement label)

A *statement label* is a special identifier that designates the place in the program where program control is transferred. Statement labels should be descriptive in nature. Additionally, the controller identifies a label by two colons (::), which must immediately follow the instruction. Figure 7-32 shows an example KAREL program using communication instructions, an IF statement, and GO TO instructions identifying two different statement labels.


```
MAIN: – Main Logic Section
– Set Outputs
DOUT[3] = OFF
DOUT[4] = ON
IF DIN[1] = ON THEN GO TO UNLOAD
ENDIF
    .        – Additional Program Instructions
    .
    .
    .
GO TO MAIN
UNLOAD
    .  – Unloading Motion Routine Instructions
    .
    .
    .
GO TO MAIN
```

Figure 7-32 Example KAREL program using IF and GO TO instructions

Note the location of the statement labels MAIN and UNLOAD in the above figure, and that the two colons are absent when the label is listed in the GO TO instruction. The arrows show where program control is transferred when the GO TO instructions are

executed. The DOUT instructions set the devices wired to output terminal 3 and 4—OFF and ON, respectively. The IF statement checks if the digital input signal 1 is ON. If it is, program control is transferred to the UNLOAD motion routine. If it is not ON, additional program instructions are executed and eventually program control is transferred back to MAIN to repeat the main logic section. Note that after completion of the UNLOAD motion routine program, control is transferred back to MAIN.

The last type of program logic instruction that will be discussed in this text is the repetitive or *looping instruction*. Looping instructions cause the controller to repeat one or more program instruction either a specified number of times or until some condition is met. There are numerous ways to perform looping within a program, including through the use of GO TO and IF statements. However, when it is desirable to repeat a set of instructions a certain number of times in a KAREL program, the FOR instruction simplifies the programming. The FOR instruction provides looping based on an integer counter. The format of the FOR instruction is:

FOR count = initial TO final DO
(statements)
ENDFOR

Count is an integer variable that the program increments. "Initial" precedes the start value of count and "final" is the maximum value of count. Statements are program instructions that are repeated until the count equals "final." Figure 7-33 demonstrates how to use a FOR instruction to repeat a motion routine five times.

Figure 7-33 FOR looping instruction example

The FOR instruction sets the value of I to 1, then executes the motion instructions. When program focus gets to the ENDFOR, it is returned to the beginning of the FOR instruction. I is then incremented by 1. This continues until I attains the value of 6, which causes the program focus to jump to the first instruction following the ENDFOR.

Note that even though the communication and logic instructions discussed here are specific to Fanuc's KAREL language, all intelligent robots with a higher-level structured

334   Chapter 7

robot programming language containing similar if not identical instructions. The formats may be different, but the function is the same. Consider Figure 7-34a. This is a robot program for a Mitsubishi RV-E2 6-axis robot. Even though the formats for the instructions are vastly different than those of KAREL, their function is the same as that of KAREL.

Figure 7-34a Mitsubishi RV-E2 robot program

In the OB –1 instruction on line 40, the "OB" stands for output bit in the Mitsubishi robot programming language (called "movemaster"). The "1" indicates the output (bit) number and the hyphen (-) indicates that it is to be turned OFF. The equivalent instruction in KAREL would be:

DOUT[1] = OFF

Instead of label statements that designate the location to which a program should jump, the Mitsubishi language uses line numbers. Line 90 in the figure is essentially an IF statement. It checks whether input 7 is ON. If it is ON, the program focus jumps to line 330. Lines 150 and 160 are GO TO instructions. The reader is encouraged to attempt to rewrite this program in KAREL.

Armed with knowledge of motion, communication, and logic instructions one is prepared to program moderately complex robot applications. In the next section the program of instructions for the application shown in Figure 7-30 is written.

7.5 Writing Robot Program of Instructions

In this section the program of instructions for the application introduced in Figures 7-0 and 7-30 will be written. Recall from Section 7.4 that program of instruction writing is the first of four major program development stages. All stages are important, of course, but the first is critical because a correct and well-written program of instructions can dramatically reduce time and effort spent on the other stages.

The first step of writing program of instructions is to create a sketch or drawing of the application. This is shown in Figure 7-34b. The sketch represents the relative location of the application equipment, namely the conveyor, part, robot, and processing station. Each of the robot arm positions (A, D, B, E, C, and SF) is identified; the relative position and labels are the same as those in the motion routine developed in Section 7.4.2. Since most material handling application motions are very similar (for example, pick a part up from one location and move it to another), the reader will find many uses for the routine developed in this previous section.

Figure 7-34b Mitsubishi RV-E2 robot

Identify the inputs and outputs in a sketch of the application. Referring to Figure 7-30 and using the same assumptions given in Section 7.4.3, the two inputs, part-in-position and turntable-in-position sensors, are assumed to be wired into input terminals 1 and 5, respectively. In KAREL they are identified as DIN[1] and DIN[5], respectively. Correspondingly, the robot ready light is labeled DOUT[1] and the robot busy light is labeled as DOUT[3].

The next step is to develop the process flow. The robot will execute a motion routine to pick up the part and move it to the turntable when both part-in-position sensor and turntable-in-position sensor are activated. When the robot is executing this motion routine, the robot ready light will be off and the robot busy light will be on. At all other times the robot ready light will be on and the robot busy light will be off. A flow chart of the process flow is shown in Figure 7-36.

The first few steps involve setting the robot's status. The robot ready light is turned on and the robot busy light is turned off. Next the status of the part in position sensor is checked. If it is off the program loops back and resets the outputs and checks again. This loop continues until the part-in-position sensor detects a part. When a part is in part-in-position, the turntable-in-position sensor is then checked. If it is not in position, the program loops back through setting the output and checking the other input again. Finally, when both the part-in-position and turntable-in-position sensors are both activated, the robot's status is set to "busy" and the "move part to turntable" motion routine is executed. Once the robot has moved the part to the turntable, the program focus returns from the motion routine and the program loops back to the beginning. The flow chart for the motion routine is shown in Figure 7-35. The trajectories for each arm motion were shown previously in Figure 7-27.

Figure 7-35 Motion routine flow chart

Figure 7-36 Process flow chart

Following the robot program organization convention of Section 7.4 and as demonstrated in Figure 7-11, one recognizes that the flow chart of Figure 7-35 obviously shows the motion routine and Figure 7-36 represents the main logic section of the program

The next step in writing the program of instructions is to translate the process flow into the robot's programming language. A program sheet developed for the robot used in the application is a good aid for this activity. The robot for this application is a Fanuc 6-axis robot using an RH controller and KAREL programming language. Figure 7-37 shows a blank program sheet for this application.

KAREL RH Controller Program Sheet								
Program Name:						Sheet No.:		
Programmer:			Motion Instructions					Date:
Process Flow/Work Cycle Description	Line #	Logic & Communication Instructions	Motion Type	Term. Type	Speed	Position	Instruction	Complete Program Code

Figure 7-37 Blank programming sheet

Note that on the left side of the program sheet there is a column for writing out the process flow in standard English. The process flow chart serves as a guide for this activity. However, the process flow written on this sheet will typically have much more detail and many more steps than the typical process flow chart. The next several columns are for translating the process flow into robot instructions. The first column after the line numbers is for logic and communication instructions. The next five columns are used to develop motion instructions. These columns list the pertinent data necessary to specify a motion instruction (motion type, termination type, speed, position, and instruction). The far right column is a culmination of all the program instructions written in the other columns. The complete program instructions or code appears in this column in the precisely the format that will be entered into the robot controller. This sheet should be filled out top to bottom and left to right.

Sheet 1 of the completed program sheet is shown in Figure 7-38. On it is listed the logic portion of the program. Sheet 2, shown in Figure 7-39, shows the motion routine. In Figure 7-38, observe the symbol "==>" in the logic and communication instruction column when a motion instruction is required. This directs the reader's attention to the motion instruction columns. Also, recall that two dashes (--) indicate comments in a

KAREL program. A well-commented program is easier to follow and will aid with future troubleshooting and debugging.

KAREL RH Controller Program Sheet							
Program Name: Program 1						**Sheet No.:** 1	
Programmer: DEK		**Motion Instructions**				**Date:**	
Process Flow/Work Cycle Description	**Line #**	**Logic & Communication Instructions**	**Motion Type**	**Term. Type**	**Speed**	**Position**	**Instruction**	**Complete Program Code**
Specify program name	1	PROGRAM PROG1						PROGRAM PROG1
Program Comments	2	-- This program moves a part from a conveyor to						-- This program moves a part from a conveyor to processing station
I/O Comments	3	-- I/O Definition						-- I/O Definition
"	4	--DIN[1] - part in position sensor						--DIN[1] - part in position sensor
"	5	--DIN[5] - turntable in position sensor						--DIN[5] - turntable in position sensor
"	6	--DOUT[1] - robot ready light						--DOUT[1] - robot ready light
"	7	--DOUT[3] - robot busy light						--DOUT[3] - robot busy light
Declare variables including positions	8	VAR						VAR
"	9	SF: POSITION						SF: POSITION
"	10	A: POSITION						A: POSITION
"	11	B: POSITION						B: POSITION
"	12	C: POSITION						C: POSITION
"	13	D: POSITION						D: POSITION
"	14	E: POSITION						E: POSITION
Start program	15	BEGIN						BEGIN
Set default motion type	16	==>	JOINT					$MOTYPE=JOINT
Set default termination type	17	==>		FINE				$TERMTYPE=JOINT
Set default speed to maximum	18	==>			$SPEEDLIM			$SPEED=$SPEEDLIM
Extablish main logic section and label	19	MAIN::						MAIN::
Turn robot ready light ON	20	DOUT[1] = ON						DOUT[1] = ON
Turn robot busy light OFF	21	DOUT[3] = OFF						DOUT[3] = OFF
Check status of part in position sensor	22	IF DIN[1] = OFF THEN						IF DIN[1] = OFF THEN
If off, loop back to MAIN::	23	GO TO MAIN::						GO TO MAIN::
End the If statement	24	ENDIF						ENDIF
IF ON, check turntable in position sensor	25	IF DIN[5] = OFF THEN						IF DIN[5] = OFF THEN
If off, loop back to MAIN::	26	GO TO MAIN::						GO TO MAIN::
End the If statement	27	ENDIF						ENDIF
If it is ON, go to Motion Routine	28	GO TO MOVE::						GO TO MOVE::

Figure 7-38 Sheet 1 of completed program sheet

KAREL RH Controller Program Sheet

Program Name:	Program 1								
Programmer:	DEK						Sheet No.:	2	Date:
				Motion Instructions					
Process Flow/Work Cycle Description	**Line #**	**Logic & Communication Instructions**	**Motion Type**	**Term. Type**	**Speed**	**Position**	**Instruction**		**Complete Program Code**
Label motion routine	29	MOVE::							MOVE::
Comment	30	-- Approach and grasp pin							-- Approach and grasp pin
Move near position A by -75	31	=>	Joint	Fine	-	A	MOVE NEAR		MOVE NEAR A BY -75
Delay half a second	32	DELAY 500							DELAY 500
Move to grasp pin	33	=>	Linear	Fine		A	MOVE TO		WITH $MOTYPE = LINEAR MOVE TO A
Delay half a second	34	DELAY 500							DELAY 500
Close the gripper	35	CLOSE HAND 1							CLOSE HAND 1
Delay half a second	36	DELAY 500							DELAY 500
Perform arm motion 2	37	-- Move part to C							-- Move part to C
Pull part out	38	=>	Linear	Fine		D	MOVE TO		WITH $MOTYPE = LINEAR MOVE TO D
Move near B without stopping	39	=>	Joint	Nosettle		B	MOVE TO		WITH $TERMTYPE = NOSETTLE MOVE TO B
Move to E	40	=>	Joint	Fine		E	MOVE TO		MOVE TO E
Delay half a second	41	DELAY 500							DELAY 500
Put part in turntable	42	=>	Linear	Fine		C	MOVE TO		WITH $MOTYPE = LINEAR MOVE TO C
Release part and go to safe position	43	OPEN HAND 1							OPEN HAND 1
Delay half a second	44	DELAY 500							DELAY 500
Move near position C by -75	45	=>	Linear	Fine	-	C	MOVE NEAR		WITH $MOTYPE = LINEAR MOVE NEAR C BY -75
Delay half a second	46	DELAY 500							DELAY 500
Move near B without stopping	47	=>	Joint	NOSETTLE		B	MOVE TO		WITH $TERMTYPE = NOSETTLE MOVE TO B
Move to the safe position	48	=>	Joint	Fine		SF	MOVE TO		MOVE TO SF
Comment	49	--Return to Main::							--Return to Main::
Go to main	50	GO TO MAIN::							GO TO MAIN::
End the program	51	END PROG1							END PROG1

Figure 7-39 Sheet 2 of completed program sheet

There are several logic and communication instructions in Figure 7-38 that have not been discussed thus far. Each of these instructions is required in KAREL programs. In line 1 the PROGRAM instruction specifies the name of the program as PROG1. Lines 8 though 14 represent the variable declaration portion of the program. KAREL requires that all variables, including positions, be declared at the beginning of the program. The VAR instruction denotes the beginning of variable declaration. The variable name and type are separated by a colon. Accordingly, each position used in the program is listed with the type set as POSITION. Other types of variables include INTEGER and REAL for integer and real number variables, respectively. Line 15 contains the BEGIN instruction. This instruction denotes the beginning of the actual program code. Line 51 shown in Figure 7-39 is also required in a KAREL program. The END instruction informs the robot controller that it has reached the end of the program.

Figure 7-40 shows the completed program with each program section identified. Although, different robot languages may have unique requirements, the general organization convention shown in the figure should be followed. The first part of the program should be heavily commented to describe the program, identify important variables, and indicate the inputs and outputs. The next section, if required, should be used for declaring variables. This is followed by the main logic portion of the program and then the subsequent motion routines.

Recall from Section 7.4 that the last step of writing the program of instructions is to simulate the program, which we next address.

Program name

Program and
I/O description

Variable
declaration

Start of
program

Motion defaults

Main Logic
Section

Motion Routine

Program end

```
PROGRAM PROG1
-- This program moves a part from a conveyor to processing station
-- I/O Definition
--DIN[1] - part in position sensor
--DIN[5]  - turntable in position sensor
--DOUT[1] - robot ready light
--DOUT[3] - robot busy light
VAR
SF: POSITION
A: POSITION
B: POSITION
C: POSITION
D: POSITION
E: POSITION
BEGIN
$MOTYPE=JOINT
$TERMTYPE=JOINT
$SPEED=$SPEEDLIM
MAIN::
DOUT[1] = ON
DOUT[3] = OFF
IF DIN[1] = OFF THEN
GO TO MAIN::
ENDIF
IF DIN[5] = OFF THEN
GO TO MAIN::
ENDIF
GO TO MOVE::
MOVE::
-- Approach and grasp pin
MOVE NEAR A  BY -75
DELAY 500
WITH $MOTYPE = LINEAR MOVE TO A
DELAY 500
CLOSE HAND 1
DELAY 500
-- Move part to C
WITH $MOTYPE = LINEAR MOVE TO D
WITH $TERMTYPE = NOSETTLE MOVE TO B
MOVE TO E
DELAY 500
WITH $MOTYPE = LINEAR MOVE TO C
OPEN HAND 1
DELAY 500
WITH $MOTYPE = LINEAR MOVE NEAR C  BY -75
DELAY 500
WITH $TERMTYPE = NOSETTLE MOVE TO B
MOVE TO SF
--Return to Main::
GO TO MAIN::
END PROG1
```

Figure 7-40 Completed program

7.6 Robot Simulation

Robot simulation involves simulating the robot program and the program of instructions in the virtual world of a computer graphics program. Program simulation is typically accomplished by modeling the robot, tooling, and peripheral equipment in the software and then executing the program of instructions for the application. The result is an animation of the robot arm executing the program of instructions. The simulation provides a means to test, simulate, and verify program logic and arm motions prior to running the robot program on the actual robot. The animation gives visual confirmation that the program is performing as intended, so when the program is finally sent to the machine to be run, the programmer will have a high degree of confidence in the outcome. Simulations are also beneficial for interference detection, arm motion planning, and trajectory generation. Additionally, if a robot must be purchased for a new application, the user can first easily run simulations of several different robots to determine the one that best satisfies the application requirements.

For the beginning robot programmer, robot simulation software is an invaluable tool. It enables the student to see, firsthand, how the robot arm will move in response to the program of instructions. As it true for all simulation programs, robot simulation helps the user to more deeply understand both arm motion instructions and programming process. Additionally, simulations are performed in the safety of the virtual world, so when program errors occur, which inevitably happens with beginning programmers, risks to the student, tools, and machine are eliminated.

There are numerous robot simulation software packages available for purchase. Most major robot manufacturers have developed and marketed simulation software for their particular robots. In most situations the level of software sophistication is directly proportional to the required investment. In this section we introduce the reader to an inexpensive simulation program capable of simulating most types of robots. It is called RobotAssist^{TM}, published by New River Kinematics. RobotAssist^{TM} is capable of simulating robot configuration that the user builds and models. It is supplied with a library of over 50 robot models. A screen shot of the software with a robot from the library is shown in Figure 7-41. Because the publishers consider the software educational, they make it available to students for a very low price. At time of publication, the cost of RobotAssist^{TM} to students was under $20. For non-students, a single user license was available for only $69.95. For its low price and high capability, this software offers excellent quality.

Figure 7-41 RobotAssist^{TM} screen shot

RobotAssist^{TM} is available for download from the New River Kinematics at www.kinematics.com/products/educational/robotassist/. The system requirements listed for best performance are:

Windows 95®, 98®, ME®, NT®, 2000® or XP® operating system
Intel Pentium® microprocessor
32 megabytes of system RAM
40 megabytes hard drive storage space
1024 x 768 video driver with 64K color depth (4 meg VRAM preferred).

RobotAssist^{TM} installation includes a comprehensive electronic user manual file. The reader is strongly encouraged to review this information in detail. In the next several sections we only highlight some key aspects of this software's powerful capabilities, including installation, the basics of the user interface, building a custom robot, and simulating simple robot programs. For more detailed information the reader is encouraged to consult the user manual.

7.6.1 RobotAssist^{TM} Installation

Using a web browser, go to New River Kinematics website (www.kinematics.com/products/educational/robotassist/) and purchase and download an individual student license.

Locate and execute the robotassist_install.exe file. This starts RobotAssist^{TM} setup (Figure 7-42). This first screen shows the License Agreement. Review the license agreement and select the appropriate button.

Figure 7-42 RobotAssist^{TM} License Agreement screen

The next screen is the Choose Destination Location (Figure 7-43). The default location is C:\Program Files\New River Kinematics\RobotAssist. Click Next when ready to continue; cancel to terminate setup.

Figure 7-43 Destination location screen

The program will begin installing. When installation is complete the screen will appear as shown in Figure 7-44. Select the desired options and click the close button.

Figure 7-44 Installation Finished screen

7.6.2 User Interface

When RobotAssist^{TM} is started the screen will momentarily appear as is shown in Figure 7.45. After approximately three seconds the RobotAssist^{TM} logo, or splash, in the center of the screen will disappear and the user interface will look as is shown in Figure 7-46.

Figure 7-45 Startup interface

Figure 7-46 RobotAssist^{TM} user interface at startup

The user interface consists of three main areas: A pulldown menu for selecting commands, a toolbar containing icons of commands, and the graphics area where robot models will be displayed. Many of the icons on the toolbar are grayed out and thus inactive because no robot file is open.

Select a robot file to open from the pulldown menu. This causes the Open dialog box to appear as shown in Figure 7-47. The Sample Files directory is the collection of sample robot files supplied with the program. In a standard installation the directory is located at C:/Program Files/New River Kinematics/RobotAssist/Sample Files. Additionally, this directory serves as the default directory for newly created robot models. Select the FANUC1.ROB as shown in the figure and click the Open button. The screen now appears as is shown in Figure 7-48.

Figure 7-47 Open *dialog box*

The object on the left is the 5-axis Fanuc 1 robot. Also shown is a conveyor with four parts sitting on it. The software lets the user model peripheral equipment, tooling, and parts for any given application. Opening the robot file activates the toolbar icons and also adds additional pulldown menus. The File pulldown provides a means for manipulating robot files. The Edit pulldown menu can be used to copy robot states and images to the clipboard and start the creation of animation files. Additionally, the RobotAssist[TM] System Options dialog box (called User options in the pulldown) is accessed through this menu. As would be expected, the View pulldown provides options for manipulating the view of the graphics area. The Object, New, and Group pulldown menus allow modeling of robots and other objects. Control is for exercising manual control of the robot and executing programs. The Kinematics pulldown is for setting the parameters for the robot's configuration.

The toolbar contains icons that serve as shortcuts to the more commonly used commands. To determine what each icon does, simply place the cursor over an icon; within a second the name of the icon will appear. For instance, to change to a front view, simply select the Front icon as shown in Figure 7-49.

Figure 7-48 FANUC1.ROB robot file

Figure 7-49 Front view

To create a custom view, select Point of View from the View pulldown. This opens the Point of View dialog box as shown in Figure 7-50. From this dialog you can manipulate the view by selecting the appropriate button manipulator, as shown in the

figure. To close the Point of View dialog box click the X in upper right corner of the dialog box.

RobotAssist^{TM} also has three options for shading objects, including wireframe, flat shading, and advanced shading. Advanced shading gives the smoothest and most realistic appearance. These options can be accessed through the View pulldown menu or via icons on the toolbar.

Figure 7-50 Point of View dialog box

Manual manipulation of the robot model is accomplished through a teach pendant interface much the same way as it is with a real robot. To access the teach pendant for the model simply select Teach Pendant from the Control pulldown menu. This causes the Pendant dialog box to open (Figure 7-51). The top select box of the pendant is for selecting the appropriate jog coordinate system: joint, world, or tool. The default jog coordinate system is joint. It is listed as Joints (1−6) in the select box pulldown list. When this coordinate system is selected, each joint of the robot model can be moved individually. World causes the robot to move in the *x*-, *y*-, or *z*- axes of the world coordinate system located at the robot's base. Tool causes the robot to move relative to the coordinate system of the tool. The appearance of the teach pendant in each of these cases is shown in Figure 7-52. Once the jog coordinate system is selected the user can jog the robot by selecting the appropriate button manipulator. The robot model will move accordingly. The reader is encouraged to experiment with the teach pendant control to

gain a thorough understanding of jogging coordinate systems. Additionally, it may be beneficial to load other robot files and experiment with other types of robot configurations. The user manual gives additional information on the user interface and general operation of the RobotAssist^{TM}.

Figure 7-51 Pendant dialog box)

Joint Jog Coordinate System

World Jog Coordinate System

Tool Jog Coordinate System

Figure 7-52 Pendant jogging coordinate systems

352   Chapter 7

7.6.3 Building a Custom Robot

RobotAssist[TM] comes with over 50 standard robot models. However, if a robot of interest is not included, RobotAssist[TM] provides the capability to build a custom or user-defined robot. This will be demonstrated in the building of a robot model of the Fanuc A510 robot. The completed robot model with an attached gripper is shown in Figure 7-53. This model was built in essentially four steps, given below.

Figure 7-53 Fanuc A510 robot model

The first step in creating a custom robot model is to gather information about the robot. For instance, the modeler needs to know the number of robot joints, type of joints, and link lengths. This information can easily be obtained from the manufacturer's specifications for the robot to be modeled. For example, the Fanuc A-510 robot is a 4-axis robot with three rotational joints and one linear joint. The joints along with the relative link lengths are shown in Figure 7-54.

Figure 7-54 Fanuc A510 joints and link lengths

The next step in modeling the robot is to use the Robot Generator in RobotAssist[TM] to construct a skeletal starting model that contains the correct number of links and joints. Prior to accessing the Robot Generator be sure to close any open robot files. Next, select the "create a new robot" file by selecting New from the file pulldown menu and then select Robot when prompted. The Robot Generator is accessed from the file pulldown menu. This opens the Robot Generator dialog box (Figure 7-55). Set the Total Number of DOF to 4 and Length of Typical Link to 400, as shown in the figure, and click OK. The skeletal serial linkage along with the default Denavit–Hartenberg parameters are

shown in Figure 7-56. Obviously, the skeletal linkage does not resemble, in the least, the finished robot model shown in Figure 7-53. The following steps, beginning with the editing of the Denavit−Hartenberg parameters, enable the modeler to change the skeletal linkage to match the desired linkage configuration.

Figure 7-55 Robot Generator dialog box

Editing the parameters listed in the Denavit-Hartenberg Parameters dialog box alters the skeletal model's configuration. The parameters are based on the standard Denavit−Hartenberg notation, which uses only two values to describe the link and two values to describe the joint between adjacent links. Links are rigid members between joints. Each link is associated with the joint at the end of the link. The naming convention for links and joints in RobotAssist[TM] is shown in Figure 7-57. Note that link 1 is considered the base of the robot.

Figure 7-56 Skeletal linkage and Denavit-Hartenberg dialog box

Figure 7-57 RobotAssist^{TM} link and joint naming convention

The Denavit–Hartenberg parameters enable the development of a transformation matrix such that any point on any of the links defined by the local link coordinate system can be mapped to the world coordinate system. In this way the position and orientation of the robot's links and joints, called a *pose*, can be evaluated. Note that RobotAssist[TM] documentation refers to coordinate systems as *coordinate frames*.

Links are defined by length (a) and twist angle (alpha) depicted graphically in Figure 7-58. *Link length* (a) is the distance between the two joint axes measured along a line that is normal (i.e., perpendicular) to both joint axes. This normal line between both joint axes is called the *common normal*. The *twist angle* (alpha) is the angle between the first and second joint axis when viewed along the common normal. See figure 7-58 again to see this for a single link between two joints. For this example a = 10 and alpha = −45 deg. Also, note that in some industrial robot configurations both the link length (a) and twist angle (alpha) can be zero.

Figure 7-58 Defining a and alpha for a single link

Consider Figure 7-59. Joints are described in terms of *offset* (d) and *joint angle* (theta). Offset (d) is the distance between the common normal line between joint axes of one link and the common normal line between joint axes of the next link measured along the axis of the joint connecting the two links. Joint angle (theta) is the angle between the common normal of one link and the common normal of the next measured about the joint connecting the two links. Note that the joint angle (theta) will change as the robot changes positions, whereas the other parameters will stay fixed once set in the dialog box. This figure shows two arbitrarily named links, Link X and Link Y. Note that Link X has a nonstandard shape. Verify that the offset (d) and joint angle (theta) are dimensioned correctly in figure.

$\mathbf{a}_X = 10 \qquad \mathbf{d}_X = 5$

$\mathbf{alpha}_X = 0 \qquad \mathbf{theta}_X = 25 \text{ deg.}$

Figure 7-59 Defining d and theta for Link X

To determine the Denavit–Hartenberg parameters for the A510 robot refer to Figure 7-54 and prior definitions for link length (d), twist angle (alpha), offset (d), and joint angle (theta). Accordingly, edit the parameters as shown in Figure 7-60. To edit a parameter select the link number in the dialog box and enter the correct values in the boxes at the bottom of the dialog box. Be sure to select the appropriate joint type in the bottom right of the box. Note that Revolute indicates a rotating joint and Prismatic is a linear joint.

Notice the appearance of the skeletal model after editing the Denavit–Hartenberg parameters. Even with a great imagination it may be difficult to correlate the skeletal model to the actual A510 robot configuration. However, keep in mind that the dark slender cylinders (which are blue on the computer screen) protruding from the side of the joints represent the common normal of the link, not the actual link! Thus, the key to seeing the skeletal model as the actual robot is to try ignoring the blue cylinders and to overlay the actual shapes of the links over the skeletal model. This is shown in Figure 7-61, which is a wireframe view of the finished robot. As one gains experience with the software, this visualization becomes much easier.

Figure 7-60 Denavit–Hartenberg parameters for a Fanuc A510 robot

Figure 7-61 Wireframe view of finished robot showing joints and common normals for

Fanuc A510 robot model

The last step in modeling a robot is to create or import the robot structure. This will give the robot a more realistic appearance. RobotAssist^{TM} provides a means to create realistic models of the robot's links, called *objects*. Objects are made by creating and combining primitives, user-defined extrusions, and user-defined rotations. Also, one can create custom objects by importing either stl or iges files exported from commercially available solid modeling software such as AutoCAD, SolidWorks, or Pro/Engineer. Whichever method is used one must understand the concept of grouping to create realistic robot models.

Grouping in RobotAssist^{TM} enables the user to associate an object with other objects. The first object created in the group is called the *parent object*. Other objects grouped to parent objects are called *child objects*. Child objects are always positioned relative to and move with the parent object. For our robot model, the joint object (the red cylinder) is the parent object of each link group. The common normal object (the blue slender cylinder) is a child of the group. Thus, in order for robot link objects to move realistically, they must be grouped to the appropriate link. This is demonstrated below.

Within RobotAssist^{TM}, double click on the first joint and select cylinder from the pulldown menu, as shown in Figure 7-62. Double clicking on the joint causes it to change color to black. Also, this action defines the joint as the parent of the new object created and adds the new object to its group. Thus, when this joint moves the new object will move with it.

Figure 7-62 Creating a new object that is grouped to the first joint

Clicking on cylinder from the New pulldown menu causes the cylinder dialog box to open. From within this dialog the size and number of facets for the new object can be specified. See that the higher the number of facets, the smoother the object's appearance when shaded. Clicking on the Transform button opens the cylinder position dialog box. This dialog box specifies the location of the new object's coordinate frame relative to the parent object's coordinate frame. Clicking on the Material button opens the Material dialog box, from which one can specify the color of the new object (Figure 7-63).

Figure 7-63 Cylinder, Cylinder Position, *and* Material *dialog boxes*

The reader is encouraged to experiment with creating objects, linking them to different groups, and observing how they move when the robot joints move. Once created, the objects can be modified from the Object pulldown menu. Simply double click on the object to be edited and select modify from the Object pulldown menu. The contents of this menu are shown in Figure 7-64. Refer to the RobotAssist[TM] user manual for additional information on editing objects.

Figure 7-64 Cylinder, cylinder position, and material dialog boxes

Since most engineers and technologists are already proficient with one of the many commercially available solid modeling programs, it is typically easier and faster to use that software to model the link geometry and then import the models into RobotAssist^{TM}. This is the method the author used to model the A510 robot. The link geometry was modeled using Pro/Engineer, exported from Pro/Engineer as an ASCII stl file and then imported into RobotAssist^{TM} as a user-defined custom object. This was accomplished by selection of User Defined from the New pulldown menu and then clicking on Custom, as shown in Figure 7-65. This action opened the Custom dialog box (Figure 7-66). From this dialog the type of file to import was selected. The author used the stl file type and made sure to double click the parent of the import object prior to selecting Custom from the pulldown menu. Once imported, the object was repositioned to the correct orientation using the Transform button. See RobotAssist^{TM}'s user manual for additional information.

Figure 7-65 Importing stl files

The above method allows that the robot's link geometry and gripper can be imported to give a very realistic model, as is seen in Figure 7-54. Other objects also can be modeled, including conveyors, parts, and other machines. If new objects are going to be immobile, that is to say, not move as the robot moves, then they should be grouped to the world coordinate system. To accomplish this, we do not double click on any objects prior to creating the new object. Once the model and all peripheral equipment are created, robot motion can be simulated.

Figure 7-66 Importing link geometry from stl files

7.6.4 Simulating Robot Motion

RobotAssist[TM] provides realistic simulation of industrial robots. A robot model can be jogged into position using the teach pendant just like a real robot. Similarly robot positions and orientations can be stored. The robot can then be driven to these positions at a later time. Refer to RobotAssist[TM]'s user manual for additional information.

The robot model can also be programmed to simulate motion using one of two methods. The first method is a form of motion programming using the Sequence Control dialog box. This method is very similar to Fanuc's teach pendant programming. Points are stored and motion instructions are entered as the robot is jogged through the steps of the work cycle. The second method is called Control Programming. It is robot language programming method using RobotAssist[TM]'s control language. It is simple to use and is effective in creating realistic animations of robot arm motions. Each method is discussed below.

To access the Sequence Control dialog box, select Sequence from the Control pulldown menu as is shown in Figure 7-67. This opens the Sequence Control dialog box. When using this method also open the Teach Pendant dialog box from the Control pulldown menu. This enables the user to jog the robot through the work cycle steps. Rearrange each dialog box such that they do not overlap and the robot model is visible, as shown in Figure 7-68.

Figure 7-67 Opening the Sequence *dialog box*

Figure 7-68 Creating a motion program dialog box

To create a motion program, select the New from the Sequence Control dialog box. This clears any other programs. To enter a program step, perform the following:
Jog the robot to the desired location using the Teach Pendant.

Click on the Insert After button to insert a sequence step. The first step will be labeled Step #0.
Select the appropriate command from the Motion Command select box.
Click on Update Using Current Values button to store the current robot position
Repeat.

These steps are also shown in Figure 7-69. When the program is complete, select the Run button to execute the program animation. The program can then be saved by clicking on the Save button. Accordingly, previously created programs can be loaded by using the Load button.

Figure 7-69 Steps to create a motion program

RobotAssist^{TM} can create complex animations with its control language. As mentioned, this method is analogous to using a robot programming language on a real robot. First the robot is jogged to the desired location and orientation and that position, or *state*, as it is referred to in RobotAssist^{TM}, is identified and stored. These states then get pasted into the control program. After all the states are stored, a control program written using RobotAssist^{TM}'s control language can be created. The control program is written using a standard text editor. Each state is added to the beginning of the control program. Once all the states for the program are stored, the program is written. The control language commands then reference these states while specifying the appropriate arm motion. For a complete command reference, refer to RobotAssist^{TM}'s user manual.

7.7 Robot Program Simulation Example

In this section the program developed in Section 7.5 and shown in Figure 7-40 will be simulated using RobotAssist^{TM} simulation software. Although RobotAssist^{TM}'s programming language has logic and communication commands, robot programmers are typically

most interested in simulating motion routines. So, only the motion routine of the program will be simulated.

The first step in simulating the program is to model the necessary application components. In this case, the application uses a Fanuc S12 robot, which conveniently is included in RobotAssist[TM]'s library of robot models. Thus, a robot model does not need to be created. However, the gripper, conveyor, obstacle, and turntable will be modeled. The completed model is shown in Figure 7-70.

Figure 7-70 Model of robot application

Once the model is complete, the control program can be written. Open a text editor such as Microsoft Notepad[TM] and position the window next to the RobotAssist[TM] window as shown in Figure 7-71. Any standard text editor can be used to create a control program.

Figure 7-71 RobotAssist^{TM} and a text editor

Next the robot is jogged into position using the teach pendant. To save the position (state) of the robot, select Copy State to Clipboard as shown in Figure 7-72. This opens the Copy State to Clipboard dialog box. Name the state in the box provided and hit OK. Use the naming convention used in the actual program that is being simulated. Accordingly, the first position is the safe position (SF). Next, paste the contents of the clipboard into the text editor. The naming of the state and pasting it into the text editor is shown in Figure 7-73. Continue this process until all the states are copied and pasted into the text editor.

Figure 7-72 Saving the robot state

The next step is to write the motion routine using RobotAssist^{TM}'s control language. Although there is an extensive list of commands, most simulations can be written using only two motion commands, *warp* and *interp*. The warp command moves the robot arm directly to a state without animation. Interp moves the arm to a state by interpolating the

motion in a specified series of steps, which provides smooth animation for the best simulation experience. Thus, warp is only used to move the arm rapidly to the start position, followed by interp for the animation. Treat interp as if it were a MOVE TO instruction in the KAREL language. A portion of the resulting motion control program is shown in Figure 7-74. A complete listing of the available commands is found in the user manual.

Figure 7-73 Naming the robot state and pasting it into the text editor

Figure 7-74 Portion of RobotAssit^{TM} control program

In the format of the interp command, the command name is followed by the state name to which it will move, and then by the number of interpolation steps it will take to get there. States AA and AboveC are positions not included in the KAREL program. They are necessary here, however, because RobotAssist^{TM} does not have an equivalent command for KAREL's MOVE NEAR instruction. Also, some program logic is applied. At the start of the program, a label is used called *main*. The jump command at the end of the program causes the program to jump back to main and start over. When this program is simulated it will run continuously until the user aborts it. A complete listing of the available RobotAssist^{TM} commands is given in the user manual.

When the control program is complete, save it, adding the .prg file extension to the end of the name as you do. Also, if you are using Notepad, set Save as type to All Files and the Encoding to ANSI as shown in Figure 7-75.

Figure 7-75 Saving the program

To run the program, select Execute Control Program from the Control pulldown menu or select the Run Program icon, as shown in Figure 7-76. This will cause the Control Program Observer dialog box to open, which enables the user to pause, resume, or abort the control program. The reader is encouraged to write a control program using one the standard robot models included with RobotAssist^{TM}. Additionally, experiment with different interpolation steps to adjust the smoothness of the animation.

This example shows just some of the capabilities of RobotAssist^{TM}. Refer to the user manual for additional information. This program and others like it can be a great help to the new programmer as he or she acquires the capability to program and operate industrial robots.

7.8 Summary

A robot program consists of logic instructions that specify when actions are to occur, named locations of where the robot arm is to move, motion instructions to specify how the arm is to move, and I/O instructions to provide communication with peripheral equipment.

A robot program is a set of program language instructions that specify a robot's end effector path, makes logic decisions, and executes peripheral actions that are necessary to support a work cycle. The end effector's path is specified with a combination of stored robot arm positions and program motion instructions. The stored positions are locations in space that the end effector must achieve or pass near in order to execute the work cycle. Motion instructions dictate the stored position the arm is to move to and the trajectory of the path to reach it. I/O instructions provide a means to communicate with peripheral equipment. Logic instructions specify when work cycle actions are to occur.

Robot programming occurs when the robot program is entered into the robot controller's memory. This is accomplished in one of two ways. Motion programming involves moving the robot arm through the work cycle, storing arm positions and entering instructions for each step of the cycle. In this method the robot is dedicated to the programming task. Robot language programming, on the other hand, separates the programming of the motion, logic, and I/O instructions from the teaching of the robot arm positions. The only time the robot is unavailable is when positions are being taught. Development of the actual program is performed offline away from the robot.

Robot positions are taught by jogging the robot to the desired position with a teach pendant and then storing the robot's joint positions to the robot controller. The three jogging coordinate systems used to aid the programmer in orienting the robot arm in the correct position include the joint, world, and tool coordinate systems.

There is no standard robot programming language. Each language is unique to the robot brand under consideration. However, programming concepts are generally transferable across robot platforms. Most languages are typically higher-level structured languages designed to simplify the amount of data that the programmer needs to input. The Fanuc brand of robots utilizes the KAREL programming language.

A robot program should be organized to have a main logic section wherein the robot communicates with the outside world and makes logic decisions. This serves as the jumping off point to the motion routines, which define where and how the robot arm will move. Each motion routine may consist of several arm motions.

The required information that arm motion instruction must specify includes the trajectory, acceleration/deceleration, speed, and termination. The trajectory of the motion

is determined by the interpolation method specified. The three standard interpolation methods are joint, linear, and circular.

Communication instructions enable the robot to gather information about its environment by checking the status of equipment and sensors electronically connected to it. Based on the status of the inputs to the controller, the robot program will make decisions according to the decision-making protocol established by logic instructions.

Once the robot program of instructions is written it can be simulated in the virtual world of a computer graphics program prior to loading it on the actual robot. Program simulation is accomplished by modeling the robot, tooling, and peripheral equipment in simulation software and then executing the program of instructions. This creates an animation of the arm motion. Simulation provides a means to test, simulate, and verify program logic and arm motions prior to running the program on the actual robot.

7.9 Key Words

arm motions
circular interpolation
communication instructions
input and output instructions
joint coordinate system
joint interpolation
jump instructions
KAREL
linear interpolation
logic instructions
looping instructions
manual leadthrough
motion instructions
motion interval
motion programming
motion routines

multiple segment motion
powered leadthrough
robot arm positions
robot language programming
robot program
robot programming
robot simulation
RobotAssist^{TM}
teach pendant
teach pendant programming
termination
tool coordinate system
trajectory
user coordinate system
velocity profile
world coordinate system

7.10 Review Questions

1. Define a robot program.
2. How is a robot end effector's path programmed?
3. Discuss the role of logic instructions in a robot program.
4. Explain the difference between motion programming and robot language programming.

  5. What is the role of the teach pendant in teaching robot arm positions?
  6. Explain the difference between the joint coordinate system and the tool coordinate system with respect to jogging a robot's end effector into position.
  7. Explain how a user coordinate system is used in robot programming.
  8. What is the name of the programming language used by the Fanuc brand of robots?
  9. Define teach pendant programming.
 10. Name the four major program development stages.
 11. List and explain the four steps for writing a program of instructions.
 12. Define termination in terms of a motion interval.
 13. List and explain the three standard interpolation methods.
 14. Define a robot's perch position.
 15. List and discuss four types of motion instruction termination possible in the KAREL language.
 16. Which KAREL motion instruction cases the tool centerpoint to approach a desired position by an offset distance?
 17. What is the KAREL instruction to close the robot gripper?
 18. What is the KAREL instruction to turn on a digital output?
 19. Write the program of instructions in the KAREL language to have a Fanuc S12 6-axis robot pick up a pen from a pen holder and write your initials to fit in a 10-inch wide by 5-inch high box. Have the robot return the pen to the pen holder when complete.
 20. Using RobotAssist[TM], simulate the program written in review question 20.

7.11 Bibliography

  1. Groover, M.P. 2001, *Automation, Production Systems and Computer-Integrated Man-ufacturing*, 2nd edition, Prentice Hall, Upper Saddle River, New Jersey.
  2. Rehg, J.A. 2003, *Introduction to Robotics in CIM Systems*, 5th edition Prentice-Hall, Upper Saddle River, New Jersey.
  3. KAREL Reference Manual, Version 3.06P, 3.06PA, 3.06 PB, MARSKKREF072 04E, 1992,FANUC Robotics North America, Inc.
  4. Robot Assist[TM] User's Manual, 2002, New River Kinematics, Inc., Pulaski, Virginia.

Chapter 8

Introduction to Programmable Logic Controllers (PLCs)

Contents

Objective

The objective of this chapter is to provide a thorough explanation of the terminology and basic operating concepts of programmable logic control (PLC) technology.

8.1 Programmable Logic Control Overview

In discrete *manual* manufacturing processes the operator typically (1) does the material handling of a workpiece, (2) processes it using tools and/or machines, and (3) makes decisions regarding when each task will be performed. In Figure 8-0 a human operator is is shown carrying out a hole-punching process using a piercing die with the aid of a manually actuated press. The operator picks up a workpiece from the raw material bin, accurately positions the workpiece in the die, and actuates the press with a foot switch. After the workpiece is pierced, the operator removes it from the die, inspects it, and places it in either the finished goods or scrap bin. These steps are repeated until the raw material bin is empty. So, the operator handles material, moves it, processes it, and perhaps most important, makes decisions for controlling the process (e.g., when to pick up the workpiece and place it in the press, when to actuate the press, and when to place the workpiece in the finished goods or scrap bins).

Figure 8-0 Manual piercing die process

Now, let us consider what would be involved in the *automation* of this process. Automation of the same process shown in Figure 8-0 would require some form of material handling *equipment*, which would move the workpiece from the raw material bin to the press, then from the press to the finished goods or scrap bin—an ideal application for a robot. The press and piercing die already perform the processing step effectively, when so instructed by the operator (the foot switch). The challenge lies in the programming of the automated decision-making and/or process control: The robot must be told when to load the press, when to unload the press, when to place the workpiece in the finished goods bin, and so on. Additionally, the press needs to be actuated when a part is correctly positioned in the die. The standard device used in industry to provide this type of control over processes is

called a *programmable logic controller*, more commonly known as a PLC, which is the same acronym for *programmable logic control*. PLC technology imparts automatic control over tasks or events through the use of electrical and computer technology. A PLC device is essentially a standardized computer system specifically designed to interface with industrial components and equipment. PLCs are made to be rugged withstand harsh industrial environments. A standard industrial PLC is shown in Figure 8-1.

Figure 8-1 Picture of a PLC

In order for a PLC to control the process depicted in Figure 8-0, for example, it would have to collect information from the process and implement decisions based on this information by interfacing with the process through sensors and actuators. Figure 8-2 shows how a controller, such as a PLC, interfaces with a process. *Sensors* collect information from the process, and *actuators* execute decisions by acting on the process.

Figure 8-2 PLC–process interface

As an example, consider what occurs when an operator places a workpiece in a piercing die. The operator senses, through vision, that the part is located correctly in the die, then presses the foot switch to execute the piercing operation; that is, the operator's eyes are the sensors and the footswitch is the actuator. So, for every instance in which a human operator would collect data from a process, a PLC needs a sensor, and, when an operator acts on the process, a PLC actuator is needed.

One possible way to automate our example is shown in Figure 8-3. The sensors in this type of application are simply electrical switches; the actuators, robot, and press, are by electrical switches or signals.

Figure 8-3 Sensors and actuators integrated into Figure 8-0

The PLC makes its decisions based on the type of control that is to be implemented, which in turn depends on the tasks to be done and the process being controlled. We now address this in detail.

8.2 Industrial Process Control

Process variables provide information about a process. They are used to evaluate how a process is performing. They are measured by sensors and *input* to the controller. *Process parameters,* on the other hand, are the *set points* or *targets* for the process. The controller compares the process variables to the process parameters and reacts accordingly by *outputting* instructions to the actuators.

Figure 8-4 is an example of a rudimentary oven that is used in a manufacturing process. The oven has an internal electrical heating element and a thermocouple to monitor oven temperature. An operator (the controller) adjusts oven temperature by switching a heating element on or off. If oven temperature is less than desired temperature

(referred to as the *temperature set point*) then the heating element is turned on. Once the set point is reached the operator turns off the heating element. Hence, temperature set point is the process parameter, the actual temperature is the process variable, and the heater is the actuator.

Figure 8-4 Oven control

Figure 8-5 shows a plot of actual temperature and set point temperature of an oven over time. The amount of variation between the two values is due to the nature of the system under control. The response of the system (its actual temperature) lags behind the heating element's actuation. In other words, by the time the operator sees that the set point temperature is reached and turns off the heating element, much energy has been pumped into the oven and it continues to heat up. Thus, the actual temperature overshoots the set point temperature. A similar scenario occurs as the oven cools. Therefore, the controller must *continually* monitor the process and actuate the heating element accordingly to minimize variation between target and actual temperatures. Because the actual temperature can have numerous values it is considered a *continuous process variable*. Since numerous temperature set points can also be specified, the set point temperature is

considered a *continuous process parameter*. Accordingly, this type of process control is called *continuous process control*.

Figure 8-5 Actual vs. set point

temperature

In contrast to continuous process control, consider a tank-filling process (Figure 8-6). In this process an operator opens a valve to allow fluid to enter the tank. A light, actuated by a float switch located in the tank, indicates when the desired fluid level (the process parameter) is reached. When the light comes on, the operator closes the filling valve. The on/off status of the light is the process variable. Note that the float switch does not measure intermediate values of the fluid level. It only measures one position. Therefore the light can only have one non-zero value: on. Variables of this type are called *discrete process variables*. Correspondingly, since the fluid level in the tank can only have one non-zero value, full it is considered a *discrete process parameter*. Since both variables and parameters are discrete, controlling the level of fluid in the tank is a form of *discrete process control*.

Figure 8-6 Tank-filling

The two main types of industrial process control are summarized below:

Continuous process control—Variables and parameters are continuous; they can take on numerous values.

Discrete process control—Variables and parameters are discrete; they can have binary values only: 0 (off) or 1 (on).

Whether a process is continuous or discrete, the controller may employ one of two methods to control the process, either *closed loop control* or *open loop control*. Figure 8-7 shows schematic diagrams of both types of systems. In closed loop control the process variable is continuously compared to the process parameter by way of a feedback loop. In open loop control the process is controlled without monitoring any process variables. Both the oven temperature control and the tank-filling process use closed loop control.

Figure 8-7 Closed loop and open loop control

However, in some process control applications open loop control is employed, in which, as the name implies, there is no feedback loop. A process is controlled without measuring a process variable. Open loop control should only be used when the following criteria are met:

- The process being controlled is simple, well-defined, and clearly understood.
- The response of the process to actions of the actuators is predictable and highly reliable.
- Reaction forces opposing actions of the actuators are minimal.

Figure 8-8 shows an open loop control example in a continuous process control application. The figure shows control of a single axis of a CNC machine. In this case, a stepper motor is used to position the table. Each electrical pulse from the controller causes the stepper motor to rotate—a rotation that is converted into a small constant linear movement of the table. Figure 8-9 shows the same application, but here it depicts closed loop control. *For most industrial process control applications closed loop control is required.*

Figure 8-8 Open loop control example

There are different *levels of control*. They are:
- basic device control,
- procedural machine control,
- coordinated system control.

The level used in turn influences which of the two types of *process control* are used:
- continuous or
- discrete.

Figure 8-9 Closed loop control example

Figure 8-10 demonstrates the hierarchical nature of industrial control.

Figure 8-10 Control hierarchy

At the first level *basic device control*, devices or components of a larger process are controlled. Sensors, actuators, and an individual controller execute the task performed by a device. Additionally, the task is executed independent of the higher level controller.

Control of a single axis of a CNC machine (Figures 8-8 and 8-9) is a good example of basic device control, as is the oven example.

Procedural machine control is typical of machine and workstation level control applications. Tank-filling, which uses discrete process control, is more than likely part of a larger process, so it belongs at procedural machine control. (It would be at the level of basic device control, however, if it had its own controller dedicated to the performed task.) Manufacturing processes, executed as they are by machines and workstations, are inherently procedural and sequential: An operator completes step A before completing step B and moving on to step C. A controller monitors, checks, and controls the status of each step of a process before it proceeds to the next.

Consider the polymer compression molding process shown in Figure 8-11. Material is loaded in the press; the operator presses the start button. Relying on input from the start button sensor, the machine controller actuates press closing. The speed, acceleration, and distance of the mold closure is controlled by a basic device controller similar to single axis control shown in Figure 8-9, except the press uses hydraulic servo valves and a hydraulic cylinder as actuators. Once the press is closed it must remain closed long enough for the material to solidify or cure. When curing is complete the press will open, again at a precisely controlled rate, and the part will be unloaded. To sum up, the machine controller exercises procedural control over the process, uses sensors to collect data from the process, and actuates devices and other actuators to execute the process steps. CNC machines and robots are other great examples of procedural machine control.

Figure 8-11 Polymer compression molding process

In flow-line manufacturing or work cell processing, groups of machines and support equipment work in conjunction to process a product. This requires that sensors, actuators, devices, and individual machines be controlled and coordinated to perform the processing. This level of control is called *coordinated system control*. Refer back to Figure 8-3, the automated piercing operation. The individual devices, part feeder, press, and robot must be coordinated to process the workpiece.

As our focus is on PLCs, we will limit our discussion a higher level of control—the coordinated system control level. This discussion could continue on to higher levels of control, such as at the *plant and corporate level*, but this is beyond the scope of this text. PLCs are used predominantly at the procedural machine and coordinated system levels. In fact, PLCs were developed specifically for procedural machine and coordinated system control. Hence, they exercise what is called *discrete sequential process control*. The parameters and variables of the system are changed at discrete moments in time based on the program of instructions (the sequential list of actions necessary to complete the process). Changes to the process parameters and variables, as defined by the program of instructions, are made when either the state of the system under control has changed (*event-driven change*) or a certain amount of time has elapsed (*time-driven change*).

An event-driven change is pictured in Figure 8-11 (the polymer compression molding process example). After the operator loads the raw material in the press, he or she presses the start button, so the state of the system has changed, causing the PLC to execute the action that closes the press. This is also an example of a time-driven change. After the press is closed, it must remain closed for a given amount of time, to allow the material to cure. Once the PLC has determined that the press is closed, it counts down the required cure time and then executes the action of opening the press. The PLC has executed a response at either a certain time in the process or after a specific time lapse has occurred.

This is the essence of discrete sequential process control executed by a PLC in an industrial environment at the procedural machine and coordinated system levels: The PLC changes or switches the status of the actuators in response to changes in events or time as defined by the program of instructions.

8.3 PLC Terminology

8.3.1 Programmable Logic Controller

Now that we have a basic understanding of industrial control, we can provide a more thorough definition of programmable logic controller: The PLC is a microprocessor-based discrete process controller that uses stored instructions in programmable memory to implement logic, sequencing, and math control functions for procedural machine and/or coordinated system control of machines and processes.

8.3.2 Work Cycle Program

The stored instruction set that is programmed into a PLC is called the *work cycle program*. The work cycle program is derived from the program of instructions and/or the process flow for the application being controlled. It is an intermediate step in the actual programming of the PLC. A work cycle program is created by dividing and subjugating the sequential list of actions specified by the program of instructions into logic and sequencing instructions. It can also be represented as a *timing diagram*. Examples of work cycle programs and their development are discussed in subsequent sections.

Consider Figure 8-12, which demonstrates how a PLC functions. The PLC compares the status of the inputs, which are wired to the sensors of the process, against the work cycle program logic and sequencing instructions. It then takes appropriate action by switching on and off outputs that control the actuators acting on the process. The logic instructions specify which outputs to turn on or off in response to event-driven changes. The output values are determined solely by the value of the inputs. The PLC does not consider the previous values of inputs—only the current state. The sequencing instructions, on the other hand, specify output activity based on time-driven changes. In this case the PLC uses an internal timing device to determine when to initiate changes to outputs.

Figure 8-12 PLC operating cycle

8.3.3 Operating Cycle and Scan Time

The *operating cycle* of a PLC, as shown in Figure 8-12, appears to be instantaneous and continuous. In other words, the PLC appears to instantly respond to process or system changes. In fact, it does respond extremely rapidly, but not instantaneously. For each operating cycle the PLC actually completes three *scans*. The first scan is called the *input scan.* During this scan the PLC will read the status of the inputs and store them into memory. The second scan is the *program scan*. During the program scan the PLC will take the stored input values and conduct the logic control calculations specified by the work cycle program. Lastly, the PLC will perform an *output scan* in which it will update status of the outputs. The time it takes to perform these scans is called the PLC's *scan time*. It is important the PLC's scan time be faster than the time between input changes; otherwise

the PLC will not be able to respond quickly enough. Scan times for most PLCs range from 1 to 3 milliseconds.

8.4 PLC Hardware Components

A PLC consists of the following component hardware:

- Processor
- Memory unit
- Power supply
- Input/output (I/O) module
- Programming device.

The *processor* examines the status of the input signals, executes the logic and sequencing functions, and operates on the outputs. It consists of one or more microprocessors specifically designed for input and output (I/O) type operations.

The *memory unit* stores the work cycle program, I/O status information, and controller system operation information. It is broken down into two areas: user memory and system memory. The work cycle program is stored in the user memory and the controller operational information is kept in the system memory. The combined processor and memory unit is called the central processing unit, also known as the CPU.

The *power supply* provides the power to the unit. PLCs are specifically designed to run off a standard 120-volt AC line, a design feature that was part of the specification for the first PLC.

The controller physically connects with the sensors and actuators through the *I/O module*. This is the interface to the process application being controlled. Like the power supply, I/O modules were originally specified to accept 120-volt AC signals, intended to enable the PLC to easily connect with standard push buttons and limit switch sensors. However, since the early development of the PLC, I/O module capabilities have greatly expanded. I/O modules are now available in a wide variety of configurations, including, but not limited to:

- AC input/AC output
- AC input/DC output
- AC input/relay output
- DC input/DC output
- DC input/AC output
- DC input/relay output.

Many additional I/O module configurations are available, including *analog I/O*, which enables the PLC to perform continuous process control, and advanced communication/networking modules. The type and number of I/O modules selected is strictly dependent on the application.

The last hardware component of the PLC is the *programming device*. This device provides a means of entering the work cycle program into the memory module of the PLC. In the early days of PLCs this device was similar to the *robotic teach pendant*. It was connected to the PLC with an interface cable as shown in Figure 8-13. The program was entered into the memory module by pressing the required keys of the device. When programming was completed the device could be used to edit, test, and debug the program. After proving out the program, the device was disconnected and stored or used for programming other PLCs. Although pendant-type programming devices can still be found in industry, the modern programming device is a laptop personal computer, as shown in Figure 8-14.

Figure 8-13 PLC pendant-type programming device

Figure 8-14 Laptop used as a programming device

The laptop PC has many advantages over the pendant-type device, including ease of program entry and superior program editing and debugging capabilities, thanks to its large LCD screen. Additionally, the program can be developed, edited, and tested offline away from the PLC. Once the program is verified, the laptop can be connected to the PLC and

downloaded to the memory unit of the PLC. Thus, the laptop also serves as an additional storage device for the PLC.

The programming, editing, and debugging of PLC programs are discussed in great detail in Chapter 9.

8.5 PLC Applications

All the processing examples discussed thus far have been simple by intention, in order to demonstrate the concepts of industrial process control and the capabilities of PLCs. These applications of these examples could be controlled by traditional electrical *hardwiring* methods instead of with a PLC. Hardwiring implies physically connecting, with wire, individual electrical components, such as relays, timers, counters, and other electrical devices that perform the process control. Before the advent of the PLC, hardwiring was the only method available to control industrial processes. In fact, essentially all pre-1968 industrial control applications involved hardwiring.

Consider the control of the polymer compression press shown in Figure 8-11. Controlling this process through hardwiring would require the use of numerous electrical switches and relays to perform even the simplest of tasks, including turning on the hydraulic pump, actuating the hydraulic valve to close the press, and turning on the electric heaters to heat the mold. Additionally, at least one timer would be required to control the cure time. The wiring necessary to connect these devices and perform the control logic would be quite involved, requiring a rather large, complex control cabinet.

An additional complication is that, typically, in compression molding a press closes at different speeds: first, quickly until it is almost closed, then at a slower pressing speed. Dictating how this task would be accomplished with hardwiring would be somewhat difficult in control logic and add even more complexity to the hardwired system. Because of this complexity, maintenance and troubleshooting would be challenging. If a different mold were placed in the press, many switches, timers, and temperature controllers would have to be readjusted or reprogrammed for the new mold. A major change to the control logic would require rewiring. Interfacing the press with other machines would be very difficult if not nearly impossible. PLCs were specifically developed to overcome the complications and limitations of hardwired control applications.

In the late 1960s the Hydramatic Division of General Motors, recognizing the limitations of hardwired control systems, developed a set of specifications for a system that would replace them (as outlined in M.P. Groover's book) and put out a call for proposals. The specifications included requirements that the device:

- be programmable and reprogrammable;
- be designed to operate in an industrial environment;
- accept 120-volt AC signals from standard pushbutton and limit switches;
- have outputs designed to switch and continuously operate loads such as motors and relays.

In 1968, Richard Morley, a partner in a research firm that specialized in control systems for machine tool companies, presented the winning proposal, earning for himself the title "Father of the PLC." Morley and his partners formed a new company called Modicon to produce the new PLC, and the era of the PLC was born.

Thus, since its inception the PLC has proven to possess many advantages over hardwired control systems, including these attributes:

- PLC programming is much easier to effect than the hardwiring of relays, timers, counters, and other devices. As explained in Chapter 9, a PLC program can be developed, edited, and tested offline, away from the machine. Thus, proof of concept can be determined and the process well understood before the actual machine programming.
- PLCs can be reprogrammed quickly and easily. A new program can be developed and written offline while the machine is still running. Once the program has been verified it can be downloaded to the PLC and run.
- Electrical cabinets for PLC-controlled applications are much smaller than hardwired electrical cabinets for the same application, so the PLC cabinet takes up much less floor space.
- Maintenance and troubleshooting on PLC controllers is much easier than with hardwired control systems. Hardwired systems have numerous components, making maintenance and troubleshooting time-consuming and complicated. Additionally, PLCs tend to be much more reliable than hardwired systems.
- PLC-controlled equipment can be easily interfaced with other PLCs and automated equipment, allowing its integration with other PLCs and programmable automation (robotics and CNC machines) to create automated work cells and systems.
- PLCs can perform a multitude of control functions, for example, analog control, arithmetic functions, matrix functions, and data processing.
- Despite the numerous advantages of PLCs over hardwired control, the latter is still a viable option for many simple control applications. However, as the required level of control becomes more complicated and the specifications for the control system come to match the original list proposed by GM decades ago, PLCs inevitably prove to be the best solution.

8.6 Sensors and Actuators

One of the most important considerations in sizing and selecting the correct PLC for an industrial control application is type and number of inputs and outputs required. This choice is dictated solely by type and number of *sensors* and *actuators* necessary to control the process. Additionally, the programmer's understanding of how sensors and actuators function, their variety, and their capabilities are critically important to effective PLC programming.

8.6.1 Sensors

Sensors provide a means by which the controller interfaces with the process. Sensors are measurement devices that monitor and feed back information to the controller about the status of the process variables. Whereas humans rely on their sensory perception to monitor environment, a controller must rely on mechanical/electrical devices for feedback about the process environment. Each measurement device is designed to measure a very specific process variable and feed back the information to the controller in a distinct way. The type of process variable measured and how the information is fed back to the controller determines the classification of the sensor. For general automation purposes, sensors can be separated into three major categories: (i) *switches*, (ii) *transducers*, and (iii) *special purpose sensors*. A diagram of the major sensor categories is shown in Figure 8-15.

Figure 8-15 Sensor categories

Switches measure discrete process variables. They provide binary (on or off) data back to the controller about the status of the discrete process variable. For discrete process control at the procedural machine and coordinated system level of control, switches are the primary sensors used. However, at the basic device level of control, transducers dominate. Transducers convert a physical variable, temperature, force, or position, for example, into an alternative electrical form that is fed back to the controller. Thus, transducers measure continuous process variables. Finally, special purpose sensors consist of self-contained devices with sensors and a controller specifically designed to interface with PLCs and other programmable automation machines (robots and CNC equipment). They are used at the coordinated system level of control. Examples include vision systems and barcode reading devices.

We now look at these three sensor categories in detail.

8.6.1.1 Switches

The tree basic switch types available for control applications are *mechanical*, *electrical*, and *optical*. Each type can be wired in a binary fashion, that is, to close or open an electrical contact. The physical state of a switch can be either *normally open* (*NO*) or

normally closed (*NC*). NO switches have contacts that close upon action, whereas NC switches have contacts that open upon action. The three types of switches and corresponding physical states are shown schematically in Figure 8-16.

Figure 8-16 Switch styles and physical states

A *pushbutton* is a spring-loaded switch, mechanically actuated by a person. For NO switches the contacts remain closed only as long as the button is held down. Once the button is released the spring returns the switch to its normal state.

A *toggle switch* is also mechanically actuated; however, once actuated, it does not return to its normal state unless it is actuated again. A typical household light switch is a good example of a toggle switch.

Figure 8-17 shows a typical control panel with pushbuttons and toggle switches. Note that even though the emergency stop (E-stop) button looks like a pushbutton, it is actually a toggle switch. This is necessary from a safety standpoint: Once a process is stopped in an emergency situation it should remain stopped: the E-stop cannot return to its normal state until the operator takes action.

Figure 8-17 Control panel

Common places for pushbuttons and toggle switches are start/stop buttons (which initiate/stop a process), motor start buttons, and mode (automatic or manual) selector switches. They are typically placed in the *control panel* of the machine or workstation, where an operator (top-level controller) gathers information about a process. It is the primary interface between the human operator and the PLC. Besides having pushbuttons and toggle switches, a control panel typically consists of lights, digital readouts, and perhaps a CRT or LCD user interface.

 Another kind of switch is the *limit switch.* Limit switches provide the primary binary interface between the PLC and the process. They are considered "presence" sensors because they detect the presence of some object in the process, such as the product itself or some linkage or part of the machine being controlled.

Both contact and non-contact methods exist for detecting the presence of objects. A *contact limit switch*, as the name implies, must physically come into contact with the object it is meant to detect. In this way it mechanically actuates the switch linkage. Figure 8-18 shows a schematic of a contact limit switch. Note that it may be wired normally open (NO) or normally closed (NC). An NO switch causes the switch contacts to close; an NC switch causes them to open.

Figure 8-18 Lever type contact limit switch

Contact limit switches are simple and are available with a wide variety of linkages and corresponding actuation forces. Some common linkage arrangements are shown in Figure 8-19. These designs have been proven accurate and reliable.

Figure 8-19 Contact limit switch linkages

Contact limit switches must be rugged to withstand a harsh industrial environment. However, the constant opening and closing of the contacts presents wear issues. These issues, along with concerns about the speed of response, may dictate the use of *non-contact limit switches*. Because of their capacity to sense the presence of an object only when it is in close proximity, non-contact limit switches are traditionally termed *proximity* or "prox" switches.

The three common non-contact limit switches are *inductive proximity switches*, *capacitive proximity switches*, and *optical proximity switches*. These switches, according to their type, use built-in transducers and electrical circuits to open or close internal contacts. The inductive proximity switch utilizes a transducer that measures changes to a magnetic field in order to detect the presence of electrically conductive materials. The switch is activated when it detects these changes. Changes in the magnetic field occur when an electrically conductive object is present. This is illustrated in Figure 8-20. Inductive proximity switches are relatively small, rugged, and low cost. This makes them the most widely used proximity switches. Their limitations include insurance of a consistently repeatable sensing distance and ability to detect only those materials that are electrically conductive.

The capacitive proximity switch solves some of the limitations of the inductive proximity sensor. It can sense virtually any object: if an object can hold an electrical charge, a capacitive proximity switch can sense it. The switch uses half a capacitor as its transducer. Recall that capacitors are devices used to store electrical energy. They consist of two plates separated by a dielectric (non-conducting material). Without these two plates the capacitor could not store electricity. This is what enables a capacitive proximity switch

to detect the presence of objects. When the object is in position to be sensed, it becomes the other half of the capacitor of the switch, with the air separating the switch and object serving as the dielectric. So, when an object gets close to the switch, the now completed capacitor accepts a charge and its internal circuitry causes the switch to open or close, depending on its state (NC or NO).

Figure 8-20 Inductive proximity switches

Capacitive proximity switches look very similar to inductive proximity switches, though they are larger. They are typically more expensive than inductive proximity switches. These switches require calibration for sensing different types of materials. Additionally, the sensing distance can be adjusted. Limitations include a tendency for errant readings when many different types of objects are in the field of view. In general, capacitive proximity switches must be used with care because of their tendency to be overly sensitive.

The optical proximity switch consists of a light source and a transducer, called a *light sensor*, for the obvious purpose of detecting light. Optical proximity switches switch on or off according to whether the light sensor can detect light from the light source. They are also called "photoelectric sensors." They come in two basic configurations: *through-beam* or *retroreflective*. Through-beam optical proximity switches have the light source and light sensor in separate housings. The switch signals the presence of an object that blocks the light beam between the source and sensor. Some types can be calibrated finely enough to detect changes in the quantity of the beam received. For example, such a type could be used in bottle-filling workstations to determine if a bottle has been filled properly. Retroreflective optical proximity switches have light source and receiver in the same housing. This configuration requires that the object sensed be reflective. Figure 8-21 depicts a through-beam application.

Figure 8-21 Optical through-beam proximity switch application

8.6.1.2 Transducers

The second category of switches is transducers, which are used in applications where monitoring of continuous process variables is required. Recall that transducers convert a physical variable into an electrical signal that is fed back to a controller. The electrical signal can be either analog or digital pulse. Transducers are most commonly used in basic device level control.

Consider the oven control example discussed previously and shown in Figure 8-4. An operator controlled oven temperature by turning a heating element on and off. A more realistic example of how the oven would be controlled is shown in Figure 8-22. Here the temperature readout and off/on switch (of Figure 8-4) is replaced with a temperature controller. The temperature controller automatically regulates the oven temperature by comparing the temperature set point (process parameter) against the actual temperature (process variable) measured by the thermocouple transducer, then actuates the heating element accordingly. It utilizes PID (proportional, integral, and differential) control loops to adjust the power to the heating element, providing precise temperature control. Since temperature control is of great importance in many manufacturing processes, temperature controllers are readily available for purchase from numerous vendors. Note, however, that most modern PLCs equipped with analog I/O can provide continuous process control at the basic device level as well.

Figure 8-22 Oven control example

If the oven were to be used at the procedural machine or coordinated system control levels, the device level controller would need to discretely interface with the main controller. Thus, dedicated temperature controllers and other basic device level controllers typically come with discrete event inputs and outputs.

Examples and descriptions of common analog transducers are listed in the table of Figure 8-23. The analog transducers listed here could be further categorized by the type of application to be controlled. For environmental control applications, RTDs, thermistors, and thermocouples typically measure temperature, while strain gages can be configured to measure pressure. LVDTs and potentiometers measure position, accelerometers measure acceleration and velocity, and strain gages measure force. Clearly these variables are of interest to one who designs a machine for controlling motion.

Transducer	Description
Accelerometer	Measures acceleration of objects to which it is connected. Used in vibration and shock applications
LVDT	Linear variable differential transformer. A position transducer that utilizes primary and secondary coils to monitor the position of a moveable core.
Potentiometer	A position transducer that monitors the position of a contact slider on a resistor. Measured resistance determines position. Linear and rotary potentiometers are available.
RTD	Resistance temperature detector. Temperature measuring transducer in which temperature is determined by change in resistance of temperature sensitive transducer material.
Strain gage	Used to measure force, torque or pressure through the change in electrical resistance of the gage due to mechanical strain.
Thermistor	Temperature-measuring transducer in which temperature is determined from decreases in electrical resistance of transducer material as temperature increases.
Thermocouple	Temperature-measuring transducer in which temperature is determined from the small voltage, which is proportional to temperature, emitted between two dissimilar metal wires exposed to a temperature source. Typical metals used in thermocouples include chromel-alumel, iron-constantan and chromel-constantan.

Figure 8-23 Common analog transducers

One of the more common devices used in motion control—the *optical encoder*—is not shown in the Figure 8-3, as it is a *digital* device. It digitally emits a series of electric pulses back to the controller as an indication of the value of the physical variable measured.

An optical encoder uses a light source, light sensors, and a perforated rotating disk to determine position, velocity, and acceleration of a rotating shaft. The basic configuration is shown in Figure 8-24. The perforated disk separates the light source and light sensors. The disk's perforations or slotted holes are arranged in a precise pattern. As the disk rotates the light sensors detect the light as a series of pulses. The number and frequency of the pulses are proportional to the shaft position and speed. Hardware differentiation of the velocity can be used to determine acceleration of the shaft.

There are two types of optical encoders: *incremental* and *absolute*. The output signals of incremental encoders repeat with each revolution. Thus, the precise position of the disk is not known when power is first applied to the encoder at startup; incremental encoders require homing or initializing each time power is applied. Absolute encoders, on the other hand, know exactly the mechanical position each time power is applied; thus, homing is not required. Incremental encoders are simpler and less expensive than absolute encoders.

Figure 8-24 Optical encoder

8.6.1.3 Special purpose category

Finally, we discuss the special purpose category. The main sensor of interest in this group is the vision system. As automation applications continue to grow the ability of a system to gather thoroughly the data from the process environment becomes ever more important. Whereas individual sensors provide one small "bit" of information about the process environment, a vision system can provide a complete "picture" of the same environment. Although a comprehensive discussion of vision systems is beyond the scope of this text, it is important we present an overview of the basic operating principles and potential applications.

Figure 8-25 shows a typical application of a vision system. In this application a vision system evaluates the acceptability of a molded part. If the part is acceptable the vision system takes no action and the part proceeds down the conveyor. If, however, the molded part is unacceptable, the vision system sends a signal to the process controller, which in turn actuates a cylinder to knock the part into the scrap bin.

Figure 8-25 Vision system application

As shown in the figure the major components of a vision system are camera, light source, image-capturing system (also called a "frame grabber"), and vision system controller. The light source illuminates the scene to be captured. Proper lighting is critically important to the quality of scene capture. The choice of tactic used from among the numerous methods and techniques depends on the type of vision system desired and the process environment. The camera captures the scene by converting the light levels to electrical current; the current is passed as analog data to the image-capturing system. The image-capturing system consists of electrical circuitry and programming that converts the analog picture data to condensed, digital data that is passed to the vision system controller. The vision system controller then compares the image to the one stored in memory. Based on this comparison the vision system controller outputs the appropriate course of action to the main process controller.

This example oversimplifies the workings of a vision system, but it is intended to identify some key components and provide a simple example of a vision system's capabilities. Other more complex capabilities of vision systems are the capacity to:

- recognize parts presented in various positions or orientations
- recognize parts that are touching or piled on top each other
- measure part dimensions
- determine part orientation and distance
- determine part speed and direction of motion
- determine part position in three dimensions.

The last three items are typical of the type of information needed by a robot. In fact, many of the applications for vision systems occur when a robot is involved. However,

vision systems are used in many different applications and are an excellent method of gathering a comprehensive amount of information about the process environment without using an excessive amount of main controller inputs.

It is desirable for the person selecting sensors for a specific application to obtain ones that are highly accurate, reliable, and precise. Equally important is that the sensors have a wide operating range, a high speed of response, and be low in cost.

8.6.2 Actuators

Actuators in a process controller activate the program of instructions; simply put, they perform work on the process. They allow process variables to come into agreement with process parameters. In automation, actuators are hardware devices, machines, or systems that convert a controller command signal into a change in a physical parameter. Often the control signal must be amplified if it is to reach sufficient power for the work that is involved. An actuator may be a simple electrical relay, a pneumatic logic circuit of solenoid valves and cylinders, a robot, or even a CNC machine.

The process and type of control signal amplification generally dictates actuator type. Actuators are best classified according to control signal amplification type: *pneumatic actuators* use compressed air to amplify the control signal; h*ydraulic actuators* use a pressured fluid, typically oil-based, for amplification; *electrical actuators* use electricity.

8.6.2.1 Pneumatic actuators

Air cylinders are by far the most common type of pneumatic actuators. They consist of a piston, rod, and cylinder housing for converting pressurized air into a corresponding linear force. A cutaway of a typical air cylinder is shown in Figure 8-26. Cylinders are specified by stroke length and inner diameter. The force exerted by a cylinder is equal to the product of its piston area and air pressure applied. Air cylinder types include single-acting (spring return), double-acting, and rodless. Single-acting cylinders use air pressure to actuate the cylinder in one direction and a spring to actuate it in the other. Double-acting cylinders use air pressure to move the cylinder in both directions. Rodless cylinders have only a cylinder and piston. An outer carriage is coupled to the piston with a strong magnet as shown in Figure 8-27. Air cylinders are available in a variety of standard inner diameter sizes, stroke speeds, and numerous mounting configurations to suit almost any application.

Figure 8-26 Air cylinder cutaway view

Figure 8-27 Rodless air cylinder

Other types of pneumatic actuators include air motors and vacuum cups. Air motors use either pistons or vanes to institute rotary motion. Piston air motors offer higher starting torque while vane-type air motors are simpler and more compact. Vacuum cups use the *Venturi effect* to form vacuum suction within flexible cups, enabling the cups to stick to objects. ("Venturi effect" is simply air pressure reduction caused by constriction of airflow through a tube or pipe.) Vacuum cups are often used as grippers in robot applications.

Regardless of type of pneumatic actuator device in use, the process controller cannot interface directly with the device. A *control valve* is needed to take the controller's electronic signal and convert it into an air signal, which goes into the actuator. A complete pneumatic system is shown in Figure 8-28. The major components of a pneumatic system typically consist of an air supply system, air conditioning system, various air control valves, and actuator devices.

Figure 8-28 Pneumatic system

The supply system is made up of an air compressor and either single or multiple storage tanks. Most industrial facilities have an air supply system installed with a plentiful supply of 90–100-psi compressed air. Whenever air is used it should be conditioned before it enters the actuated devices. Conditioning systems consist of an air filter, which cleans the air of debris, a pressure regulator, and a lubricator, to ensure proper lubrication of the pneumatic components. A control valve consists of a housing and a movable internal spool, which controls direction of airflow through the valve.

The directional control valve (Figure 8-28) is a *4-way, 2 position control valve*. It consists of five ports and has two spool positions. Figure 8-29 shows one valve with its ports labeled in two ways. The "4-way" of its name refers to the four combinations of flows possible between the ports: P to A; B to T; P to B; and A to T. "2-position" specifies that the spool has two possible positions, as depicted in the figure. Each spool position dictates the path the air flows between ports. Spool position can be changed manually with pilot air or electronically with solenoids. The air control valves of interest in process automation and control are *solenoid-actuated directional control valves*.

Spool
Position 1

Spool
Position 2

"Ways":   P to A
           B to T

"Ways":   P to B
           A to T

Figure 8-29 4-way, 2-position directional control valve

Solenoid-actuated control valves use a solenoid to change the position of the spool. A spring is used to hold the spool in the non-energized position in a 2-position control valve. The solenoid is used to change the spool to the other position. A solenoid works by utilizing electrical current in a coil to generate a magnetic field, which pulls an armature into the magnetic field. In a solenoid-actuated control valve the spool is connected to, or is part of, the armature. Thus, as the armature moves, so does the spool. When the current is removed, the spring causes the spool to return to the first position. This is demonstrated in Figure 8-30. The schematic symbol for the valve is also shown in the figure.

Spring

Housing

Armature

Coil

Electrical
Connections

Schematic Diagram

Figure 8-30 Solenoid-actuated directional control valve

Solenoid-actuated directional control valves are available in a variety of configurations, including 3-position spools with dual solenoids and spring centering. The major benefit of a solenoid-actuated directional control valve in programmable automation applications is that the solenoid easily interfaces with the controller. The air is then used to amplify the control signal to perform the actual work.

Pneumatic actuation systems are readily available and have been in use in automation for some time. The components are easy to implement and relatively inexpensive. Air leakage causes only minor performance problems and no environmental concerns. However, speed control is limited to flow control valves, and position control is limited to

employing hard stops. Hence, pneumatic systems cannot provide precise speed and position control and have a tendency to be noisy. The banging of the pneumatic devices off hard stops accounts for another name for them: "bang-bang systems."

8.6.2.2 Hydraulic actuators

Hydraulic actuators work in much the same manner and have many of the same components as pneumatic systems, including solenoid-actuated directional control valves. However, there are some major differences, most important of which is that the system uses an oil or oil-over-water liquid instead of air to amplify the control signal. This offers some major advantages over pneumatic systems, including a much higher operating pressure (3000 psi versus 90 psi) and, because of the incompressibility of the hydraulic fluid, position control. Thus, position, speed, and acceleration can be accurately controlled with proportional servo-valves. Thus, cylinders, for example, can be stopped in midstroke, whereas in pneumatic systems the cylinder can only be fully extended or fully retracted. There are many disadvantages to consider in the use of hydraulic actuators, including that fluid must move in a complete circuit—i.e., returned to the tank and not vented to the atmosphere, unlike what is done in pneumatics. This consideration requires much more piping that used in pneumatics. Additionally the most modern facilities no longer use centralized hydraulic supply systems. Thus, each machine or work cell need a self-contained hydraulic system. These units are noisy and prone to leaks. The higher load capacity of hydraulic systems also makes them more dangerous to the operator. Despite these concerns, hydraulic systems are commonly used in programmable automation systems primarily because of the higher load capacity and the ability to provide position, speed, and acceleration control of the actuators.

8.6.2.1 Electrical actuators

There are numerous electrical actuators used in automation control. A very simple and important type is the *electrical relay*. An electrical relay is nothing more than a switch that is actuated electronically to engage a set of electrical contacts. It works similar to a solenoid in that it uses a coil to magnetize a metallic core. The core then attracts a lever causing electrical contacts to close. This is demonstrated schematically in Figure 8-31. Relays are vital in control applications because electrical actuators typically use much higher current than a controller can provide. Thus, the relay provides a means to amplify the control signal. Relays are used in conjunction with other electrical actuators, including *electrical motors* and *electrical heaters*.

Figure 8-31 Relay

There are several types of electrical motors: *induction*, *servo*, and *stepper*. Induction (alternating current (AC)) motors are rotational electromagnetic motors, which are commonly seen in industrial applications, since they are low cost and simply constructed. This makes them as quite reliable and they need only low maintenance. Induction motors are commonly used in constant torque and fixed speed applications. However, a frequency controller can be used in conjunction with the motor to control rotational speed and acceleration. Induction motors can be brought to an abrupt stop with dynamic braking.

If high starting torque, precise positioning, speed, and acceleration are required then a *servomotor* should be considered. Servomotors are used in many automation applications. These direct current (DC) motors can rotate to, and hold, a fixed angular position. Speed and acceleration can be accurately controlled with a DC servo controller, which is often necessary in control applications.

A *stepper motor* takes electrical pulses from the pulse generator of the stepper motor controller and rotates the output shaft at rpm in direct proportion to the number of pulses received. Stepper motors have very high accuracy and are commonly used in open-loop control applications. Disadvantages as compared to DC servomotors include lower torque and limited speed. Any type of motor can be and often is used with either a rack-and-pinion or ball-screw system to provide linear motion.

Electrical heaters, another common type of electrical actuator, are resistors that generate heat upon exposure to current. They generate heat in industrial applications and come in various configurations, including *cartridge* and *band*. Cartridge heater resistance wire is coiled in tubes; band heater resistance wire is wound in flat strips. Examples of cartridge and band heaters are shown in Figure 8-32. Common applications have electrical heaters used in conjunction with temperature sensors and temperature controllers.

Cartridge Heater   Band Heater

Figure 8-32 Cartridge and band heaters

Other types of simple electrical actuators include *solenoids* and *torque motors*. As mentioned previously, solenoids utilize an electromagnet to push or pull a ferrous plunger. They generate only light loads, but when combined with springs and mechanical linkages they are effective light duty actuators. Torque motors generate a rotational force without continuous motion; their solenoids rotate an armature about an axis.

As has been demonstrated, in many cases the actuator used is not an individual component but an actuation system consisting of many components. In some cases the actuator is a nested control system, as in the case of a robot or CNC machine. By themselves, they are independent entities with their own sensors, actuators, and control system. However, at the coordinated system level of control these independent machines are simply viewed as powerful nested discrete actuators to the top-level controller.

The type of actuator chosen for a given application depends on many factors, including type of actuation needed, availability, accuracy, reliability, and cost. Figure 8-33 gives a comparison of the advantages and disadvantages of some common actuators used in industrial automation applications.

Actuator System	Advantages	Disadvantages
Pneumatic Actuator Systems	Readily available and inexpensive	Noisy, limited position and speed control, moderate load capabilities
Hydraulic Actuator Systems	High load capacity, precise position, speed and acceleration control.	Noisy self-contained hydraulic system, leaks and costly.
AC Induction Motors	Readily available, and inexpensive Speed control available with frequency controller.	Limited position control, low starting torque.
Electrical Servomotors	Precise position, speed and acceleration control. High load capability.	Expensive.
Stepper Motors	Precise position, speed and acceleration control. Low cost	Low load capability, slower than servomotors.

Figure 8-33 Comparison of actuators

The type and number of sensors and actuators used in a process dictate the type and number of inputs and outputs that should dictate selection of a PLC. This idea is explored in more detail in the next section, as are other PLC considerations.

8.7 Implementing Automation with PLCs

As discussed previously, PLCs were developed specifically for procedural machine and coordinated system control. Thus, they employ discrete sequential process control of machines, workstations, and work cells. The system is controlled per the program of instructions, which is the sequential list of actions necessary to complete the process. Recall the USA principle of automation implementation (see Chapter 2). USA is used in PLC automation. It reminds the operator to **u**nderstand, **s**implify, and **a**utomate a process.

Consider the typical washing machine process. The program of instructions or process steps for such a process is:
- fill washtub with water
- agitate clothes to wash
- drain tub of dirty soapy water

- refill tub to rinse residual dirt and soap from clothes
- agitate clothes to rinse thoroughly
- drain tub again
- spin tub to remove excess water.

Following step 1 of the USA principle, we gain an understanding of the process. Most people are familiar with washing machines from a *user* point of view. However, for implementing automation, one must understand the process from the controller's perspective. To effectively control the machine, one must break the process steps down into a work cycle program. When a designer develops a work cycle program, he or she must put in place appropriate sensors to monitor the process variables (PLC inputs) and select actuators (that act on the process) (PLC outputs). The work cycle program for the washing machine process is shown in Figure 8-34. Clearly, it is a more detailed listing of the process steps listed above. It can also be referred to as an *I/O status table*; in it, process steps are linked to the corresponding status of PLC inputs and outputs.

Process Steps	PLC Inputs (Sensors)				PLC Outputs (Actuators)			
	Start Button	Stop Button	Tub Empty Sensor	Tub Full Sensor	Fill Solenoid Valve	Agitator Motor	Drain Pump	Spin Motor
Start Cycle	ON	off	ON	off	off	off	off	off
Fill tub to wash	off	off	ON	off	ON	off	off	off
Wash (Agitate tub)	off	off	off	ON	off	ON	off	off
Drain	off	off	off	off	off	off	ON	off
Fill tub for rinse	off	off	ON	off	ON	off	off	off
Rinse (Agitate tub)	off	off	off	ON	off	off	off	off
Drain	off	off	off	off	off	off	ON	off
Spin	off	off	ON	off	off	off	off	ON
End Cycle	off	off	ON	off	off	off	off	off

Figure 8-34 Washing machine work cycle program (or I/O status table)

Notice that this work cycle program shows the status of the inputs and outputs at the *start* of each process step. For example, in the "Fill tub to wash" step, the "Tub Empty Sensor" is marked "ON." Once the tub begins to fill the sensor is switched to "off." The work cycle diagram is a valuable aid in the PLC programming process because it defines the relationship between the process step and I/O status.

A *timing diagram* may also be a beneficial aid when a process is particularly complicated or confusing. In a timing diagram, the columns and rows of the work cycle diagram are reversed, with process steps across the top and PLC inputs and outputs down the left hand side. The duration of each step's I/O status is visually represented in the length of the bars running under each step of the program. The corresponding timing diagram for the work cycle program listed in Figure 8-34 is shown in Figure 8-35.

For some programmers the timing diagram provides a clearer picture of the status of the inputs and outputs of the PLC as the process is executed. Typically, both work cycle program and timing diagram are used to develop the control program, along with other tools, such as process flow charts and state diagrams. All these are crucial to a programmer's clear understanding of the process and the corresponding logic necessary for proper execution.

The next step in the USA principle is *simplifying* the process as much as possible. For those familiar with a washing machine cycle, it is obvious that the above work cycle program is already very simple. Options for selecting different wash cycles or water levels are omitted. While these are important options, it is often desirable to oversimplify the process at first by leaving out even necessary options, which can then be added after the basic work cycle program is fully implemented. This makes the final step—*automating* the process—even easier.

1. Automating the process involves:
2. Identifying and procuring an appropriate PLC for the application.
3. Designing and developing the control system hardware, including the control panel, electrical cabinet(s), and sensor and actuator mounting brackets.
4. Procuring the sensors and actuators as necessary.
5. Installing the hardware, and wiring the sensors and actuators to the PLC.
6. Converting the work cycle program into the PLC program of instructions.
7. Simulating the program to verify program integrity.
8. Entering the program in the PLC and verifying the program on the machine or workstation.

Item 1 above will be discussed in the next section. Items 2 through 4 are important. They may be handled internally by a firm's engineering and maintenance departments if their level of expertise is adequate, or they may be subcontracted to other firms who specialize in these types of activities. These items are outside the scope of this text, but items 5 through 7 are covered in detail in Chapter 11.

Process Steps:		Start cycle	Fill tub for wash	Wash (Agitate tub)	Drain	Fill tub for rinse	Rinse (Agitate tub)	Drain	Spin	End Cycle
PLC Inputs (Sensors)	Start Button	On / Off									
	Stop Button	On / Off									
	Tub Empty	On / Off									
	Tub Full Sensor	On / Off									
PLC Outputs (Actuators)	Fill Valve	On / Off									
	Agitator Motor	On / Off									
	Drain Pump	On / Off									
	Spin Motor	On / Off									

Figure 8-35 Washing machine timing diagram

8.7.1 PLC Selection Considerations

It is vitally important to select the PLC correct to the application at hand. Many brands of PLCs are available, and within each brand are many models with a wide range of capabilities and prices. For optimum control of the process the PLC needs to be properly sized. Most manufacturing facilities have a favorite brand of PLC and tend to stick with that brand for new automation projects. This favoritism develops over time as the maintenance and engineering departments become familiar and adept with the brand, accumulating all the necessary programming software and hardware for the PLC of choice. Personnel become well-acquainted with this PLC's instruction set, so opportunities to introduce a new brand may be limited. Regardless of the brand used, the correct model *within* the brand must be specified. The four major categories to be considered for PLC model selection include:

- I/O configuration
- CPU performance
- communications
- programming.

I/O configuration refers to the quantity and type of inputs and outputs an application requires. The quantity of the inputs and outputs is critical as it is the primary differentiating factor between PLC model types. For example, an inexpensive PLC model may only have 14 I/O (8 inputs and 6 outputs) whereas a more expensive model may have 36 I/O (20 inputs and 16 outputs). Note, however, that many models have expandable I/O where both inputs and outputs can be added with the addition of expansion boards. This is an important consideration if the control requirements of a certain process have the potential to increase over time.

The other very important I/O consideration is the type of inputs and output required. Recall from Section 8.4 there are many different ways in which the inputs and outputs can be configured, including discrete 120-volt AC, 24-volt DC, relay type, or even continuous analog I/O. Additionally, I/O boards can be mounted locally with the PLC or placed in a remote location closer to the actual process. The configuration selected must match the application requirements as outlined in the work cycle program with some room for expansion. Often during the actual installation and startup of the application, the need for additional inputs and/or outputs is identified. Therefore, it is good that one select a PLC capable of expansion or with slightly more I/O than is initially needed to support future process requirements.

CPU performance refers to the capabilities of the central processing unit of the PLC. The amount of memory, contact execution time, scan time, and the size of the instruction set all fall into this category. Size of the instruction set refers to the number of programming commands available. As will be seen in the next chapter there is a basic set of programming instructions that all PLCs have. However, different PLC brands and models have expanded instruction sets to provide more capabilities. Executing

subroutines—for/next loops, table functions, or even trigonometric calculations—are available. The number of these capabilities is directly proportional to the price of the unit. In general, greater memory, speed, and program instructions all increase the cost of the PLC.

With the ever-increasing need for an integrated facility, individual machines and workstations need *communications* capability occurring in real time so they may share data and information. Thus, the application may dictate that the selected PLC needs to communicate with other PLCs, computers, or I/O devices. Types of communications available include ASCII, point-to-point communications between two PLCs, and Ethernet communications.

Finally, PLC *programming* is a prime consideration. The most widely used programming method for PLCs is a graphical language called *ladder logic programming*, accomplished by inserting program instructions into a rung of a ladder diagram. Ladder diagrams were originally used to specify the logic and wiring of older, hardwired relay-type control systems. Ladder diagrams were widely used when PLCs were first introduced into production, so programming of PLCs was modeled after ladder logic to ease the transition from hardwiring to PLC. Other graphical methods of programming PLCs are available, as are some text-based methods. Ladder logic programming is addressed in detail in Chapter 9.

Another programming consideration is the programming device. Most modern PLCs are programmed with a laptop or personal computer. Each PLC brand typically requires proprietary software for programming. This software can be expensive to purchase and license. However, once obtained it can be used to program any number of PLCs that it supports. Thus, when one considers different PLC brands, the cost and availability of the programming software is often a major consideration.

The overall goal is to select a PLC that adequately meets the application requirements with some room for expansion. If the process to be controlled is simple and straightforward, a low-end model should be adequate. If an application's future growth is unpredictable, a scalable higher end model may be required.

Once the appropriate PLC is selected, control system hardware can be designed. Then, the sensors and actuators can be procured and all hardware installed. After the work cycle program and/or timing diagram is prepared, the system will be ready for simulation and actual programming. This is covered in great detail in the next chapter.

8.8 Summary

Programmable logic control (PLC) technology is used to impart automatic control over tasks or events through the use of electrical and computer technology. It is a standardized computer system specifically designed for interfacing with industrial components and equipment. In order to provide control over the process it must collect information from a process, upon which it bases its decisions, and then implement those

decisions by acting on the process. This is accomplished by interfacing with the process through sensors and actuators.

The sensors measure process variables that provide information about the state of the process. Actuators receive information from the controller to act on the process to achieve the set points or targets for the process. These targets are called process parameters. Thus, control of the process is achieved by comparing the process variables against the process parameters and acting on the process accordingly.

Process variables can be continuous or discrete. Continuous process variables can have more than one value, whereas discrete variables can only have one non-zero value. When process variables are discrete the controller exercises discrete process control; when continuous, continuous process control is exercised.

Whether a process is continuous or discrete, the controller may employ one of two methods to control it: closed loop control or open loop control. In closed loop control the process variable is continuously compared to the process parameter by way of a feedback loop. In open loop control the process is controlled without monitoring any process variables. Level of control influences the type of control utilized. Basic device level control is used to control devices or components of a larger process. Procedural machine control is typical of machine and workstation level control applications. Coordinated system control is the highest level and occurs when sensors, actuators, devices, and individual machines are controlled and coordinated to perform processing.

PLCs were developed specifically for procedural machine and coordinated system control. Thus, they exercise what is called discrete sequential process control in which the parameters and variables of the system are changed at discrete moments in time. An event-driven change occurs when the PLC executes a response to some event that has caused the state of the system being controlled to change. A time-driven change occurs when the PLC executes a response at either a certain time in the process or after a specific time lapse has occurred.

A PLC is a microprocessor-based discrete process controller that uses stored instructions in programmable memory to implement logic, sequencing, and math control functions for procedural machine and/or coordinated system control of machines and processes. The stored instruction set that is programmed into a PLC is called the work cycle program.

PLC hardware includes processor, memory unit, power supply, I/O module, and programming device. The processor examines the status of the input signals, executes the logic and sequencing functions, and operates on the outputs. The memory unit stores the work cycle program, I/O status information, and controller system operation information. The power supply provides the power to the unit. The controller physically connects with the sensors and actuators through the I/O module. The programming device provides a means of entering the work cycle program into the memory module of the PLC.

PLCs have many advantages over hardwired control systems including easier programmability, reprogrammability, small electrical control cabinets, easier

maintenance, more convenient interfacing with other equipment, and the capacity to perform many different control functions.

Sensors provide a means by which the controller interfaces with the process. Sensors can be separated into three major categories; switches, transducers, and special purpose sensors. Switches measure discrete process variables; numerous types are available including pushbuttons, toggle switches, and limit switches. Transducers measure continuous process variables and can be either analog or digital. Special purpose sensors consist of self-contained devices having sensors and a controller specifically designed to interface with the PLC. (A vision system is an example of a special purpose sensor.)

Actuators act on a process so that process variables are brought into agreement with process parameters. They are classified according to the type of control signal amplification used; these classifications include pneumatic, hydraulic, and electrical. Pneumatic actuators include cylinders, air motors, and vacuum cups. Hydraulic actuators work in much the same manner and have many of the same components as pneumatic systems, including solenoid-actuated directional control valves. However, hydraulic systems can generate much greater operating pressure and better positional control. Electrical actuators include relays, electric motors, electric heaters, solenoids, and torque motors.

The USA principle of "understand, simplify, and automate" is the preferred way to approach automating processes. Conversion of a program of instructions to a work cycle program helps give an operator a thorough understanding of the process and enables simplification. A program of instructions lists the steps necessary to complete the process. The work cycle program is a more detailed listing of the process steps linked to the corresponding status of PLC inputs and outputs. A timing diagram also links the process steps to the status of the PLC inputs and outputs; with it, the status of each of the inputs and outputs can be tracked as a process moves through each step of the work cycle program. The first task of the automating step of the USA principle is selection of appropriate PLC for the process at hand.

The four major items for PLC model selection are I/O configuration, CPU performance, communications, and programming. Once the appropriate PLC is selected, the control system hardware can be designed and the sensors and actuators procured and installed. At this point the system is ready for programming. PLC programming is the topic of the next chapter.

8.9 Key Words

basic device control
capacitive proximity switches
closed loop control
contact limit switches
continuous process control
continuous process parameter
continuous process variable
control panel
coordinated system control
discrete process control
discrete process parameter
discrete process variables
discrete sequential process control
electrical actuators
electrical relay
event-driven change
hardwiring
hydraulic actuators
inductive proximity switches
input scan
limit switch

non-contact limit switches
open loop control
operating cycle
optical encoder
optical proximity switches
output scan
pneumatic actuators
procedural machine control
process parameters
process variables
program scan
programmable logic control(ler) (PLC)
pushbuttons
scan time
solenoid actuated directional control valves
special purpose sensors
switches
time-driven change
timing diagram
toggle switches
transducers
USA principle
work cycle program

8.10 Review Questions

1. Discuss how a controller interfaces with a process.
2. Define process variables and process parameters.
3. Explain the difference between discrete and continuous process control.
4. List the three levels of process control and discuss each.
5. Explain the difference between event-driven and time-driven change.
6. Define a work cycle program and discuss how it is created.
7. Is the operating cycle of a PLC instantaneous? Explain.
8. List and discuss each of the hardware components of a PLC.
9. Discuss the three major types of sensors.
10. Explain the differences between contact and non-contact sensors. Give examples of each.

11. Discuss the different classifications of actuators. Which classification is best for high load and precise position control applications? Explain.
12. Explain the USA principle's role in automating processes.
13. Explain the differences and similarities between a work cycle program and a timing diagram.
14. List and discuss the seven steps required to automate a process.
15. List and discuss the four major considerations for selecting a PLC.

8.11 Bibliography

1. Groover, M.P., 2001, *Automation, Production Systems and Computer-Integrated Manufacturing,* Second Edition, Prentice Hall, Upper Saddle River, New Jersey.
2. Derby, D., *Design of Automatic Machinery,* Marcel-Dekker, New York, NY 2005. Morriss, S.B.,1995, *Automated Manufacturing Systems*, Glencoe/McGraw-Hill, Columbus, Ohio.

Chapter 9

Programming PLCs

Contents

Objective

The objective of this chapter is to explain ladder logic programming and demonstrate the programming process for PLCs.

419

9.1 Programming Concepts

The intent of this chapter is to instruct the reader on the programming of a PLC. By "programming" we mean the act of issuing the PLC instructions or commands that will execute the manufacturing tasks or steps of a process. If we assume we use the USA principle, programming a PLC happens during the "automate" step. Consequently, the following assumptions apply:

(i)   A firm understanding of the process exists.
(ii) The process is as simplified as possible.
(iii) The process steps have been broken down into a work cycle program and timing diagram (Chapter 8).
(iv) The PLC along with all sensors and actuators have been selected, installed, and wired.

For example, consider the washing machine process of Chapter 8 and depicted in Figure 9-0. The figure is a cutaway view of a typical washing machine with process steps labeled. The figure is anatomically correct, but slightly modified for readability. For instance, the fill solenoid is shown outside the machine. Figures 9-1 and 9-2 are replicated from Chapter 8, showing the work cycle program and timing diagram, respectively.

Figure 9-0 Cutaway view of washing machine

Process Steps	PLC Inputs (Sensors)				PLC Outputs (Actuators)			
	Start button	Stop button	Tub empty sensor	Tub full sensor	Fill solenoid valve	Agitator motor	Drain pump	Spin motor
Start cycle	ON	off	ON	off	off	off	off	off
Fill tub to wash	ON	off	ON	off	ON	off	off	off
Wash (agitate tub)	ON	off	off	ON	off	ON	off	off
Drain	ON	off	off	off	off	off	ON	off
Fill tub for rinse	ON	off	ON	off	ON	off	off	off
Rinse (agitate tub)	ON	off	off	ON	off	ON	off	off
Drain	ON	off	off	off	off	off	ON	off
Spin	ON	off	ON	off	off	off	ON	ON
End cycle	off	off	ON	off	off	off	off	off

Figure 9-1 Washing machine work cycle program

Process Steps:		Start cycle	Fill tub for wash	Wash (Agitate tub)	Drain	Fill tub for rinse	Rinse (Agitate tub)	Drain	Spin	End Cycle
PLC Inputs (Sensors)										
Start Button	On									
	Off									
Stop Button	On									
	Off									
Tub Empty	On									
	Off									
Tub Full Sensor	On									
	Off									
PLC Outputs (Actuators)										
Fill Valve	On									
	Off									
Agitator Motor	On									
	Off									
Drain Pump	On									
	Off									
Spin Motor	On									
	Off									

Figure 9-2 Washing machine timing diagram

Armed with the information in these two diagrams (I/O status and process steps) it is now possible for us to transform the work cycle program and timing diagram into the language of the PLC. The industry standard language of the PLC is a graphical language called *ladder logic*. It uses graphically represented instructions to convert the work cycle program into a *logic control* program that the PLC can use to control the process. This chapter is dedicated to instructing the reader how to convert the work cycle program and timing diagram into a ladder logic program. In order to accomplish this, we must understand the basics of logic control and sequencing and a have a firm grasp of the basic input and output PLC ladder logic instructions.

9.1.1 Logic Control, Sequencing, and Ladder Logic Diagrams

Logic control is a means to set the status of PLC outputs solely on the status of its inputs. In logic control there is no memory of the historic values of the inputs; the values assigned to the outputs are based on the input's current state. Consider the work cycle program shown in Figure 9-1. The first process step is to press the start button. When the start button is pressed and the empty tank sensor is on, the solenoid valve opens, allowing water to enter the tub, an event that depends exclusively on the status of the start button and empty tank sensor inputs. The history of the state of these inputs is not retained by the PLC or referred to when a decision is made about whether the solenoid valve should be open. Only the current state of the inputs determines if the valve is opened.

In sequencing control, time-driven changes drive the status of the outputs. In many processes certain tasks are executed after a period of time. For instance, in the washing machine work cycle program the agitator motor is started by an event-driven change, the tub being filled, but is stopped after a period of time. The majority of processes require logic and sequencing control for proper task performance. PLC ladder logic programs execute both logic and sequencing control.

Older washing machines used hardwired relay logic (and sequencing) control to execute the work cycle program, so the logic control program consisted of electrical sensors wired directly to relays and timers, which in turn were wired to the electrical actuators. The drawing that indicated how to physically wire the sensors, actuators, relays, and timers was called a *ladder logic diagram*, named this way because it resembles a ladder, with the logic elements (the sensors and actuators) making up the (horizontal) rungs and the power to the elements making up the (vertical) rails of the ladder (Figure 9-3). Figure 9-3(a) shows the physical wiring of two switches and two lights. The lights are used to represent any type of actuator, such as a relay, motor, or solenoid. The first rung light L_1 is off because switch S_1 is open. Conversely the second rung light L_2 is on because switch S_2 is closed. Figure 9-3(b) shows the same switch and light connections as in a PLC ladder logic diagram. Note the ladder nomenclature. With the addition of more logic elements (rungs), the resemblance to a ladder becomes even more apparent.

Figure 9-3 Ladder logic diagram

For Figure 9-3(a), light L_2 of rung 2 is on because electrical continuity exists. Thus, for rung 1, light L_1 will only light when switch S_1 is closed, achieving electrical continuity. In the PLC ladder diagram, rather than being concerned about electrical continuity we care about logic continuity. For the first rung to be true (i.e., output L_1 is lit) there must be logic continuity. If switch S_1 is true (equivalent to on) then L_1 is true (on) and the rung will be true (on). This is the essence of logic programming.

When there is only one input and one output per rung the logic is straightforward. However, when multiple inputs for a single output exist, the logic becomes slightly more complicated. This dictates the use of *logic gates* and corresponding *truth tables* to clarify the logic. Logic gates provide a specific output based on the status of the inputs of that rung. The gates are the switches of the rung. The truth tables shown here list all the I/O scenarios of gates (or switches); output is designated, per the truth table nomenclature, as TRUE (ON). Indeed, from this point, all words that are all capitalized refer to truth table values or operators. Three basic logic gates commonly used in PLC programming are the AND, OR, and NOT gates. The electrical circuits for each of the gates are shown in Figure 9-4. Figure 9-5 shows the truth table for each gate.

Figure 9-4 Logic gates

AND Gate				OR Gate				NOT Gate	
Inputs		Output		Inputs		Output		Input	Output
X_1	X_2	Y		X_1	X_2	Y		X_1	Y
0	0	0		0	0	0		0	1
0	1	0		0	1	1		1	0
1	0	0		1	0	1			
1	1	1		1	1	1			

Figure 9-5 Logic gate truth tables

Since we are exercising discrete process control with discrete inputs and outputs, the only possible I/O values are ON or OFF. These are binary states in which 0 represents FALSE (equivalent to OFF) and 1 represents TRUE (ON). Accordingly, truth tables typically list the I/O status in terms of 0 or 1.

In Figure 9-5 the AND gate is TRUE when both inputs are TRUE. The OR gate is TRUE if either or both inputs are TRUE. The NOT gate has only one input, and outputs are the opposite state of the input. The NOT gate (Figure 9-4, far right), is able to output the opposite of the input through the principle of least resistance. When switch X_1 is open, electrical continuity through the light is achieved and the light is lit. However, when X_1 is closed, the electricity has two paths to follow—either through the light or the switch. Since the switch will have lower resistance than the light, the electricity follows the path of least resistance and flows through the switch instead of the light. Therefore, the light is NOT lit when switch X_1 is closed.

The PLC ladder logic diagram for the AND and OR gates are very similar to Figure 9-5. The NOT gate, however, is shown very differently in a PLC ladder logic diagram. This is because there is an alternative and perhaps more conventional way to represent the NOT gate than is shown in Figure 9-4. The alternative method makes use of a normally closed (NC) contact. Recall from Chapter 8 that normally closed contacts are closed until the switch is "made" or pushed. The pushing of the switch causes the contacts to open. This is illustrated in Figure 9-6.

Figure 9-6 Alternative circuit for the NOT gate

If it is specified that the normally closed switch is OFF when it is not pushed and ON when it is pushed, the truth table will be as shown in Figure 9-5. Accordingly, the PLC ladder logic diagram for the AND, OR, and NOT gates is shown in Figure 9-7. Note how, for the NOT gate, the X_1 input is represented with a normally closed symbol. All other inputs are shown with the normally open symbol.

Figure 9-7 PLC ladder logic diagram for the AND, OR and NOT gates

Note the branching in the OR gate rung. This is how multiple inputs are combined in one rung. In general, there can be multiple inputs per rung, but only one output per rung. Additionally, an output (Y) can be represented on the same rung and/or on a different rung as an input.

NAND Gate		
Inputs		Output
X_1	X_2	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate		
Inputs		Output
X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 9-8 Truth tables for the NAND and NOR gates

Two additional logic gates, the NAND and the NOR, can be created by combining the three basic gates shown in Figure 9-7. The NAND gate is created by combining the AND gate and the NOT gate in sequence. Similarly, the NOR gate is created by pairing the OR gate and NOT gate in sequence. The truth tables are shown in Figure 9-8.

We can easily discern when we compare these tables to the truth tables of Figure 9-5 that the NAND gate is opposite the AND gate, and the NOR gate is opposite the OR gate. In order to represent the NAND and NOR gates in a PLC ladder logic diagram it is necessary we use two sequential rungs. This is shown in Figure 9-9.

Figure 9-9 PLC ladder logic for the NAND and NOR gates

The NAND gate has one rung for the AND gate followed by a NOT gate rung. The output from the AND rung (Y_1) serves as the input in the NOT rung. Note that the output of interest is the output from the NOT rung (Y). The NOR gate uses the same approach.

This methodology (i.e., combining the basic logic gates within and among multiple rungs of a ladder logic diagram) is the essence of how the work cycle program gets translated, converted, or programmed into the final ladder logic diagram program that directs the PLC in control of the process. For complicated control applications, this task may appear monumental. However, with some effective programming aids, the

programming process can be greatly simplified. Accordingly, the remainder of this chapter will be devoted to instructing the reader on PLC ladder logic programming.

The next two sections address ladder logic terminology and the basic ladder logic instruction set. Subsequent sections provide a detailed programming procedure, simulation information, and programming examples.

9.2 Ladder Logic Terminology

Figure 9-10 again shows the PLC ladder logic for the NAND gate. The logic instructions on the rungs of the ladder are separated into *condition instructions* and *output instructions*. Condition instructions examine the status of the inputs; output instructions energize the outputs.

Figure 9-10 PLC ladder logic for the NAND gate

The variable listed under the instruction is the *instruction address*. For the first condition instruction on rung 1, the X_1 variable is the address of the instruction. Depending on the PLC brand, these addresses may be numbers, letters, or some combination. The instruction address typically has three functions: It provides a name for the instruction, specifies the location of the *status bit* of the instruction in the *data table*, and specifies the function of the instruction.

The instruction may reference either a real external physical sensor or an actuator wired to the PLC or an "artificial" internal logic element of the PLC. External sensors and actuators are wired to the PLC input and output terminals, respectively. An example of the external sensor and actuator wiring is shown in Figure 9-11. Note that actual wiring is dependent on type of sensor or actuator and PLC terminal available. *PLC documentation must always be consulted for specific wiring information so dangerous and damaging wiring mistakes are avoided.* Input terminals correspond with address of the condition instructions listed in the PLC ladder logic diagram.

Internal logic elements act the same way as real sensors and actuators; however, they exist only in the ladder logic program. They are used to create the appropriate conditions to complete the work cycle program logic. For example, the Y_1 output instruction in rung 1 top rung of Figure 9-10 is an internal logic element. Correspondingly, it is an internal condition instruction in rung 2. It only exists in the ladder logic program as a means to complete the logic of the NAND gate.

Figure 9-11 External input and output wiring example

The data table mentioned above is the portion of the PLC's memory wherein the status of all the inputs and outputs is stored. This status is stored as a bit. This "status bit" has value 0 for OFF (FALSE) and 1 for ON (TRUE).

The memory storage unit for an instruction is called a *word*. Typically, one instruction takes up one word of memory. Thus, PLC manufacturers often specify the amount of PLC memory available in terms of words.

There are only two types of condition instructions: *Examine_ON* and *Examine_OFF*. An Examine_ON condition instruction causes the PLC to examine the status bit at the address specified for an ON condition. Thus, when the bit is 1 or ON, the condition instruction is TRUE. The condition instructions of rung 1 of Figure 9-10 are Examine_ON instructions. When the status bit of the other type of condition instruction— Examine_OFF—is 0 or OFF, the condition instruction is TRUE.

As we show in the next section, there are many types of functions an output instruction can perform. Some are just relays that close electrical contacts to energize an actuator; others act as timers, counters, or perform some other advanced function. However, in order for a function to be executed the preceding condition instructions must be TRUE. If we have this logic continuity the PLC will set the output instruction's status bit for the address specified to 1, thereby turning it ON or making it TRUE. When the output instruction is TRUE it will perform its function.

The output instructions shown in Figure 9-10 are called *Output Energize* instructions. Output Energize instructions are the simplest output instructions. They function as relays, thus they close their contacts when energized. Additional output instructions will be discussed in the next section.

Now let us consider the PLC ladder logic program of Figure 9-10 in terms of how the PLC will evaluate it. Recall from Chapter 8 that a PLC's operating cycle consists of three scans. The PLC first performs an *input scan*, in which it reads and stores the status of the inputs (0 or 1). The next scan is the *program scan*, in which it evaluates the logic of the program. The final scan is an *output scan*, in which the outputs are updated. Thus, during the input scan of the NAND gate, the PLC will examine X_1, X_2, and Y_1 and store the status in the appropriately addressed status bit in the data table. During the subsequent program scan, the PLC will examine the status bits according to the ladder logic program and determine the corresponding status of the outputs and place that data in the status bits of the data table. During the last scan, the output scan, the PLC will transfer the status bit information to the corresponding output terminal. Older PLCs could complete this cycle about 67 times per second. Modern PLCs have operating cycles in the range of 1.5 milliseconds to 3.0 milliseconds (300+ times per second). The three scans are depicted graphically in Figure 9-12.

Figure 9-12 PLC operating cycle

Armed with the information presented in this section, the reader is encouraged to review the truth tables of each of the logic gates presented and verify the results therein.

9.3 Typical PLC Instruction Set

All of the condition and output instructions offered by the PLC and available to the programmer for converting the work cycle program into the PLC ladder logic program are collectively called the PLC's *instruction set*. Most modern PLCs have a rather extensive instruction set. In addition to five basic instructions they include instructions for subroutines, for/next loops, table functions, and analog PID functions, to name a few. Additionally, the number and type of advanced instructions available is dependent on the PLC brand and model. However, for most discrete process control applications an effective conversion of the work cycle program into a PLC ladder logic program can be achieved with a thorough understanding of just five basic instructions. All other instructions in the PLC's instruction set are derived from the operating concepts of these five. Thus, the ability to use the more advanced instructions will be much easier once an understanding of the basic five instructions is achieved. Therefore, the focus of this section is to provide the reader with a solid grasp of these five basic instructions. These instructions are referred to as the *basic PLC instruction set* because, at a minimum, all PLCs have these instructions available.

The basic PLC instruction set is shown in Figure 9-13. It contains the relay type instructions, Examine_ON, Examine_OFF, and Output Energize discussed in the last section. These instructions are for the event-driven changes that occur in the process being controlled. The last two output instructions, *timer* and *counter*, provide the PLC with sequencing control. The symbols and programming formats of these two instructions may vary among different PLC brands. However, the use and operation of the instructions will be essentially the same as we present them here.

The timer instruction provides a means for the PLC to issue a time delay before proceeding to the next operation of the work cycle program. Again consider the washing machine work cycle program of Figure 9-1. A timer would be used to control the time delay between the start and stop "Wash (agitate tub)" steps. Once the "Wash (agitate tub)" step is started, the timer would control the length of time the agitator motor runs. Timers start functioning, that is to say, timing, at the point where preceding condition instructions on the rung are TRUE. Once the set time is reached the instruction becomes TRUE. Timers are reset either by the preceding condition instructions becoming FALSE or by a separate instruction. How one is reset depends on the PLC brand. For the purposes of this text, our timer instruction will be reset when the preceding condition instruction becomes FALSE.

Symbol	Instruction Name	Instruction Type	Logic Description
⊣⊢	Examine ON	Condition	A status bit of "1" or "ON" at the specified address will cause this instruction to be TRUE
⊣/⊢	Examine OFF	Condition	A status bit of "0" or "OFF" at the specified address will cause this instruction to be TRUE
⊣)⊢	Output Energize	Output	When the logic continuity of the rung is achieved, this instruction will energize or become TRUE
⊣(TMR)⊢	Timer	Output	When the logic continuity of the rung is achieved, this instruction will energize or become TRUE after a user specified *time delay*.
⊣(CNT)⊢	Counter	Output	This instruction counts the number of times logic continuity of the rung changes. When a user specified *count is reached* the instruction will energize or become TRUE.

Figure 9-13 Basic PLC instruction set

Counters provide the PLC with a means of counting the number of times the preceding condition instructions change state from FALSE to TRUE. This is called a *logic pulse train*. When the count reaches a preset value, the instruction becomes TRUE. Counters require a separate instruction for resetting. For our washing machine cycle example, a counter could be used to control the number of rinse cycles. Assume that one rinse cycle does not adequately remove all the soapy water from the clothes. So, two rinse cycles would be desired. Instead of rewriting the work cycle program shown in Figure 9-1, a counter could be added to the logic so that two rinse cycles occur. Thus, the steps from the start of "Fill tub for rinse" to the stop for "Drain" would be repeated two times.

9.3.1 Relay Output Instruction Example

The Output Energize relay instruction is used whenever it is desirable to turn on an output in response to certain conditions being met. This type of instruction is, necessarily, used extensively in PLC ladder logic programming. Recall that the output can be an actual external device such as a light, motor, solenoid, or an internal logic device. Consider the following example.

Figure 9-14 Motor-starting example

Example 9.1

A PLC is used to control the starting of a motor (Figure 9-14). Note that the PLC is inside the control panel and already wired to the pushbutton inputs and the motor and light outputs. The motor is started when the START pushbutton is pressed. The motor will continue to run once the START button is released. The motor will not stop running until the STOP pushbutton is pressed. The motor will then remain off until the START pushbutton is pressed again. A light on the control panel will go on whenever the motor is running. Develop the work cycle program and corresponding PLC ladder logic program of instructions to control this process.

Solution

The process steps can be summarized as:

- Press the START button to turn ON the motor and light.
- Press the STOP button to turn OFF the motor and light.

An important first step is: *identify and document the inputs and outputs for the process*. The inputs are the START button (labeled S_1) and the STOP button (labeled S_2). The motor is an output labeled M and the light an output labeled L. Accordingly, the work cycle program is shown below in Figure 9-15:

Process Steps	PLC Inputs (Sensors)		PLC Outputs (Actuators)	
	START button (S_1)	STOP button (S_2)	Motor (M)	Light (L)
Press START button	ON	off	ON	ON
Press STOP button	off	ON	off	off

Figure 9-15 Example 9-1 work cycle program

For relatively simple processes a good method for one to use in developing the PLC ladder logic is to write out the input conditions and output status statement for the process in terms of the AND, OR, and NOT logic elements discussed previously. The ladder logic can be taken directly from each statement:

- The motor (M) and light (L) are ON when S_1 is ON AND S_2 is NOT ON OR if they were previously turned ON AND S_2 is NOT ON.
-

Based on the first half of the above statement, there is an AND logic gate between S_1 and S_2. S_1 will use an Examine_ON condition statement; S_2 will be an Examine_OFF condition statement (because of the NOT logic gate in the statement). Since there can be only one output per rung, we will put the motor relay output instruction (M) on the first rung. Since the status of the light is based solely on the status of the motor, we will use the motor status (M) as an internal logic element to turn on the light.

Figure 9-16 Example 9.1 preliminary ladder logic

The preliminary ladder logic is shown in Figure 9-16 is incomplete because it does not include the OR logic mentioned in the condition and output status statement. A branch in rung 1 is required to achieve an OR logic gate and essentially "latch" the motor on once it has started AND S_2 is NOT on. The AND after the OR in the second half of the status statement indicates that you must place the branch prior to the ANDed switch S_2. The branch will use an Examine_ON condition of the motor (M) status. The completed ladder logic program of instructions is shown in Figure 9-17.

Figure 9-17 Completed ladder logic program of instructions for Example 9.1

The first rung of the ladder logic program shown in Figure 9-17 is commonly used in industry to control a motor starter circuit. The reader is encouraged to verify that the logic of this diagram satisfies the process steps and the work cycle program.

9.3.2 Timer Output Instruction Example

The timer output instruction provides a method of executing a time-driven change in the process being controlled. It creates a time delay in the sequence of process steps of the work cycle program. "Timing" begins when condition instruction(s) of the rung are

TRUE. The timer output instruction, however, does not become TRUE until it has reached the preset time delay. It is reset to OFF (FALSE) (for the purposes the PLCs discussed in this text) when the condition instructions change to OFF (FALSE). The use of a timer in a ladder logic program is demonstrated in the following example.

Example 9.2

The ladder logic program in Example 9.1 is to be modified. In addition to turning the motor and light on, the PLC is to automatically shut the motor and light off after 2 minutes (120 seconds). All other logic conditions listed in Example 9.1 still apply. Develop the work cycle program and corresponding PLC ladder logic program to control this process.

Solution

The process steps are listed in the problem description and can be summarized as:
- Press the START button to turn ON the motor and light.
- Let the motor run for 120 seconds and then turn OFF the motor and light.
- Press the STOP button to turn OFF the motor and light.

Again, the inputs are the START button, labeled S_1, and the STOP button, labeled S_2. The motor is an output, labeled M, as is the light, labeled L. A timer output instruction, labeled T, will be added to turn the motor and light off after 120 seconds. The work cycle program is shown below in Figure 9-18:

Process Steps	PLC Inputs (Sensors)		PLC Outputs		
	START button (S_1)	STOP button (S_2)	Motor (M)	Light (L)	Timer (T)
Press START button	ON	off	ON	ON	off
Motor runs and light is on for 120 seconds	off	off	ON	ON	off
After 120 seconds, turn off the motor and light	off	off	off	off	ON
Press STOP button to turn motor and light off at any time	off	ON	off	off	off

Figure 9-18 Example 9.2 work cycle program

The input condition and output status statement developed in Example 9.1 is modified according to what is shown in *italics* below.

The motor (M) and light (L) are ON when S_1 is ON AND S_2 is NOT ON, *AND the timer (T) is NOT ON*, OR if they were previously turned ON AND S_2 is NOT ON *AND the timer (T) is NOT ON.*

The motor (M) and light (L) will turn OFF after being ON for 120 seconds.

Refer to Figure 9-17: The first statement above indicates that an Examine_OFF condition instruction for the timer (T) should be ANDed with the S_1 and S_2 switches. Additionally, another rung will be needed in order to start the timer. This rung will consist of an Examine_ON condition instruction of the motor (M) status. The modified ladder logic program is shown in Figure 9-19.

Figure 9-19 Example 9.2 completed ladder logic diagram

Explanation of the logic shown in Figure 9-19: Assume an initial starting state in which both the motor and light are off. When the operator presses the START switch the Examine_ON input condition S_1 becomes TRUE. Since the STOP button has not been pressed, the Examine_OFF input condition S_2 is TRUE. Additionally, since the timer (T) has not "timed out," the Examine_OFF input condition T is TRUE. Thus, logic continuity exists and the motor (M) will be switched on. When the START button is released, logic continuity still exists because of the OR branch containing the Examine_ON input condition statement (M). The moment the motor is started this input condition becomes TRUE, establishing an alternative logic continuity path, which keeps the motor running. For the next rung, when the motor (M) is switched on in the first rung, the Examine_ON input condition of rung 2 will be TRUE, which establishes logic continuity and turns on the light (L). As soon as the motor is turned on, logic continuity also exists in rung 3 because the Examine_ON condition (M) becomes TRUE. This activates the timer to begin "timing" the 120 seconds. When the timer reaches the 120-second preset value, the timer output condition turns ON or becomes TRUE. This causes the Examine_OFF input condition of rung 1 to change to FALSE, breaking the logic continuity of the rung. This then turns OFF the motor, causing the following chain of events:

- The Examine_ON input condition of the OR branch of rung 1 turns OFF or goes FALSE, thereby losing an alternative logic continuity path.
- Rung 2 loses logic continuity and the light goes off.
- Rung 3 loses logic continuity and the timer is reset to off.
- After this sequence, everything is turned off. The only way to restart the motor is to press the START button and repeat the chain of events.

9.3.3 Counter and Reset Output Instruction Example

As the name implies, the counter output instruction allows a count of the number of times a preceding condition instruction changes from FALSE to TRUE. The counter output instruction becomes TRUE, or ON, once the accumulated count value reaches a preset value. This gives the PLC the ability to count the number of times an event-driven change occurs. Since the counting is accomplished on one rung, a separate rung and instruction is needed to reset the counter back to zero. This additional output instruction is called the *reset output instruction*.

The reset output instruction, along with a counter output instruction, is shown in Figure 9-20. Its function is to reset the counter by changing the counter output instruction to FALSE and resetting the count to the start value (typically 0). This reset occurs when the reset output instruction goes TRUE.

Figure 9-20 Counter and reset output instructions ladder diagram

The ladder logic program in Figure 9-20 counts the number of times the Examine_ON condition instruction with the address of X changes from FALSE to TRUE. Once the address X has changed 10 times, the counter output instruction C turns ON, or becomes TRUE. Once the counter output instruction becomes TRUE, it stays TRUE until reset by the reset output instruction. To make the counter output instruction FALSE the Examine_ON condition instruction Y must become TRUE, which, because of logic continuity, makes the reset output instruction TRUE, thereby turning OFF, or making FALSE, the counter output instruction C. For the counter to begin counting again, the Y Examine_ON condition instruction must be turned OFF or made FALSE.

The reset output instruction must have the same address as the counter instruction. This is necessary because there may be more than one counter in a program; it is important

the intended counter be reset. The use of a counter and reset output instruction is illustrated in the next example.

Example 9.3

Consider the application shown in Figure 9-21. An electric motor is being used to drive a conveyor that moves and dumps a product into a bin. When 100 products are dumped into the bin, the conveyor automatically shuts off. An optical proximity switch (O_1) is used to count the product. A PLC located inside the control panel controls the process. Develop the work cycle program and corresponding PLC ladder logic program to control this process.

Figure 9-21 Example 9.3

Solution

The process steps are listed in the problem description; they can be summarized as:
• Press the START button to turn ON the conveyor motor and control panel light.
• The conveyor runs until 100 parts are dumped in the bin, after which the conveyor and light are turned OFF.
• The STOP button can be pressed at any time to turn OFF the conveyor and light.
• The PLC inputs and corresponding addresses are as follows:
• START button – S_1
• STOP button – S_2
• optical proximity switch – O_1.
• The PLC outputs and corresponding addresses are:
• conveyor motor – M
• control panel light – L
• counter output instruction – C
• reset output instruction – C.

Based on this information the work cycle program can be developed as shown in Figure 9-22 on page 436.

Process Steps	PLC Inputs (Sensors)			PLC Outputs				
	START button (S₁)	STOP button (S₂)	Optical prox switch (O₁)	Conveyor motor (M)	Light (L)	Counter (C)	Counter accumulator	Reset inst. (C)
Press START button	ON	off	off	ON	ON	off	0	ON
1st product passes in front of Optical Switch	off	off	ON	ON	ON	off	1	off
2nd product passes in front of Optical Switch	off	off	ON	ON	ON	off	2	off
3rd product passes in front of Optical Switch	off	off	ON	ON	ON	off	3	off
...	...	...	...	...	...	...	...	...
100th product passes in front of Optical Switch	off	off	ON	ON	ON	off	100	off
Counter has hit the preset value and turns ON shutting off motor and light	off	off	off	off	off	ON	100	off
Press STOP button to turn motor and light off at any time	off	ON	off	off	off	off	off	off

Figure 9-22 Example 9.3

The input condition and output status statements are similar to the previous examples, but with some added verbiage for resetting the counter. The statements are shown below:

The conveyor motor (M) and light (L) are on when S_1 is ON AND the optical prox. switch (O_1) is NOT ON AND S_2 is NOT ON AND the counter (C) is NOT ON, OR if the conveyor motor and light were previously turned ON AND S_2 is NOT ON AND the counter (C) is NOT ON.

The conveyor motor (M) and light (L) will turn OFF after the optical prox. switch (O_1) cycles off and on 100 times.

The counter (C) will be RESET each time S_1 is pressed.

The completed ladder logic program is shown in Figure 9-23. It is assumed that the optical prox. switch is wired such that it is "made" ON (TRUE) when a product has broken the light beam. Thus, the Examine_OFF condition instruction for the optical prox. switch in rung 1 is necessary to prevent the operator from starting the conveyor when a product is blocking the beam. This is done to prevent a miscount of the product. The START switch S1 is the logical choice to reset the counter. Each time the conveyor is started the counter will turn OFF and the count will be reset to zero. The reader is encouraged to verify that the ladder logic program of instructions shown in Figure 9-23 satisfies the process flow.

Figure 9-23 Ladder logic program for Example 9.3

9.4 PLC Programming Process

The PLC programming process involves converting the work cycle program and/or timing diagram of the machine, workstation, or system being controlled into the language of the PLC. As we saw, it is effective for a programmer to write input condition and output status statements in terms of the basic logic gates when developing ladder logic programs.

However, as applications become more complex this method alone can become complicated and cumbersome. Thus, additional programming aids are often needed to simplify the complex logic into basic logic gate terminology. One such programming method found to be effective is that of basing the ladder logic program on the different states or modes of operation of the system being controlled. A thorough discussion of this method is given next.

9.4.1 Using State Diagrams to Develop Ladder Logic Programs

As previously stated, logic control provides a means to set the status of the outputs of the PLC solely on the status of its inputs. When a PLC-controlled machine is performing its intended function(s), the status of its outputs will define its mode, or *state*, of operation. States are modes of operation for which the machine is performing an identifiable activity that has to be initiated and then stopped.

Input status facilitates the *transition* from one state to another. Thus, viewing machine operation in terms of states as defined by the outputs and transitions as defined by the inputs can greatly aid the development of the PLC's ladder logic program. An effective tool for visualizing the process in these terms is a *state diagram*.

A state diagram graphically displays the various states and corresponding transitions between those states of the machine or system. Consider the chemical treatment system shown in Figure 9-24. This system consists of a large tank, a start button, an inlet valve, an outlet valve, a tank empty float switch, and a tank full float switch.

Figure 9-24 Chemical treatment system

The basic process is this: A start button is pressed to fill the tank with chemicals; the chemicals time are allowed to react; the reacted chemicals drain from the tank. Accordingly, the work cycle program is shown in Figure 9-25.

Process Steps	Inputs				Outputs		
	Start button	Full float switch	Empty float switch	Reaction time complete	Inlet valve	Outlet valve	Begin timing
	(ST)	(FFS)	(EFS)	(RT)	(IV)	(OV)	(BT)
Tank empty	0	0	1	0	0	0	0
Fill tank	1	0	0	0	1	0	0
Allow time for chemical reaction	0	1	0	0	0	0	1
Drain tank	0	0	0	1	0	1	0

Figure 9-25 Chemical treatment system work cycle program

The first step in developing a state diagram is identification of the different states of the machine or system. For this example the states are fairly obvious and include an idle state, a fill tank state, a chemical reaction state, and a drain tank state. Most systems will have an idle state. "Idle" is typically the initial state of a system when a process has not started—the waiting period for input to switch, or transition, to another state.

When the different states of the system are not readily discernable, one should look at the work cycle program. By definition each state's output status will be unique. The work cycle program confirms there are four identifiable states, as shown in Figure 9-26.

Figure 9-26 Identifying states of the chemical treatment system

The next step involves identifying the transitions between the states. The transitions can also be identified from the work cycle program as shown in Figure 9-27. Note that transitions occur when inputs change from off (0) to on (1).

Process Steps	Inputs				Outputs		
	Start button	Full float switch	Empty float switch	Reaction time complete	Inlet valve	Outlet valve	Begin timing
	(ST)	(FFS)	(EFS)	(RT)	(IV)	(OV)	(BT)
Tank empty	0	0	1	0	0	0	0
Fill tank	1	0	0	0	1	0	0
Allow time for chemical reaction	0	1	0	0	0	0	1
Drain tank	0	0	0	1	0	1	0

Figure 9-27 Identifying state transitions of the chemical treatment system

Once the states and transitions are identified, the state diagram for the system can be constructed (Figure 9-28). States are shown within the bubbles and transitions are shown as lines between states. Arrows indicate the direction of the transition. For instance, pressing the START button causes the empty state to transition to the fill state. An engaged full float switch causes the fill state to transition to the chemical reaction state. When the chemical reaction timer times-out, the transition from chemical reaction state to drain state occurs. Finally, the tank empty float switch causes the drain state to transition to the tank empty state. The status of the outputs for each of the states is shown in table form in Figure 9-29, which is the *state table* for the system.

Figure 9-28 State diagram for the chemical treatment system

Process States	Output States		
	Inlet valve	Outlet valve	Begin timing
	(IV)	(OV)	(BT)
Idle	0	0	0
Fill tank	1	0	0
Chemical reaction	0	0	1
Drain tank	0	1	0

Figure 9-29 State table for the chemical treatment system

The PLC ladder logic program is developed when the state diagram and corresponding state table are completed. First the *state logic* is written per the state table. Then the *transition logic* is written, again as defined by the state diagram.

State logic is written through the use of internal logic elements to set the output status of each state as follows:
- Define each *active state* as an Examine_ON input condition with a unique "internal" address.
- Create a rung for each output.
- Use the appropriate state Examine_ON input condition to activate the appropriate output. If more than one state activates an output they are to be "ORed" together on that rung.
- For the chemical treatment system four states are defined; however, only three are considered active states. The idle state is not an active one because none of the outputs is ON (TRUE). The three remaining states are assigned the following unique addresses for their Examine_ON input conditions:
- Fl_T; Fill tank state
- Chm_R; Chemical reaction state
- Dr_T; Tank draining state.

These Examine_ON input conditions turn on the appropriate output for each state as defined by the state table (Figure 9-30). Note that for this example only one rung is needed for each state. However, in many applications more than one output will be in the ON condition for each state; hence, often each state has more than one rung in the state logic section.

Figure 9-30 State logic for the chemical treatment system

Now let us confirm that the state logic matches the state table. When the system is in the fill state the Fl_T Examine_ON input condition will be TRUE, which, in turn, turns on the inlet valve. When the system is in the chemical reaction state, the Chm_R Examine_ON input condition is TRUE, activating the timer. Finally, when the system is in the drain state, the Dr_T Examine_ON input condition is TRUE, activating the outlet valve.

In writing state logic it is important that a programmer focus only on turning on or setting the correct outputs for each state. The way each state gets activated or deactivated is determined by the transition logic, which will be addressed next.

The role of the *transition logic* is to provide instruction for a system's movement between each state. In so doing, the transition logic turns on, or activates, the desired state and deactivates all others. Turning on is accomplished through activation of an internal relay with the same address as the Examine_ON input condition of the state logic. The transition ladder logic is created with reference to the state diagram, following the simple rules below:

- Transition lines into the state bubble of the state diagram are Examine_ON input conditions.
- Transition lines out of the state bubble of the state diagram are Examine_OFF input conditions, which are ANDed with the lines into the bubble.
- Multiple transition lines into the state bubble should be ORed.
- Multiple transition lines out of the state bubble that lead to another state require a counter to differentiate the logic paths.

Referring to Figure 9-28 and following these rules, we develop the transition logic for the chemical treatment system. We start with the fill tank state. It has one line into it and one out. So, the transition logic for this state can be written as shown in Figure 9-31.

Figure 9-31 Transition logic for the fill tank state of the chemical treatment system

Notice the use of the Fl_T Examine_ON input condition on the ORed rung. Because the start button is a pushbutton switch, the ORed rung is necessary to latch on the fill tank state. This ORed line is not shown on the state diagram as a general rule. However, if the programmer deems it necessary, it could be shown as a *latch loop* on the state diagram, as indicated in Figure 9-32.

Figure 9-32 Modified state diagram

The transition logic for the chemical reaction state is shown in Figure 9-33. Reference the state diagram showing the full float switch was ANDed with an Examine_OFF reaction time complete relay input condition (RT). The RT is activated when the chemical reaction timer (BT) has timed out. Recall that the chemical reaction timer (BT) begins its countdown when the chemical reaction state is activated (refer back to Figure 9-30). When the countdown is complete, BT becomes TRUE, which makes RT TRUE, which, in turn, deactivates the chemical reaction state.

Figure 9-33 Transition logic for the chemical reaction state of the chemical treatment system

Accordingly, the transition logic for the drain tank state can be written as shown in Figure 9-34.

Figure 9-34 Transition logic for the drain tank state of the chemical treatment system

Once the transition logic is complete, it is combined with the state logic to form the ladder logic program shown in Figure 9-35. The reader is encouraged to verify both the state logic and transition logic in relation to the work cycle program, state diagram, and state table.

Figure 9-35 Ladder logic program for the chemical treatment system

State diagrams are not the only method for developing ladder logic programs. A more common method is to write the program based solely on prior programming experience and then debug it until it works adequately. The author terms this approach the *trial and error programming method*. It can be effective for relatively simple programs, and with great effort it can work with more complex programs. However, the time necessary to get the program to function properly is often substantial. Additionally, the lack of program organization will make future debugging and modifications difficult. Programming from state diagrams, on the other hand, yields significantly better organized programs. The programs are simple to follow, enabling much easier troubleshooting and future modification. The disadvantage is that programs written using state diagrams tend to be larger, requiring more PLC memory. However, this is a small price to pay for an effective, well-organized program. For this reason the state diagrams method will be used exclusively in this text.

9.4.2 PLC Programming Process Steps

In this section we take the method just outlined and define it in terms of basic programming steps necessary to convert a work cycle program into a ladder logic program and load it on the actual machine. Before attempting this we recall the assumptions of Section 9.1: (i) a firm understanding of the process exists; (ii) the process is as simplified as possible; (iii) process steps have been broken down into a work cycle program and timing diagram (*Important*: At this point the work cycle diagram/timing diagram typically only shows real physical or external I/O); (iv) the PLC along with all sensors and actuators have been selected, installed, and wired.

Even if these assumptions hold, the PLC programming process is a dynamic, complicated process. However, eight basic programming steps can greatly simplify the process and yield effective programs:

1. Identify and document the system states on a state diagram.

System states are modes of operation in which the machine is performing an identifiable activity that has to be initiated, and then stopped. The work cycle program details the process steps necessary to execute the process. From this list of process steps one can identify system states from the status of the outputs. Each state will have a unique set of output values. Unlike the example in the previous section, the work cycle program may show a given state more than once. Once the states are identified, document them by assigning a label for each state and drawing them on the state diagram.

2. Identify and document the system transitions on a state diagram.

From the work cycle diagram identify that which causes each state to start execution and to cease execution. Identify both types of transition—either event-driven or time-driven—and which input or output initiates the change. It is important to note that at this point the work cycle diagram/timing diagram typically only shows real physical or

external I/O. In other words, there may not be a physical external input or output that can initiate the transition. Thus, an internal relay output, timer, or counter may be necessary to institute the event- or time-driven change. It is during this step of the programming process that the internal I/Os are identified.

Once the transitions are identified, add them as lines between the states documented in the previous step. Note that it is possible for a state to have more than one transition away from the state or more than one transition to it.

3. Create the state table.
- From the information gathered in creating the state diagram, create the state table. The state table documents the status of the outputs for each of the states, as was demonstrated in the previous section.

4. Write the program state ladder logic per the state table.
- State logic is written through internal logic elements that set the output status of each state as follows:
- Define each *active state* as an Examine_ON input condition with a unique "internal" address.
- Create a rung for each output.
- Use the appropriate state Examine_ON input condition to activate the appropriate output. If more than one state activates an output, they are to be ORed together on that rung.

5. Write the program transition ladder logic per the state diagram.
- Transition logic turns on the desired state by activating an internal relay with the same address as the Examine_ON input condition of the state logic. The transition ladder logic is created in reference to the state diagram according to the following rules:
  - Transition lines into the state bubble of the state diagram are Examine_ON input conditions.
  - Transition lines out of the state bubble of the state diagram are Examine_OFF input conditions that are ANDed with the lines into the bubble.
  - Multiple transition lines into the state bubble should be ORed.
  - Multiple transition lines out of the state bubble that lead to another state require a counter to differentiate the logic paths.

6. Add process interrupt logic to the program.
The example programs developed thus far provide control of the process only under normal circumstances. However, often a process will need to be stopped or interrupted if something occurs unexpectedly. This scenario is defined as a *process interrupt*. Some design features that are used to handle process interrupts include emergency stop switches

or other sensors that are added in case something goes wrong with the process. By definition such an interrupt does not occur frequently enough to be considered a state of the process. Thus, process interrupt logic is generally added after the state and transition logic has been developed. Typically, process interrupt logic is added to the state logic. However, if process interrupts are expected to occur regularly then they may be included on the state diagram. When this is the case a process interrupt is probably more of a typical system state. The addition of process interrupts to the state logic is explained in the program example of Section 9.6.

7. Simulate the ladder logic program of instructions to verify logic continuity.

The PLC program can be developed, edited, and tested offline, away from the machine. Thus, proof of concept can be determined and the process well understood before programming at the machine occurs. For most PLCs the programming software has simulation capabilities built directly into the software package. Thus, the programmer can test and verify the program logic before loading or, more appropriately, downloading it on the PLC.

If simulation is not included in the programming software, or the programmer would like to verify a program concept prior to incurring the expense of purchasing and installing the PLC and software, one can look to the Internet for free downloadable simulators. These simulators are also great aids to learning the concepts of PLC programming. Their use is explained in the next section.

8. Input and verify the program on the actual machine, workstation, or system being controlled.

This final step loads the PLC program into the PLC and tests it. If the PLC manufacturer's software was used to simulate the program, loading it on the PLC involves a simple downloading process from the programming device (typically a laptop PC) directly to the PLC. If, however, the program was simulated on separate software, the program has to be entered into the PLC through the programming device. Once the program is loaded on the PLC, thorough testing of the logic should ensue. Program editing may be necessary to ensure optimum machine performance.

9.4.3 Ladder Logic Program Organization

Organizing the program in a specific fashion with appropriate documentation is important for program performance verification and future debugging and troubleshooting. Thus, when you use the state diagram programming technique, place the ladder rungs in the following order:

1. *Process interrupt logic.* Although process interrupt logic is actually written after the state logic and transition logic are developed, it should appear as the first section of the ladder logic program. It is placed first because it provides master control of the process cycle—starting, stopping, and restarting the process.

2. *State logic*. Listing the state logic next enables the programmer to clearly define the states of the system. This proves to be very beneficial during program debugging and program modification.

3. *Counter and timer logic*. The rungs that are placed in this section should include any rungs developed from the transition logic that increment or decrement a counter or activate a timer. Placing these rungs in a separate section ensures they are scanned and acted upon prior to execution of the transition logic.

4. *Transition logic*. All rungs that represent actual transitions from one state to another should be placed in this section. This simplifies and aids in program troubleshooting and future modifications.

9.5 PLC Program Simulation

PLC simulation software provides a means to test, simulate, or verify a PLC program prior to running it on the actual PLC. Simulating the program enables visual confirmation that the program is performing as intended and serves as the primary debugging tool to catch logic errors. Thus, when the program is finally loaded on the PLC the programmer will have a high degree of confidence in the outcome.

A simulation program will execute a ladder logic program in the same manner that a PLC controls an actual machine. However, instead of using physical inputs and outputs to evaluate the logic, the software will simulate the I/O operation in the virtual world of a computer.

For the beginning programmer PLC simulation software is an invaluable tool. It enables the student to see, firsthand, how the PLC will interpret each rung of the ladder logic program. This gives the inexperienced programmer an understanding of both the I/O instructions and programming process. Additionally, verifications are performed in the safety of the virtual world, so when program errors occur, which they inevitably do for beginning programmers, there are no risks to student and machines.

PLC program simulation can be accomplished with either PLC manufacturer software or with free software available for download on the Internet. Software from the PLC manufacturer typically has to be purchased separately and appropriately licensed. The software is often proprietary and unique to the specific PLC brand. A discussion of all licensed simulators available is not practical. Instead, we focus on introducing the reader to a readily available free simulation software program, *i-TriLogi*®, or "TriLogi" for short, developed and published by Triangle Research International, Inc. TriLogi is capable of simulating any PLC program that uses the basic instruction set. Additionally, it can simulate programs that utilize advanced instructions that go far beyond the basic instruction set. We do not discuss this advanced capability, but rather focus on how TriLogi is used for the simulation of basic ladder logic programs. A screen shot of the program displaying the ladder logic from Example 9-3 is shown in Figure 9-36.

Figure 9-36 TriLogi software displaying ladder logic program for Example 9.3

The next section explains how to acquire, install and set up the software. Additionally, the section explains the user interface along with the supported instruction set. Simulation examples are also presented.

9.5.1 TriLogi Simulation Software

The Windows® version of TriLogo is available for download from Triangle Research International's website www.tri-plc.com, which lists any of the following as compatible (required) operating systems:

Windows 98®, Windows ME®, NT®, Windows 2000®, Windows XP®, or Vista® operating systems Java Runtime Environment® (JRE) 1.4.2 or newer version.

The website www.tri-plc.com gives instructions on how to determine which version (if any) is installed on your computer. If JRE® is not installed or you have an older version, follow the website instruction for obtaining and installing a new version of JRE.

It is important to note that the only operating system the author has used with this software is Windows XP®. Its compatibility with the Windows Vista® operating system was unknown by the author at the time of publication. The website www.tri-plc.com gives additional information.

To download the educational version of the software you must first register on the download page. The company's autoresponder will then email a password-protected download link. The password is required to install the software.

9.5.2 Installation

1. Locate the "SetupTL65du.exe" file. It is available for download from Triangle Research International's website (www.tri-plc.com) download page. Copy the file to a desired folder on your computer's hard drive.

2. Execute the "SetupTL65du.exe" file by double clicking on it. This will start the TriLogi setup process. The first screen to appear is a dialog box (Figure 9-37). Click on the "Yes" button to proceed with the installation.

Figure 9-37   TriLogi's first screen

3. The next screen to appear (Figure 9-38) is a warning page to exit any Windows programs before continuing with the setup process. Click on the "Next" button when ready to continue.

Figure 9-38 Warning screen

4. The next screen requires you to enter the password that was included in the email with the download link. Enter the password as shown in Figure 9-39.

Figure 9-39 Password screen

5. The next screen to appear is the License Agreement (Figure 9-40). Review the license agreement and click "Yes" when ready to continue or "No" to terminate setup.

Figure 9-40 License Agreement screen

6. Figure 9-41 shows the next screen to appear. This is another warning screen specifying that JR 1.4.2 or newer version must be installed prior to installing TriLogi.

Follow the instructions on the screen to determine which version is installed or if it is necessary and where to obtain it. Click "Next" when ready to continue or cancel to terminate setup.

Figure 9-41 Java warning screen

7. Figure 9-42 shows the next screen. Specify the desired location to install TriLogi. Click "Next" when ready to continue or cancel to terminate setup.

Figure 9-42 Select Destination Directory screen

8. The next screen requests the name of the group in which to place the program's shortcut icons (Figure 9-43). Click "Next" when ready to continue or cancel to terminate setup.

Figure 9-43 Shortcut location screen

9. The last screen prior to installation (Figure 9-44) summarizes all information entered up to this point. Press "Back" to make any adjustments. When "Install" is pressed the software is installed. Another dialog box to track installation progress will then appear.

Figure 9-44 Ready to Install screen

10. (Figure 9-45) is an information screen, which comes up next.

Figure 9-45 Information screen

11. The final screen of the setup process (Figure 9-46) confirms that TriLogi has been successfully installed.

Figure 9-46 Setup Completed screen

9.5.3 User Interface

The *user interface* (Figure 9-47), as the name implies, gives a means for the user to interact with the software. The interface has a pulldown menu area across the top and a ladder logic editor window, which essentially takes up the rest of the screen. The ladder logic editor utilizes two modes:

1. browse mode
2. circuit editing mode

Figure 9-47 TriLogi user interface

The browse mode is the default mode at the start of the program. This mode manipulates a ladder logic circuit rung as a single entity. The browse mode can do the following:

1. view circuits   3. move a circuit
2. copy a circuit   4. delete a circuit.

When the user interface is in browse mode, circuit navigation keys appear across the top of the editor (Figure 9-48). The circuit number button appears on the left. The circuit number is the number of the circuit or rung on which the circuit cursor is resting. The editing commands act on the circuit to which the cursor is pointing. To move the cursor to a different circuit or rung, simply use the vertical scroll bar or press the up or down arrow keys on the keyboard or press the circuit number button. This action will cause a "Goto Circuit" dialog box to open. Enter the number of the circuit that you wish to view in this box and hit "Enter." The cursor will then move to the corresponding rung.

Figure 9-48 Browse mode

For larger ladder logic programs, there are five "Quick Tags" located among the circuit navigation keys. These tags provide a means for one to quickly jump from one circuit to another. Each tag corresponds to a user-defined circuit number. To link a circuit number to a tag, press the "Define Quick Tags" button and enter the desired circuit number (Figure 9-49). This enables the user to quickly move among circuit rungs.

Figure 9-49 Define Quick Tags dialog box

Once the desired circuit rung is selected, choose "Edit" from the pulldown menu to access the circuit editing functions (Figure 9-50). Any one of the tasks of adding comments, inserting, moving, appending, or deleting circuits is accomplished from the "circuit" pulldown menu (Figure 9-51).

Figure 9-50 Edit pulldown menu

Individual circuit rungs are created and edited in circuit editing mode. This mode is accessed from the browser mode by either hitting the spacebar or double-clicking on an existing rung. Figure 9-52 shows TriLogi in circuit editing mode.

Figure 9-51 Circuit pulldown menu

Figure 9-52 Circuit editing mode

Note the ladder logic icons across the top of the editor window and the yellow color highlight bar. This bar is used to select a circuit element or to specify the location where elements should be inserted. The bar can be moved with mouse clicks or keyboard arrow keys.

The next section covers how to enter and simulate a ladder logic program in TriLogi.

9.5.4 Entering and Simulating a Ladder Logic Program

The TriLogi educational version enables the user to enter and simulate a ladder logic program without being connected to an actual PLC. This feature provides an excellent environment for learning ladder logic programming. The process of entering and simulating a ladder logic program into TriLogi is described below. In this example the

ladder logic program developed in Example 9.3 is entered into and simulated with TriLogi.

Example 9.4

Enter the ladder logic program developed in Example 9.3 and shown in Figure 9-23 into TriLogi and verify its functionality through simulation.

Solution

Entering and simulating a program into TriLogi involves three steps.

1. Set up the I/O table. Setting up the I/O table involves listing the addresses or labels for each input condition, output, timer, and counter instruction used in the program. To access the I/O table, right click on the "I/O Table" button at the top right of the logic editor window (Figure 9-53).

Figure 9-53 Accessing I/O tables

The red buttons on the side of the I/O table are for scrolling to the different instruction tables (Figure 9-54). Note how the timer and counter instructions each have an extra column ("Set Value") for setting the time or count, respectively. Also, observe that TriLogi has both output instructions and relay instructions. In the past we considered only output instructions. TriLogi uses output instructions for outputs physically wired to the PLC and relay instructions as internal logic instructions not physically connected to an output. Since we are only simulating our program and want to remain true to our original discussion on outputs, we will use only the output instruction in examples.

Click on red arrows to move between tables

Figure 9-54 Available I/O tables

To enter a label, simply click in a box and type in the name followed by hitting the enter key. The completed I/O tables for this example are shown in Figure 9-55.

Figure 9-55 Completed I/O tables

2. Create logic circuits with individual rung instructions. To create logic circuits and rung instructions first enter into circuit editing mode, creating a circuit, and then add individual instructions to the circuit rung.

- To enter into circuit editing mode, either press the spacebar or double click on the red circuit pointer. This will change the screen, as shown previously in Figure 9-52, and create the first circuit. To exit circuit editing mode, simply press Esc (escape key).
- Logic instructions are placed in the rung by left clicking on the desired logic instruction. For example, to add the S1 normally open input condition, left click on the normally open switch symbol. This will cause the input I/O table to pop open, as shown in Figure 9-56. Left click on the S1 label to place the condition instruction.

Figure 9-56 Inputting S1 condition instruction

- A general rule: Do not add branching instructions until all other instructions on the rung are added. This simplifies placing the branching circuits.
- Adding a normally closed condition instruction is very similar to adding a normally open instruction with one minor caveat: Instead of left clicking on the normally open instruction one must right click on it. This is shown in Figure 9-57.

Figure 9-57 Inputting S2 normally closed condition instruction

• The rest of the instructions, including the output instruction, are added to the circuit in a similar manner. The partially completed circuit, minus only the branch instruction, is shown in Figure 9-58.
•

Figure 9-58 Partially completed first circuit

• To add the normally open branching instruction that will span the S1 and O1 inputs, use keyboard arrow keys or left click to move the yellow highlight bar to the S1 normally open instruction. Click on the normally open multibranch icon to enclose one or more instructions. Then, using the arrow keys or mouse, move the yellow cursor to the O1 instruction and press the same icon once more to close the branch. Finally, select the M label from the output table. This series of steps is shown in Figure 9-59.

Figure 9-59 Adding a branch instruction

• To input the next circuit, move the yellow highlight bar to the output instruction at the right end of the circuit and press "Enter." This will move the highlight bar down to the next rung. Add the appropriate instructions for the next two rungs.

• To add the counter reset instruction, select the output function icon (Figure 9-60). This causes the "Select a Function" dialog box to open. Select the "Reset Counter" function. The ladder logic diagram is now complete and ready for simulation. To exit circuit editing mode hit the escape key.

Figure 9-60 Inputting the counter reset function

• Before proceeding to simulation, be sure to save the program. Since the program is being used only for simulation and not for actually programming an attached PLC, the ladder logic program file will be saved to a local drive. Select "File" from the pulldown menu and click on "Save as (Local Drive)."
• Note: For additional TriLogi circuit and supported instruction set information, refer to the TriLogi Reference Manual that is available for download from the www.tri-plc.com.

3. Simulate the program. With the program entered and saved to disk, it is time to simulate the program in real-time to verify the program logic. To simplify the amount of time it takes to perform the simulation, change the count from 100 to 5 for counter C in the I/O table, so you will not have to click on switch O1 100 times to verify the program logic!

To start the simulation, go to the "Simulate" pulldown menu and select "Run (All I/O reset)." This will open the "Programmable Logic Simulator" dialog box (Figure 9-61). Prior to opening the dialog box, TriLogi will compile the program to check for coding errors. If no errors are detected the dialog box will open.

Figure 9-61 Programmable Logic Simulator dialog box

Observe how the dialog box has a column for inputs, timers, counters, relays, and outputs. Listed in the columns are labels for each of the instructions used in the program. Next to the labels are numbers in grayed out blocks. These numbered blocks represent LED lamps that light up when the logic instruction is TRUE. The timer and counter columns have an additional subcolumn. For the timer this extra column will display the time counting down. For the counter it will display the count counting down. The inputs are turned on with the mouse. Left click on the label to operate the input as a pushbutton switch or right click to latch the switch on like a light switch.

• To simulate the program logic, left click on the start switch S1. Observe what occurs in both the Programmable Logic Simulator dialog box and the ladder logic diagram. The S1 switch will light momentarily; then the motor and light outputs will turn on. This is shown in Figure 9-62.
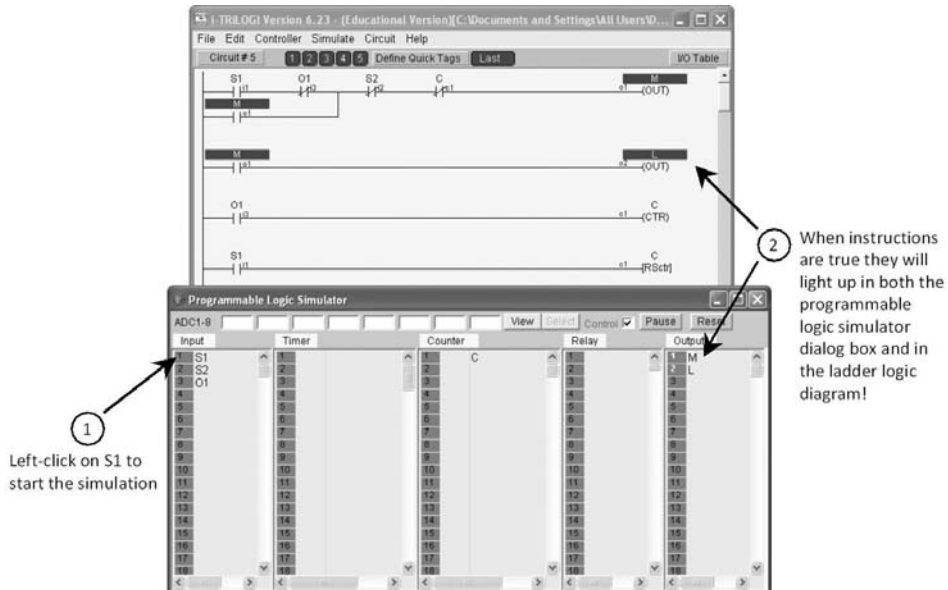

Figure 9-62 Simulation after S1 is pressed

Experiment with the program by performing the following tasks:
• Left click on S2. Do the motor and light turn off?
• Latch O1 on by right clicking on it. Left click on S1. Will the motor and light turn on? If not, why not? Unlatch O1 by left clicking on it.

• Right click on S1 to turn the motor and light on. Right click on O1. Observe the count displayed next to the C in the counter column. Note that the counters in TriLogi are countdown timers. They start at the value listed in the I/O table and decrement by 1 each time the circuit goes TRUE. This is shown in Figure 9-63. When the count reaches zero the counter instruction will be TRUE. Continue to press O1 until C goes true. Does this turn the motor and light off?
• Continue to experiment with the simulation until satisfied that the program functions as intended. To exit the simulation simply reset the I/O by pressing the "Reset" button and left click on the "x" in the upper right corner of the Programmable Logic Simulator dialog box.

Refer to TiLogi's reference manual for additional information on the program's simulation capabilities.

Figure 9-63 Counter counting down

9.6 PLC Programming Example

At the beginning of this chapter, the washing machine cycle was introduced and used to define logic control and sequencing. In this section we will develop a basic PLC Ladder Logic program for the washing machine cycle as outlined by the PLC programming process of Section 9.4. After the program is developed it will be simulated with TriLogi.

To start the programming process we will take the work cycle program shown previously in Figure 9-2 and modify it slightly as shown in Figure 9-64. Note that the modifications include adding step numbers to the left of the process step description, adding I/O addresses or labels below the I/O name, and changing off and on to 0 and 1, respectively. Each of these modifications simplifies the program development.

Step No.	Process Step Description	PLC Inputs (Sensors)				PLC Outputs (Actuators)			
		Start button	Stop button	Tub empty sensor	Tub full sensor	Fill solenoid valve	Agitator motor	Drain pump	Spin motor
		(Start)	(Stop)	(Tub_emp)	(Tub_ful)	(Valve)	(Ag_Mtr)	(Pump)	(Sp_mtr)
1	Waiting to start	0	0	1	0	0	0	0	0
2	Fill tub to wash	1	0	1	0	1	0	0	0
3	Wash (agitate tub)	0	0	0	1	0	1	0	0
4	Drain	0	0	0	1	0	0	1	0
5	Fill tub for rinse	0	0	1	0	1	0	0	0
6	Rinse (agitate tub)	0	0	0	1	0	1	0	0
7	Drain	0	0	0	1	0	0	1	0
8	Spin	0	0	1	0	0	0	1	1

Figure 9-64 Washing machine work cycle program with I/O labels

9.6.1 Identifying the System States

The first step in the PLC programming process as listed in Section 9.4.1 is identification and documentation of the system states. The work cycle diagram will be used to identify the states of the process. Recall each state will have a unique set of output values. Accordingly, there are five unique states as shown in Figure 9-65. Note that three of the states are repeated in the work cycle program. This is typical in sequential type processes.

Idle State

Fill Tub State

Step No.	Process Step Description	PLC Inputs (Sensors)				PLC Outputs (Actuators)			
		Start button (Start)	Stop button (Stop)	Tub empty sensor (Tub_emp)	Tub full sensor (Tub_ful)	Fill solenoid valve (Valve)	Agitator motor (Ag_Mtr)	Drain pump (Pump)	Spin motor (Sp_mtr)
1	Waiting to start	0	0	1	0	0	0	0	0
2	Fill tub to wash	1	0	1	0	1	0	0	0
3	Wash (agitate tub)	0	0	0	1	0	1	0	0
4	Drain	0	0	0	1	0	0	1	0
5	Fill tub for rinse	0	0	1	0	1	0	0	0
6	Rinse (agitate tub)	0	0	0	1	0	1	0	0
7	Drain	0	0	0	1	0	0	1	0
8	Spin	0	0	1	0	0	0	1	1

Agitate Tub State

Spin Tub State

Drain Tub State

Figure 9-65 Identifying system states on the work cycle program

To document the states, write the state titles around an imaginary circle in a counterclockwise fashion. Place a circle around each state title as shown in Figure 9-66.

Figure 9-66 Preliminary state diagram

9.6.2 Identifying System Transitions and Completing the State Diagram

Transitions cause the system to change states. Transitions may be either event-driven changes or time-driven. Also, states may have multiple transitions. The work cycle program will aid in identifying both event-driven and time-driven transitions. Event-driven will be clearly shown on the work cycle program as a change in an input status from 0 to 1. Note that a transition may appear more than once in the work cycle program. When no event-driven transition is identifiable on the work cycle program, a time-driven change is typically indicated. Thus, for the washing machine work cycle program, six unique transitions are identified as shown in Figure 9-67.

Step No.	Process Step Description	PLC Inputs (Sensors)				PLC Outputs (Actuators)			
		Start button (Start)	Stop button (Stop)	Tub empty sensor (Tub_emp)	Tub full sensor (Tub_full)	Fill solenoid valve (Valve)	Agitator motor (Ag_Mtr)	Drain pump (Pump)	Spin motor (Sp_mtr)
1	Waiting to start	0	0	1	0	0	0	0	0
2	Fill tub to wash	1	0	1	0	1	0	0	0
3	Wash (agitate tub)	0	0	0	1	0	1	0	0
4	Drain	0	0	0	1	0	0	1	0
5	Fill tub for rinse	0	0	1	0	1	0	0	0
6	Rinse (agitate tub)	0	0	0	1	0	1	0	0
7	Drain	0	0	0	1	0	0	1	0
8	Spin	0	0	1	0	0	0	1	1

Figure 9-67 Identifying transitions on the work cycle program

Note that transitions 2 and 3 are called *repeat transitions*. Repeat transitions occur when a transition between two identical states occurs in the work cycle program. For a given washing cycle, the tub will fill, be agitated, and be drained twice. Thus, the transition between the fill tub and agitate tub states will occur two times, as will the transition between the agitate tub and drain tub states. Note that transitions 4 and 7 are unique because they transition between different states. The semi-complete state diagram is shown in Figure 9-68.

Figure 9-68 Semi-complete washing machine state diagram

The state diagram shown in Figure 9-68 displays much more information than the state diagrams shown in Figures 9-29 and 9-32. This is mainly because the washing machine system is much more complicated than the chemical treatment system examined in Section 9.4.1. The additional information added to the state diagram of Figure 9-68 is intended to add clarity for programming purposes. The circled numbers next to the transition lines are the transition numbers listed in Figure 9-67. The work cycle program step numbers are listed near the state bubbles. Addition of the step numbers is particularly helpful for programming sequential systems.

Figure 9-68 is described as a *semi-complete state diagram*, because additional transition conditions are needed. Note that transitions 4 and 7 have the identical condition Tub_emp = 1. Thus, as the system is shown in the state diagram, the controller cannot see which path to follow. Recall that rule (d) from programming step 5 of Section 9.4.2 stated: "Multiple transition lines out of the state bubble that lead to another state require a counter to differentiate the logic paths."

This is precisely the situation and a common occurrence in sequential-type systems. Thus, a countdown timer with address "Spin_C" will be added to transition 7 to differentiate the logic paths. Each time the drain tub state goes from OFF to ON the counter Spin_C will be decremented. The completed state diagram is shown in Figure 9-69.

Figure 9-69 Completed washing machine state diagram

9.6.3 Creating the State Table

The state table documents the status of the outputs for each of the system states. The state table is created in reference to Figure 9-65 and is shown in Figure 9-70.

	Output Status			
	Fill solenoid valve	Agitator motor	Drain pump	Spin motor
State Address	(Valve)	(Ag_Mtr)	(Pump)	(Sp_mtr)
Idle	0	0	0	0
Fill_tub	1	0	0	0
Ag_tub	0	1	0	0
Drain_tub	0	0	1	0
Spin_tub	0	0	1	1

Figure 9-70 Washing machine state table

9.6.4 Writing the State Logic

Following the procedure outlined in Section 9.4.2, step 4, the state logic is written as shown in Figure 9-71.

Figure 9-71 Washing machine state logic

Note that there is a rung for each output. Also, notice from the state table that the drain tub state and the spin tub state both require the pump to be ON. Thus, the last rung of the state logic has the drain tub state ORed with the spin tub state.

9.6.5 Writing the Transition Logic

Begin writing the transition logic by starting with the first state and proceeding to the next in a counterclockwise fashion around the state diagram. Referencing Figure 9-69, note that the "Fill tub state" has two transition lines (transitions 1 and 4) into it and one transition line exiting it (transition 2). Following the procedure outlined in step 4 of Section 9.4.2, step 5, the transition logic into the fill tub state is written as shown in Figure 9-72.

Figure 9-72 Fill tub state transition logic

This rung turns on the Fill_tub state. As Section 9.4.2 specifies, the transition line into the state bubble, Start = 1, is represented as an Examine_ON input condition and is ANDed with an Examine_OFF input condition of the transition line exiting the state bubble (Tub_ful=1). Also per the rules of Section 9.4.2, transition 4 is ORed with transition 1, as is shown in Figure 9-72. Note that an Examine_OFF input condition of the spin counter must be included on this rung to differentiate transition 4 from transition 5. Finally, since the start button is assumed to be a pushbutton, the Fill_tub state is latched on by the third ORed rung. When the tub is full, the fill tub state will terminate and transition to the agitate tub state.

The transition logic for the agitate tub state is shown in Figure 9-73. As shown in Figure 9-69, the agitate tub state has one transition into it and one out. Thus, the Tub_ful Examine_ON input condition is ANDed with the Ag_tmr Examine_OFF input condition. Since a timer is used to control the duration of the agitate tub state, another rung is required as shown in the figure. The Tub_ful Examine_ON input condition will be used to activate the countdown timer. As defined in Section 9.4.3, this rung will be placed in the counter and timer section of the program. When the timer "times out" and comes on, the agitate tub state ends.

Figure 9-73 Agitate tub state transition logic

The transition logic for the drain tub state is shown in Figure 9-74. Recall that a countdown counter was added to the system to differentiate the logic paths exiting the drain tub state. The first rung in Figure 9-74 is the actual state transition logic. Following the rules of Section 9.4.2, the Ag_tmr Examine_ON input condition is ANDed with Examine_OFF input conditions exiting the state bubble. For this case there are two transition lines exiting the state. Thus, there are two Examine_OFF input conditions ANDed on the rung. The Drain_tub latch is needed because the Ag_tmr will be reset to OFF as soon as the Tub_ful switch goes off (Figure 9-73). When the tub is empty, the drain tub state will be terminated. The second rung provides a means to decrement the counter. Each time the tub is drained (tub_emp = 1) the counter will be decreased by 1. When it reaches zero the counter will turn on (Spin_C = 1). Note that the second rung is not connected to the first rung in the figure. This is because in the final program this rung will actually be located in the counter and timer section of the program. This is necessary to decrement the counter before reaching the transition logic for the fill tub state. As discussed in Section 9.4.3, this is standard procedure when counters and timers are used because of the way the PLC scans the program from top to bottom. Thus, organizing the program in this fashion will guarantee the counter is decremented before the transition logic is reached.
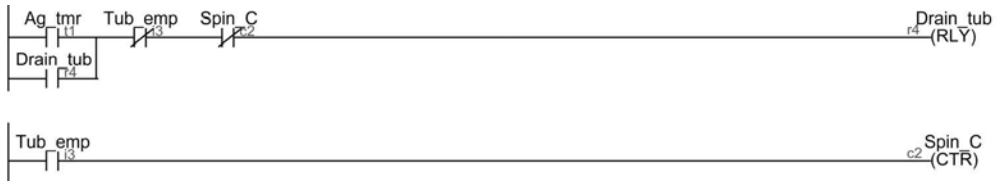
Figure 9-74 Drain tub state transition logic

The transition logic for the spin tub state is shown in Figure 9-75. Again following the rules of Section 9.4.2, the transition logic can be written as shown on the first rung. Since a timer is used to control the duration of the spin tub state, another rung is required as shown in the figure. The Spin_C Examine_ON input condition will be used to activate the countdown timer. This rung will be located in the counter and timer section of the program. When the timer "times out" and comes on, the spin tub state ends and the washing machine cycle is complete.

Figure 9-75 Spin tub state transition logic

Utilizing the program format defined in Section 9.4.3, one can now combine the state logic with the transition logic to form the completed program, as shown in Figure 9-76. Note the addition of the last rung. As it was discussed in Section 9.3.3, counters require the use of a reset output instruction. The TriLogi software instruction set includes a master reset output instruction, which resets all counters used in the program. So, the last rung of the program issues a master reset when the spin timer times out.

```
1     STATE LOGIC

2     Spin_tub                                              S_mtr
      ─┤ ┤r5                                              o4─(OUT)

3     Fill_tub                                              Valve
      ─┤ ┤r2                                              o1─(OUT)

4     Ag_tub                                               Ag_Mtr
      ─┤ ┤r3                                              o2─(OUT)

5     Drain_tub                                            Pump
      ─┤ ┤r4                                              o3─(OUT)
      Spin_tub
      ─┤ ┤r5

6     COUNTER & TIMER LOGIC

      The rung below decrements the spin counter
      each time the tub is drained

7     Tub_emp                                              Spin_C
      ─┤ ┤i3                                              c2─(CTR)

8     The agitation timer will begin counting down whenever the tub is full

9     Tub_ful                                              Ag_tmr
      ─┤ ┤i4                                              t1─(TIM)

10    The spin counter will be used to activate the spin countdown timer.

11    Spin_C                                               S_tmr
      ─┤ ┤c2                                              t3─(TIM)

12    TRANSITION LOGIC

      Fill Tub State
      Transition Logic

13    Start        Tub_ful                                Fill_tub
      ─┤ ┤i1───────┤/├i4──────────────────────────────── r2─(RLY)
      Tub_emp  Spin_C
      ─┤ ┤i3───┤/├c2
      Fill_tub
      ─┤ ┤r2
```

Figure 9-76 Washing machine ladder logic program

The program listed in Figure 9-76 contains all the necessary state and transition logic to repeatedly and successfully execute the washing machine cycle as specified by the work cycle program. However, it is not yet a program that could be used in practice because

there is no contingency to interrupt the program if a situation needs to be addressed. Thus, the next section discusses adding process interrupts.

9.6.6 Adding Process Interrupts

Process interrupts essentially pause the work cycle program midstream. In other words, issuing a process interrupt to a system is, in effect, the same as saying, "Stop, something is wrong." When the process is restarted it is typically desirable to start the process exactly where it was paused. This is the situation with the washing machine cycle. For example, if the process is paused during the tub filling state, say, to add detergent, it would be desirable to restart the process exactly where it was paused. Another example would be if the tub was unbalanced during the spin state. One would like to stop the spin state and reposition clothes. Once repositioning is complete it would be desirable to just restart the spin state. One would not want to go through the whole washing machine cycle again.

Utilizing the state diagram approach to programming makes adding process interrupts relatively easy. Since interrupts involve only pausing a particular state, the simplest approach is to add an "in process" or "in cycle" rung in the process interrupt section of the program. The output of this rung would be an internal logic relay that must be TRUE or ON to activate the outputs of any state. This relay will then be added to the state logic to enable the states to be paused if the "in cycle" relay is not active.

For the washing machine cycle program the process interrupt rung will use the start and stop buttons as the input conditions. The relay output of the rung is addressed "In_Cycle." This is shown in Figure 9-77. The start button turns on the In_Cycle relay and the stop button turns off the relay. Since both the start and stop buttons are assumed to be pushbuttons, an ORed rung is provided to latch the relay on.

Figure 9-77 Washing machine ladder logic process interrupt rung

The next step is to add the "In_Cycle" relay and an input condition to the state logic. This is shown in Figure 9-78. Thus, the only means in which a state can be active is if the "In_Cycle" is turned on.

Figure 9-78 Washing machine state logic with process interrupt relay

The final step to adding the process interrupt is to add the "In_Cycle" relay to the counter and timer section of the program, as shown in Figure 9-79. This is important because one would not want the timers to time out or the counter to count if the cycle is paused.

Figure 9-79 Washing machine counter and timer logic with process interrupt relay

Note that it is undesirable to add process interrupt logic to the transition logic. The reason for this is that process interrupts should only pause the system's current state, not the logic to reach said state. Thus, process interrupt logic is added to only the state logic and the counter and timer logic sections of the program.

The completed ladder logic program for the washing machine cycle is shown in Figure 9-80. The program is now ready for simulation.

1   Process Interrupt Logic

2
```
    Start      Stop                                                    In_Cycle
    ──┤ ├──────┤/├──                                              r6 ──(RLY)──
      i1         i2
    In_Cycle
    ──┤ ├──
      r6
```

3   STATE LOGIC

4
```
    Spin_tub  In_Cycle                                                   S_mtr
    ──┤ ├──────┤ ├──                                              o4 ──(OUT)──
      r5         r6
```

5
```
    Fill_tub  In_Cycle                                                   Valve
    ──┤ ├──────┤ ├──                                              o1 ──(OUT)──
      r2         r6
```

6
```
    Ag_tub   In_Cycle                                                   Ag_Mtr
    ──┤ ├──────┤ ├──                                              o2 ──(OUT)──
      r3         r6
```

7
```
    Drain_tub In_Cycle                                                   Pump
    ──┤ ├──────┤ ├──                                              o3 ──(OUT)──
      r4         r6
    Spin_tub
    ──┤ ├──
      r5
```

8   TIMER & COUNTER LOGIC

The rung below decrements the spin counter
each time the tub is drained

9
```
    Tub_emp  In_Cycle                                                   Spin_C
    ──┤ ├──────┤ ├──                                              c2 ──(CTR)──
      i3         r6
```

10   The agitation timer will begin counting down whenever the tub is full

11
```
    Tub_ful  In_Cycle                                                   Ag_tmr
    ──┤ ├──────┤ ├──                                              t1 ──(TIM)──
      i4         r6
```

12   The spin counter will be used to activate the spin countdown timer.

13
```
    Spin_C   In_Cycle                                                   S_tmr
    ──┤ ├──────┤ ├──                                              t3 ──(TIM)──
      c2         r6
```

14   TRANSITION LOGIC

Fill Tub State
Transition Logic

Figure 9-80 Continued on next page

Figure 9-80 Complete washing machine ladder logic program

9.6.7 Program Simulation

The reader is encouraged to simulate the program listed in Figure 9-80 per the TriLogi simulation instructions specified previously in Section 9.5.4. Set the Spin_C to 3. The time settings for the agitation timer and spin timer can be arbitrary. However, set them to be short enough so that the simulation can be accomplished in a reasonable amount of time, but not so short that the process steps can't be easily verified.

When performing the actual simulation, be sure to manipulate the inputs realistically. In other words, prior to starting the process, the Tub_emp input will be ON (left click to latch on). Once the start button is pressed and the fill valve opens, the Tub_emp input will go off. When the tub is full, the Tub_ful input will go on. Note that unrealistic manipulation of the process inputs may result in misleading simulation results. Figure 9-

81 shows a screen shot of the program simulation prior to the start button being pressed to initiate the process.

Figure 9-81 Washing machine ladder logic program simulation

9.7 Summary

Programming a PLC involves issuing instructions or commands for the PLC to follow in order to execute the manufacturing tasks or steps of the process being controlled. The industry standard language of the PLC is a graphical language called ladder logic. It uses graphically represented instructions to convert the work cycle program into a logic control program that the PLC can use to control the process.

Logic control provides a means to set the status of the outputs of the PLC solely on the status of its inputs. The status of the inputs is altered by event-driven changes. The program logic then sets the status of the outputs. In sequencing control, time-driven changes drive the status of the outputs. PLC ladder logic programs execute both logic and sequencing control.

Ladder logic diagrams came to their name because the drawing resembles a ladder with the logic elements (the sensors and actuators) making up the horizontal rungs and the vertical rails representing the power to the elements. When logic continuity is achieved across a rung the output for that rung will be activated, or turned ON. To aid in sorting out the logic of the ladder diagrams, logic gates and truth tables are often used. The three basic logic gates are the AND, OR, and NOT gates. Combining of basic logic gates within and among multiple rungs of a ladder logic diagram is the essence of how the work cycle program gets translated, converted, or programmed into the final ladder logic diagram program that directs the PLC in control of the process.

The logic instructions on the rungs of the ladder diagram are separated into condition instructions and output instructions. Condition instructions examine the status of the inputs. There are only two types: the Examine_ON and Examine_OFF conditions. Output instructions energize the outputs. The three basic types of output instructions include an output energize instruction, a timer instruction, and a counter instruction. These instructions collectively make up the basic PLC instruction set. Instructions may reference either a real external physical sensor or actuator wired to the PLC or an "artificial" internal logic element of the PLC. Internal logic elements are used to create the appropriate logic conditions to complete the work cycle program logic.

PLC programming involves converting the work cycle program and/or timing diagram of the machine, workstation, or system being controlled into the language of the PLC. For relatively simple processes a good method to use to develop the PLC ladder logic is to write out the input conditions and output status statement for the process in terms of the AND, OR, and NOT logic gates. As processes become more complicated, thinking of the process's operations in terms of states, as defined by the outputs and transitions as defined by the inputs can greatly aid in the development of the PLC's ladder logic program. An effective tool for visualizing the process in these terms is a state diagram. A state diagram graphically displays the various states and corresponding transitions between those states of the machine or system being controlled.

State diagrams are created through, first, identification of the states of the process. By definition, each state's output status will be unique. The next step involves identifying the transitions between the states. Transitions occur when inputs change from OFF (0) to ON (1), so, they are easily identified from the work cycle program. Once the states and transitions are identified, the state diagram for the system is constructed and a state table constructed.

Upon completion of the state diagram and corresponding state table, the PLC ladder logic program can be directly developed. This is accomplished by writing the state logic

per the state table, then writing the transition logic. State logic is written using internal logic elements to set the output status of each state. The transition logic will turn on, or activate, the desired state and deactivate all others. The transition ladder logic is created in reference to the state diagram and follows the simple rules outlined in the chapter. State diagrams program results into much better organized programs than would be created without such as diagram. State diagrams that are simple to follow and enable easy troubleshooting and future program modification.

Programs should be organized for easy debugging and troubleshooting. The first section of a program contains the process interrupt logic, followed by the state logic, counter and timer logic, and finally the transition logic.

The basic steps to follow when programming a PLC are (i) identifying and documenting the system states and transitions on a state diagram, (ii) creating a state table, (iii) writing the program state logic, (iv) writing the transition logic, (v) adding process interrupt logic, (vi) simulating the program for logic continuity, and finally (vii) inputting and verifying the program on the actual system.

PLC simulation software provides a means to test, simulate, or verify a PLC program prior to running it on the actual PLC. Simulating the program enables visual confirmation that the program is performing as intended. It serves as the primary debugging tool to catch logic errors. PLC program simulation can be accomplished with either the PLC manufacturer's software or with free software available for download on the Internet. Free simulation software called TriLogi, as extensively reviewed in the chapter, is recommended.

9.8 Key Words

AND gate
basic PLC instruction set
condition instructions
counter instruction
data table
Examine_OFF
Examine_ON
instruction address
ladder logic
ladder logic diagram
latch loop
logic control
logic gates
logic pulse train
NOT gate
OR gate

Output Energize
output instructions
process interrupts
reset output instruction
state diagrams
state table
status bit
timer instruction
transition
truth tables
user interface

9.9 Review Questions

1. Define programming from a PLC perspective.
2. Discuss the differences between logic control and sequencing control. Which type(s) is (are) utilized by PLCs?
3. Draw the ladder diagram and list the truth tables for the AND, OR , NOT, NAND, and NOR logic gates.
4. Explain the difference between condition instructions and output instructions.
5. Describe the functions of an instruction address.
6. How are internal logic elements used in PLC programs?
7. List and discuss the two types of condition instructions.
8. List and discuss the function of each of the instructions in a basic PLC instruction set.
9. Name and explain the three scans of a PLC's operating cycle.
10. Using the TriLogi simulation program, simulate and verify the ladder logic program developed in Example 9.2.
11. Using the TriLogi simulation program, simulate and verify the ladder logic program developed in Example 9.3.
12. Develop a state diagram and state table for Example 9.3.
13. List and discuss the four rules for writing transition logic.
14. List and discuss the eight basic programming steps.
15. Explain how a PLC ladder logic program should be organized, assuming the state diagram programming method is used.
16. Using the TriLogi simulation program, simulate and verify the ladder logic program developed for the chemical treatment system discussed in Section 9.4.1. Add a process interrupt to the program assuming a stop button is added to interrupt the process and cause an immediate dump of the tank. Modify the state diagram and state table accordingly.

17. Using the TriLogi simulation program, simulate and verify the ladder logic program developed for the washing machine cycle developed in Section 9-6 and shown in Figure 9-80. Modify the program to enable two rinse cycles, assuming a pushbutton input is added to indicate when two rinse cycles are desired. Modify the state diagram and state table accordingly.

9.10 Bibliography

1. Groover, M.P., 2001, *Automation, Production Systems and Computer-Integrated Manufacturing,* Second Edition, Prentice Hall, Upper Saddle River, New Jersey.
2. Derby, D., *2005, Design of Automatic Machinery,* Marcel Dekker, New York, New York.
3. Morriss, S.B., 1995, *Automated Manufacturing Systems,* Glencoe/McGraw-Hill, Columbus, Ohio.
4. www.automationdirect.com
5. Jack, H.,2008, *Automating Manufacturing Systems with PLCs, Version 5.0*, May, 2008, jackh@gvsu.edu, accessed.

Chapter 10

Automated Workstations and Work Cells

Contents

Objective

The objective of this chapter is to explain how programmable automation, with other components and devices, is used in automated workstations and work cells.

10.1 Automated Workstations and Work Cells

A workstation performs a specific task, process, or job during the manufacture of a product. A *manual workstation* groups resources—physical or otherwise (devices, equipment, tools, and labor)-in a logical way so that a well-defined activity may take place. The manual piercing die process discussed in Chapter 8 and shown in Figure 8-0 is an example of a manual workstation. This manual workstation was transformed into an *automated workstation* through automation of the piercing die process—a robot replacing a person to do the material handling and a PLC-coordinated system to control the process. So, much like a manual workstation, an automated workstation is a logical grouping of devices, equipment, components, and tools that execute a processing work cycle—but does so in an automated way.

To be deemed "automated" a system must be able to function unattended for more than a single cycle: An operator is required only to periodically tend to it. Figures 10-0 and 10-1 show examples; each workstation performs an assembly operation.

Figure 10-0 Example of an automated workstation

Figure 10-1 Example of an automated workstation

The workstations shown in the figures have material handling and storage devices, automatic assembly devices, and PLCs. The material handling and storage devices facilitate unattended operation, so an operator need only periodically load and unload the workstation. The PLC imparts machine procedural control of the assembly process.

Note that in some of the literature, the automated workstation as we define it here is called a *single station automated cell*. In fact, the terms "workstation" and "work cell" are often used interchangeably. However, we will use the term "workstation" so as not to confuse the reader that it is related to a *manufacturing cell*. (Recall from Chapter 1 that we defined "manufacturing cell" as an interconnected group of manufacturing processes tended by a material handling system. A manufacturing cell typically contains *individual* workstations.) One could replace the phrase "interconnected group of manufacturing processes" with "interconnected group of workstations." Like workstations, manufacturing cells can be manual or fully automated. Figures 10-2 and 10-3 show typical automated work cells.

Figure 10-2 Automated work cell

Figure 10-3 Automated work cell

Another distinction between workstations and work cells is this: Workstations are self-contained entities that perform one type of operation (more like a machine), whereas work cells contain multiple workstations that perform multiple operations. For instance, the workstations shown in Figures 10-0 and 10-1 perform only one operation: the

assembly of two components. The work cell of Figure 10-2 performs two operations—the molding of a pressed wood disk and the routing of the inside and outside diameters of each disk. It is left to the reader to identify the number of operations performed in the work cell of Figure 10-3.

Programmable automation is evident in either an automated workstation or work cell: CNC equipment often provides specific machining operations within both; robots are often the system of choice for material handling; and PLCs impart either procedural machine control or coordinated system control.

The next section discusses some other common components used in workstations and work cells.

10.2 Workstation and Work Cell Components

Besides extensive use of programmable automation, other workstation and work cell components include sensors and individual actuators, as discussed in Chapter 8, and the following:

- structural members
- part feeders and storage devices
- drive mechanisms
- material handling system

10.2.1 Structural Members

The structural members of workstations or work cells support, hold, and accurately locate the individual components. This is typically accomplished with a frame and tooling plate. Many different types and configurations of structural members have been used over the years. The relative location of the components in a workstation or work cell is very important.

Consider the workstation shown in Figure 10-1. The vibratory hoppers, bowl feeders, and assembly tooling are bolted to 1/2-inch steel tooling plate. A welded steel frame (Figure 10-4) supports the tooling plate. Dowel pin holes are drilled in both component mounting bracket and tooling plate to provide accurate location. Bolts can be used to hold the components together.

Welded steel frames may be used when strength and rigidity is important. Frames that are built up or bolted together may lose their rigidity over time as bolts become loose. In many applications, 2-inch square steel tubing with wall thickness of 3/16 inch provides a very rigid frame. Built-up extruded aluminum frames are popular when lightweight framing is essential. These frames can be purchased cut to length, with a wide assortment of accessories. There are many suppliers of this type of frame. One can essentially design a frame and purchase all necessary components and accessories as a kit for later assembly. A built-up extruded aluminum frame was used as the enclosure for the automated

workstation shown in Figure 10-0. A typical extruded aluminum frame member is shown in Figure 10-5.

Figure 10-4 Welded steel frame and tooling plate

Figure 10-5 Typical extruded aluminum frame member

10.2.2 Part Feeders and Storage Devices

The general role of part feeders is to place a part in the correct orientation at the right time and not let this orientation change throughout processing. The three standard types of feeders are:

- escapement feeder
- vibratory bowl feeder
- centripetal feeder

Escapement feeders release parts one at a time or multiple parts at one time. They can be used with rigid parts that have adequate part tolerances. Figure 10-6 shows one of the many different design concepts, a simple slide-type escapement. Here, the parts are stacked vertically in a tube. A plunger slides forward, pushing one part out while supporting the rest of the stack. As the plunger retracts the parts drop down, preparing the next part for feeding.

Plunger Moves
Forward Feeding
Part

Figure 10-6 Slide escapement

A *rotary-type escapement feeder* was used in the assembly workstation shown in Figure 10-0. A detailed view of this type of escapement feeder is shown in Figure 10-7. In this case, instead of a plunger, a dial rotates clockwise to move the insert into alignment with the grommet for assembly.

Vibratory bowl feeders are perhaps the most common type of part feeders in automated workstations and work cells. They feed parts by alternating vibrations to move parts up a spiral ramp on the inside of a curved surface bowl. *Fiducials*, built into the bowl, allow only those parts in correct orientation to make it to the top and exit the bowl. They knock parts in the wrong orientation back into the bowl. Part motion results from the combined effects of friction coefficient, bowl incline angle, angle of bowl vibratory motion, vibration frequency, and vibration amplitude. Vibratory bowls are custom-made, and, though expensive, they are highly reliable and dependable. Figure 10-8 is a sketch of a vibratory bowl feeder.

Figure 10-7 Rotary escapement

Figure 10-8 Vibratory bowl feeder

Centripetal feeders use centripetal force to move parts in a spinning bowl to the outer edge of the bowl, where they eventually exit the bowl. They can generally feed parts faster than vibratory bowl feeders. Fiducials are built into the side of the bowl to prevent parts in the wrong orientation from exiting the bowl. The automated workstation shown in Figure 10-1 used a centripetal feeder for the part 2 bowl feeder. Its centripetal feeder is shown in Figure 10-9. Figure 10-10 is a top view of the feeder, showing detailed information on the fiducials used in the bowl.

Figure 10-9 Centripetal feeder

Figure 10-10 Centripetal feeder (detail)

Storage devices are used to store large quantities of parts at the workstation. They are necessary, because part feeders typically handle only a relatively small volume of parts at a time. Storage devices replenish part feeders on demand as the feeders run low on parts, and they typically consist of a hopper and some kind of device to extract the product from hopper. Figure 10-11 shows the storage device for the centripetal feeder previously discussed in the workstation shown in Figure 10-1. This particular storage device uses a vibratory feeder to refill the centripetal bowl feeder.

Figure 10-11 Storage device with vibratory feeder

10.2.3 Drive Mechanisms

Drive mechanisms often transform actuator motion into some other form. For instance, a rotary actuator such as an electrical motor can provide linear motion with a ball-screw drive mechanism. Drive mechanisms may also move actuator motion from one location to another, such as happens with a chain and sprocket. Displacement, speed, and torque are transferred from the drive sprocket to the driven sprocket, located some distance away, by the chain. Drive mechanisms may also alter the speed and torque of the actuator to values more appropriate for a particular workstation or work cell. A worm gear speed reducer is an example of this type of drive mechanism.

In general, drive mechanisms fall into five major categories:
- V-belts and pulleys
- chain and sprockets
- timing belts and sprockets
- gears and gearboxes
- cams

V-belts and pulleys are generally quiet but have slippage issues, which can be of concern when accurate transfer of rotary motion is required. *Chains and sprockets* are more accurate than V-belts but have their drawbacks. They are noisy, dirty, and require regular maintenance.

Timing belts and sprockets capture the benefits of both V-belts and chains, without the drawbacks. Timing belts and sprockets are quiet, low maintenance, and very accurate. A timing belt and sprocket is shown in Figure 10-12.

Figure 10-12 Timing belt and sprocket

Gearboxes are also effective drive mechanisms for altering speed and torque. *Speed reducers* are commercially available gearboxes that come in a wide variety of configurations with many different gear ratios possible. Figure 10-13 shows a cutaway view of a worm gear speed reducer. Worm gear speed reducers can produce the greatest speed reduction and corresponding torque increase. Other types of speed reducers include *helical gear* and *bevel gear* speed reducers. Custom-designed *spur gear trains* are often used, as well.

Figure 10-13 Worm gear speed reducer

Cams offer some very interesting capabilities. Cams can be designed to provide essentially any motion profile desired. They can transform rotary motion into linear motion, rotary motion into oscillatory motion, or they can even produce intermittent motion. Figure 10-14 shows some examples of cam mechanisms. Note that a barrel cam, similar to the one shown in Figure 10-14(c) was used to produce a linear pressing stroke in the assembly workstation shown in Figure 10-0.

Figure 10-14 Cam examples

Pick-and-place units are cam-driven devices that provide intermittent motion and lift. For instance, a linear pick-and-place unit can provide two-dimensional linear motion, as shown in Figure 10-15(a). *Mechanical indexers* provide intermittent motion as "dwell – index – dwell – index." This is shown in Figure 10-15(b). Indexing drives can be used to provide rotary material handling when a turntable is attached to an output shaft. They can

also be used to drive conveyors and provide intermittent motion. Typical motion is "stopping – lifting – indexing – lowering – stopping." Examples of pick-and-place units and mechanical indexers are given in the next section.

Figure 10-15 (a) Pick-and-place motion; (b) indexer motion

10.3 Automated Workstation and Work Cell Examples

In this section we show how, through examples, programmable automation in conjunction with many of the components and devices discussed previously are utilized in automated workstations and work cells. The role of programmable automation will be highlighted as the function of the other major components is identified.

Example 10.1

Consider again Figure 10-0, and recall that it shows an automated workstation that assembles two components: a metal insert with a rubber grommet. The inner workings of the workstation are shown in Figure 10-16. An electric motor drives a worm gear speed reducer, which is connected to the shaft on the left through a torque-limiting device. The shaft on the left rotates continuously to drive the Geneva mechanism and the barrel cam. A *Geneva mechanism* is a mechanism that gives intermittent motion. Thus, the shaft on the right rotates intermittently to feed parts into the assembly position (i.e., dwell – rotate – dwell). When this shaft dwells, the barrel cam pushes the slide forward, assembling the two components. The cam then retracts the slide and dwells as the Geneva mechanism indexes two new parts into position for assembly. The assembled parts get knocked out the back of the assembly mechanism.

A PLC provides procedural machine control of the process. Although CNC technology is not an integral part of the workstation, many of the components, including

the barrel cam, were produced with CNC machines. Robotics plays no role in the workstation.

This workstation is an example of fixed automation because the assembly processing station is linked with the material handling. Therefore, the processing sequence is fixed. Additionally, the workstation can process only one type of insert and grommet. Note the use of a tooling plate for mounting of components and the extruded aluminum frame for enclosing the workstation.

Figure 10-16 Automated assembly workstation.

Example 10.2

Figure 10-17 shows a workstation for routing small round pressed wood disks. The workstation makes use of a rotary pick-and-place unit for material handling to pick the parts up from the conveyor, move the parts to the routing stations, and release the parts to the exit shoot. Note the motion of the pick-and-place unit. The first routing station routes the inside diameter and the second station the outside diameter. The unit is driven by an electric motor and worm gear speed reducer. A welded steel frame and tooling plate is used for mounting the components.

A PLC provides procedural machine control over the workstation. Again, CNC technology was used to produce many of the components of the workstation. This workstation also is an example of fixed automation.

Example 10.3

Figure 10-18 shows two photos of another automated workstation, which routes the outside and inside diameters of a large pressed wood disk. The workstation makes use of a mechanical indexer to index and dwell a turntable. While dwelling, a cam lifts the part into a mechanism, which spins and routes the disk.

Figure 10-17 Automated routing workstation

Figure 10-18 Automated routing workstation for large disks

Figure 10-19 shows the inner workings of the workstation. An electric motor with a worm gear speed reducer (not shown) drives the indexer. The cam drive shaft is connected to the drive shaft of the indexer with a timing belt. This enables exact timing of the lifting during the dwell of the turntable. Again, a PLC provides procedural machine control over the workstation. CNC technology was used to produce many of the components of the workstation, and the workstation is an example of fixed automation.

Figure 10-19 *Automated routing workstation*

Example 3.4

Figure 10-20 shows two photos of the automated work cell shown previously in Figure 10-2. Recall that this work cell performs two operations: molding of a pressed wood disk and routing of the inside and outside diameter of two different-sized disks. A 4-axis robot serves as the material handler and performs the routing operations by holding a part in the gripper and then lowering it onto the router, as shown in the lower picture. The part is not shown because the photographs were taken during a dry run. Once the part's inside diameter is in contact with the routing tool, the robot rotates the part slightly more than 360 degrees. The robot then lifts the part and moves to engage the outside diameter with the routing tool. Again, it rotates the part slightly more than 360 degrees. Once the routing is complete, the robot then drops the part on the scale for weighing. If the weight is within tolerance, an air cylinder extends to push the part into the finished goods bin. If the part is underweight, a separate air cylinder pushes the part into a scrap bin. This is repeated until all parts are routed.

Figure 10-20 Automated work cell

A PLC provides coordinated system control of the work cell. It coordinates the action of the press, shuttle system, robot, scale, and air cylinders. Recall that this cell processes two different-sized parts. Per molding cycle, three of the parts are 10-inch diameter and two are 8-inch diameter. Consequently, this is an example of flexible automation. The robot recognizes the size of the part it has in its gripper and processes it

accordingly. Such cell flexibility is only achievable with programmable automation as performed with a robot.

10.4 Summary

An automated workstation is defined as a logical grouping of devices, equipment, components, and tools that automatically execute a processing work cycle. It can function unattended for more than one cycle and requires an operator to only periodically tend to it. Manufacturing cells are defined as an interconnected group of manufacturing processes (workstations) tended by a material handling system. To understand the distinction between workstation and work cell keep in mind that workstations are self-contained entities performing one type of operation (similar to a machine) whereas work cells contain multiple workstations performing multiple operations.

Regardless of whether one is referring to an automated workstation or work cell the presence of programmable automation is evident. CNC equipment is often utilized to provide specific machining operations within the cell or workstation. Robots are often the system of choice for material handling. PLCs are used to impart either procedural machine control or coordinated system control.

In addition to programmable automation technologies, workstations and work cells have many other components, including structural members, material handling, part feeders, storage devices, and various drive mechanisms.

10.5 Key Words

automated workstation
centripetal feeder
escapement feeder
fiducials
manufacturing cell
vibratory bowl feeders
workstation

10.6 Review Questions

1. Explain the difference between an automated workstation and an automated work cell.
2. How is an automated workstation different than a manual one?
3. Name four major components of workstations and work cells.
4. Explain the difference between an escapement feeder, vibratory bowl feeder, and centripetal feeder.
5. Referring to question 4, which type of feeder is fastest?

6. List and describe three types of drive mechanisms.
7. Which type of drive mechanism is quiet, low maintenance, and very accurate?
8. Explain the difference between a mechanical pick-and-place unit and a mechanical indexer.
9. Discuss the operation of a Geneva mechanism.
10. In Example 10.4, what is the technology that makes the cell an example of flexible automation?

10.7 Bibliography

1. Groover, M.P. 2001, *Automation, Production Systems and Computer-Integrated Manufacturing,* Second Edition, Prentice Hall, Upper Saddle River, New Jersey
2. Derby, Stephen J. 2005, *Design of Automatic Machinery*, Marcel Dekker, New York.
3. Morriss, S. Brian. 1995, *Automated Manufacturing Systems*, Glencoe/McGraw-Hill, Columbus, Ohio.
4. Rehg, James A. 2003, *Introduction to Robotics in CIM Systems*, Fifth Edition, Prentice Hall, Upper Saddle River, New Jersey.

Acknowledgements

Dedication

For my wife, Tammie and my children—Dan, Ian, and Sydney; to my father Bill, and to my mother Evelyn, in loving memory

INDEX

This page has been reformatted by Knovel to provide easier navigation.

This page has been reformatted by Knovel to provide easier navigation.

Index Terms	Links				
E					
electrical actuators	401	405	407	415	423
electrical power systems	290				
electrical relay	401	405	416		
encoder	87	112	275	290	398
	416				
end effector	261	263	268	277	281
	284	287	294	303	320
	329	371			
end effector (EE) cable	277	290			
end-of-arm tooling	266	282	289		
escapement feeder	495	506			
event-driven change	385	414	416	423	438
Examine_OFF	429	431	434	437	441
	446	450	475	484	
Examine_ON	429	431	434	437	445
	447	450	475	484	
executive processor	290				
F					
F word	124	133	144	164	167
	183				
feed function	124	183			
feed rate	76	78	84	94	105
	112	133	136	142	144
	156	162	164	182	187
	188				
fiducials	495	497	506		
finishing cuts	177	179	183		
fixed automation	13	27	73	110	502
fixed costs	62	69			
fixed position manufacturing system	28				

This page has been reformatted by Knovel to provide easier navigation.

This page has been reformatted by Knovel to provide easier navigation.

This page has been reformatted by Knovel to provide easier navigation.

This page has been reformatted by Knovel to provide easier navigation.

Index Terms	Links				
manual part programming	116	182			
manual prove-out	192	250			
manufacturing	1	13	18	31	37
	41	52	54	62	67
	108	116	183	188	192
	270	277	302	376	378
	384	396	412	417	420
	487	491	506		
manufacturing cell	18	28	44	277	491
	506				
manufacturing lead time	21	23	25	27	
manufacturing operations	3	26	28	118	
manufacturing setup	28				
manufacturing support systems	9	11	13	27	
manufacturing systems	2	4	11	13	27
	41	47	417	487	507
M-code	94	124	153	156	168
	183	217			
mechanical arm	258	261	289	291	
mechanically actuated grippers	266	291			
milling options dialog box	222	228	232	234	236
	240	250			
milling tool dialog box	228	234	237	242	250
miscellaneous function	124	153	164	166	183
modal	125	130	133	136	138
	142	148	152	155	164
	167	182			
modality	125	153	173	182	
motion instructions	294	304	309	318	324
	333	338	343	363	371
motion interval	311	314	319	372	
motion programming	295	297	303	305	363
	371	372			

Index Terms	Links				
motion routines	306	309	316	326	331
	341	367	371		
multiple segment motion	316	318	372		

N

non-contact limit switches	394	416			
non-cutting move	130	183			
non-dimension words	121	123	162	183	
NOT gate	425	485			
nullpointX	223	232	250		
nullpointY	223	232	250		

O

open loop control	87	112	274	290	381
	414	416			
operating cycle	386	416	430	486	
operational cycle time	36	47	68	70	
optical encoder	398	416			
optical proximity switches	394	416			
OR gate	425	485			
orthogonal joint	261	264	291		
output energize	430	433	484	486	
output instructions	372	428	430	438	463
	484	486			
output scan	386	416	430		

P

partial productivity	32	49	54	60	68
	70				
payload capacity	268	273	287	291	
peripheral equipment control	277	290			
pneumatic actuators	401	415			

This page has been reformatted by Knovel to provide easier navigation.

Index Terms	Links			
pneumatic power systems	291			
postprocessor	117	183		
power source	261	269	273	289	291
powered leadthrough	297	372		
preparatory functions	112	119	121	123	126
	130	153	159	183	210
procedural machine control	382	384	414	416	493
	501	503	506		
process interrupts	450	479	486		
process manufacturing system	6	14	27	39	47
process parameters	378	385	401	414	
process variables	328	378	380	391	396
	401	409	414		
product complexity	4	8	14	26	
product definition	4	6	8	26	28
product quantity	4	6	28		
product variety	4	8	14	25	
product volume	6	28	62		
production capacity	20	28	44	68	70
production rate	20	28	33	35	39
	48	54	59	63	65
	68	70			
productivity	2	11	15	21	31
	44	48	53	58	65
	72	107	111	192	260
	280	289			
productivity index	55	69			
program code verification	192	249			
program coordinate sheet	172	183			

Index Terms	Links				

Q

quantify	20	35	48	53	68
	70				
quantity breakeven point	64	69	73		
quantity manufacturing system	6	8	27	47	

R

radius coordinates	232	244	247	249	253
reduction drive	271	273	291		
reset output instruction	438	477	486		
revolving joint	261	291			
risk assessment	285	291			
robot arm positions	294	306	335	371	
robot controller	261	273	277	287	294
	298	301	305	328	330
	338	341	371		
robot language programming	295	303	305	363	371
robot program	275	277	283	288	290
	292	302	304	309	327
	331	334	337	343	366
	371				
robot programming	260	293	301	322	330
	334	366	371		
robot simulation	293	343	372		
RobotAssistTM	343	350	359	363	366
	370				
robotic technology	15	27	280	282	290
rotational joint	261	264	291		
roughing cuts	177	180	183		

Index Terms	Links				
status pane	201	204	207	209	225
	230	232	237	245	249
stepper motor	87	112	381	406	
straight cut	92	112			
switches	152	279	295	297	328
	378	385	389	391	415
	423	437	450		
system shutdown	93	111	119	166	168
	173	183	188	224	
system shutdown section	112	120	168	173	188
	224				

T

T word	124	156	167	183	
tachometer	87	112	275	291	
teach pendant	261	273	279	289	297
	300	303	307	328	331
	350	363	368	371	388
teach pendant programming	303	363	372		
termination	310	315	318	325	338
	371				
text buffer	226	250			
time-driven change	385	414	416	435	450
	472				
timer instruction	431	484	486		
timing diagram	385	409	413	415	420
	422	441	449	484	
toggle switches	279	328	392	415	
tool coordinate system	297	372			
tool function	124	183			
tool handling time	37	70			
tool lists	106	111	118		

This page has been reformatted by Knovel to provide easier navigation.

Index Terms		Links			
tool path	84	89	100	111	116
	130	133	136	142	148
	152	168	171	179	182
	186	192	204	235	244
trajectory	310	316	323	326	343
	371				
transducers	391	394	396	415	
transition	179	302	413	442	472
	475	480	485		
truth tables	424	427	431	484	486
twisting joint	261	264	291		

U

USA principle	31	67	70	408	415
	420				
user coordinate system	300	372			
user interface	109	130	191	194	201
	210	225	249	273	289
	297	344	346	351	393
	453	459	486		
utilization	2	21	23	25	28
	44	68	70	116	

V

vacuum grippers	266	291	
variable costs	62	69	
velocity profile	315	372	
vibratory bowl feeders	495	497	506

Index Terms	Links				
W					
ways	2	68	78	83	110
	112	116	157	202	259
	295	333	371	403	412
word address format	115	120	172	176	182
work cycle program	295	385	409	412	420
	423	427	429	431	433
	439	443	448	470	478
	483				
work envelope	264	283	289	291	298
workpiece coordinate system	96	99	102	105	111
	120	169	172	238	240
workpiece handling time	37	70			
workstation	42	384	393	410	414
	441	451	484	489	493
	497	500	506		
world coordinate system	297	350	356	362	372