CHAPTER 16

GENERAL-PURPOSE CONTROL DEVICES

James H. Christensen
Holobloc, Inc.
Cleveland Heights, Ohio

Robert J. Kretschmann
Rockwell Automation
Mayfield Heights, Ohio

Sujeet Chand
Rockwell Automation
Milwaukee, Wisconsin

Kazuhiko Yokoyama
Yaskawa Electric Corporation
Tokyo, Japan

1 CHARACTERISTICS OF GENERAL-PURPOSE CONTROL DEVICES 678
1.1 Hierarchical Control 678
1.2 Programmability 680
1.3 Device Architecture 681
1.4 Sequential Control 682
1.5 Path Control 686

2 PROGRAMMABLE CONTROLLERS (PLCs) 687
2.1 Principles of Operation 687
2.2 Interfaces 688
2.3 Programming 690
2.4 Programmable Controller Standard, IEC 61131 691

3 NUMERICAL CONTROLLERS 696
3.1 Introduction and Applications 696
3.2 Principles of Operation 698
3.3 Point-to-Point and Contouring Numerical Controllers 701
3.4 NC Programming 702
3.5 Numerical Controllers and CAD/CAM 709

4 ROBOT CONTROLLERS 711
4.1 Composition of a Robot System 711
4.2 Control Program 712

REFERENCES 716

BIBLIOGRAPHY 716

1 CHARACTERISTICS OF GENERAL-PURPOSE CONTROL DEVICES

1.1 Hierarchical Control

As shown in Fig. 1, *general-purpose control devices* (GPCDs) occupy a place in the hierarchy of factory automation above the closed-loop control systems described in Chapters 12–15. The responsibility of the GPCD is the coordinated control of one or more machines or processes. Thus, a GPCD may operate at the ''station'' level, where it controls part or all of a single machine or process, or at the ''cell'' level, where it coordinates the operation of multiple stations.

In fulfilling its responsibilities, the GPCD must be capable of performing the following functions, as shown in Fig. 2:

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

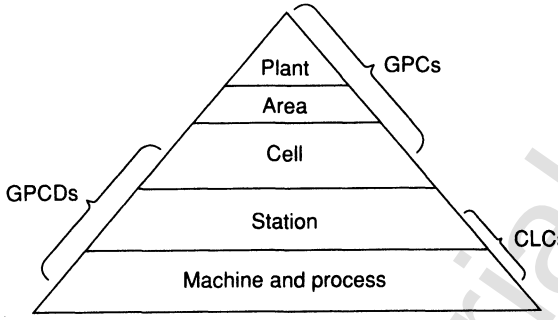


Figure 1 Plant control hierarchy using GPCDs, general-purpose computers (GPCs), and closed-loop controllers (CLCs).

- Issuing commands to and receiving status information from a set of closed-loop controllers that control individual machine and process variables such as velocity, position, and temperature. These closed-loop controllers may be separate devices or integral parts of the GPCD hardware and/or software architecture.
- Issuing commands to and receiving status information from a set of actuators and sensors directly connected to the controlled operation. These actuators and sensors may include signal-processing and transmission elements, as described in Chapter 6.

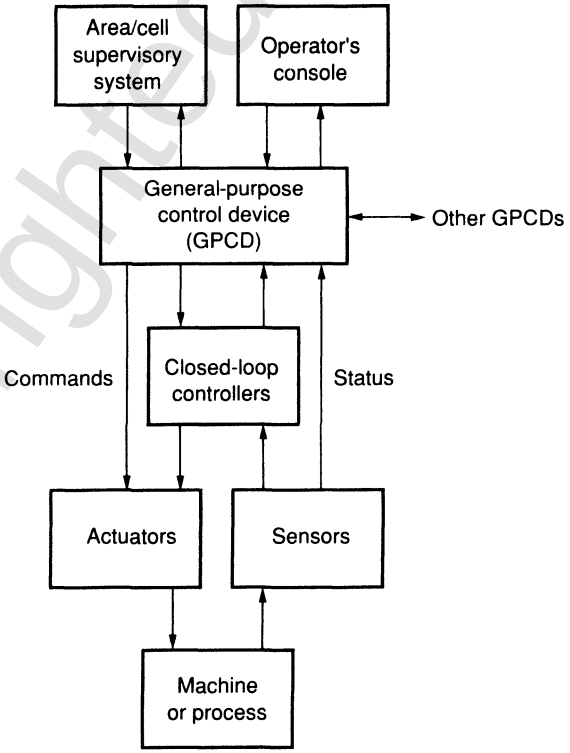


Figure 2 Communication and control paths of a GPCD.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

This capability may not be required if all interface to the controlled operation is through the closed-loop controllers described above.

- Receiving commands from and sending information to a control panel or console for the operator of the machine or process.
- Receiving commands from and sending status information to a manual or automated system with the responsibility of supervising the operation of a number of GPCDs within the boundaries of a ''cell,'' for example, a number of coordinated machines or unit operations, or over a wider ''area,'' such as a chemical process or production zone in a factory.
- Interchanging status information with other GPCDs within the same cell or across cell boundaries.

It should be noted that not all of these capabilities are necessarily required for every application of a GPCD. For instance, special closed-loop controllers may not be necessary in an operation requiring only simple on–off or modulating control. Similarly, communication in an automation hierarchy may not be required for simple ''stand-alone'' applications such as an industrial trash compactor.

However, the capability for expansion into a communicating hierarchy should be inherent in the GPCD architecture if retrofit of stand-alone systems into an integrated production system is considered a future possibility.

1.2 Programmability

To be truly general purpose, a GPCD must be programmable; that is, its operation is controlled by sequences of instructions and data stored in internal memory. The languages used for programming GPCDs are usually *problem oriented:* Programs are expressed in terms directly related to the control to be performed, rather than in a general-purpose programming language such as C++ or BASIC. These languages will be described in appropriate sections for each type of GPCD.

Depending on the application, the responsibility for development and maintenance of GPCD programs may reside with:

- The *original equipment manufacturer* (OEM) of a machine that includes a GPCD as part of its control apparatus
- The *system integrator* who designs and installs an integrated hierarchical control system
- The *end user* who wishes to modify the operation of the installed system

The degree to which the operation of the system can be modified by the end user is a function of:

- The complexity of the system
- The degree to which the end user has been trained in the programming of the system
- The extent to which the operation of the process must be modified over time

For instance, in a high-volume chemical process, only minor modifications of set points may be required over the life of the plant. However, major modifications of the process may be required annually in an automotive assembly plant. In the latter case, complete user programmability of the system is required.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Depending on the complexity of the control program and the degree of reprogrammability required, GPCD programming may be supported by any of several means, including:

- An integral programming panel on the GPCD
- Portable programming and debugging tools
- Minicomputers, personal computers, or engineering workstations that may or may not be connected to the GPCD during control operation
- An online computer system, for example, the area or cell controller shown in Fig. 2.

1.3 Device Architecture

Figure 3 illustrates a GPCD architecture capable of providing the required functional characteristics:

- The *memory* provides storage for the programs and data entered into the system via the *communications processor*.
- The *control processor* performs control actions under the direction of the stored program as well as coordinates the operation of the other functions.
- The *communications processor* provides the means of accepting commands from and providing status information to the supervisory system and operator's console, interchanging status information with other GPCDs, and interacting with program development and configuration tools.

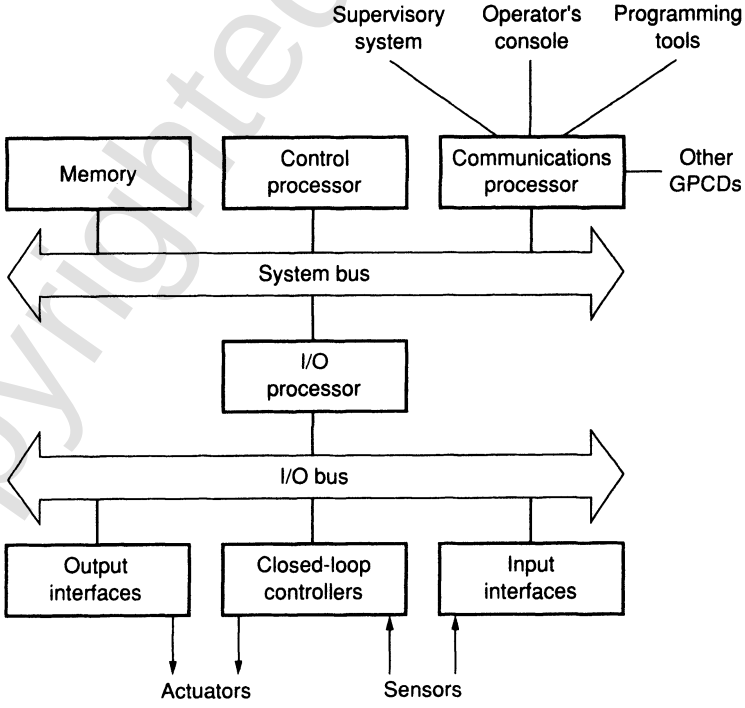


Figure 3 Typical architecture of a GPCD.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

- The *I/O* (*input/output*) *processor* provides the means by which the control processor can issue commands to and receive status information from the closed-loop controllers as well as interchange information directly with the controlled operation via the *output* and *input interfaces*.
- The *system bus* provides for communication among the functional blocks internal to the GPCD, while the *I/O bus* provides for communication between the internal functional blocks and the "outside world" via the closed-loop controllers and I/O interfaces. As an option, the I/O controller may extend the I/O bus functionality to remote locations using data communications methods such as those described in Section 9.

The architecture shown in Fig. 3 is not the only one possible for GPCDs, nor is it necessarily the most desirable for all applications. For instance:

- Large systems may require multiple control processors on the system bus.
- In applications requiring only a few control loops, it may be more economical to perform the closed-loop control function directly in the software of the control processor.
- A separate I/O processor may not be required if the number of separate I/O interface points is less than a few dozen.
- In small systems, the programmer's console may be interfaced directly to the system or I/O bus.

However, if it is anticipated that control system requirements will grow substantially in the future or if total control system requirements are only partially understood, the use of a flexible, extendable GPCD architecture such as that shown in Fig. 3 is recommended.

1.4 Sequential Control

It is obvious that GPCDs must perform complex sequences of control actions when they are applied to the coordination of material handling and machine operation in the fabrication and assembly of discrete parts or in batch and semibatch processes such as blast furnace operation and pharmaceutical manufacture. However, sequential control is also increasing in importance in "continuous" process control, since no process is truly continuous. At the very least, the process must be started up and shut down for maintenance or emergencies by a predetermined sequence of control actions. In large, integrated processes, these sequences are too complicated to be carried out manually and must be performed automatically by GPCDs.

The increasing importance of sequential control, coupled with the increasing complexity of the controlled processes, have generated the need for graphical programming and documentation techniques for the representation of large, complex sequential control plans. These plans must provide a straightforward representation of the relationship between the operation of the control program in the GPCD and the operation of the controlled machine or process as well as the interrelationships between multiple, simultaneous control sequences.

Recognizing this need, the International Electrotechnical Commission (IEC)[1] has undertaken several efforts to standardize the representation of sequential control plans:

- The IEC 60848 standard[2] defines the GRAFCET specification language for the functional description of the behavior of the sequential part of a control system.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

- The IEC 61131-3 standard[3] defines a set of sequential function chart (SFC) constructs, specifically intended to make the IEC 60848 concepts usable for the programming of programmable logic controllers (PLCs).
- The IEC 61512-1 standard[4] defines SFC-like constructs for the description of the sequential operations involved in the control of batch chemical processes.
- The IEC 61499-1 standard[5] defines an execution control chart (ECC) construct which provides for the sequential execution of control algorithms in function blocks for distributed automation systems, similar to the Harel state chart notation of the Unified Modeling Language (UML)[6].

As shown in Fig. 4*a*, an SFC is constructed from three basic types of elements:

- A *step,* representing the current state of the controller and controlled system within the sequential control plan
- A set of associated *actions* at each step
- A *transition condition* that determines when the state of the controller and controlled system is to evolve to another step or set of steps

An SFC consists of a set of independently operating *sequences* of control actions built up out of these basic elements via two mechanisms:

- *Selection* of one of a number of alternate successors to a step based on mutually exclusive transition conditions, as shown in Fig. 4*c*

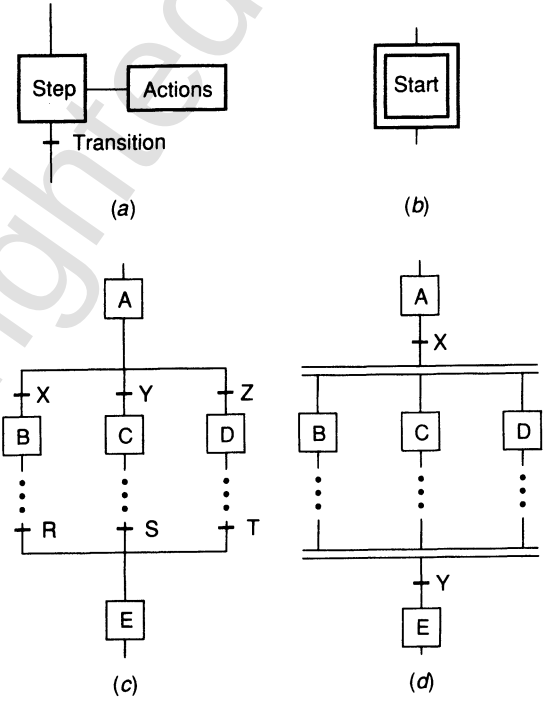


Figure 4 Sequential function chart constructs.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

- *Divergence,* that is, initiation of two or more independently executing sequences based on a transitioncondition, as shown in Fig. 4*d*

The mechanisms for representing convergence, that is, resumption of a main sequence after step selection or parallel sequence initiation, are also shown in Fig. 4*c* and Fig. 4*d*, respectively.

Figure 4*b* illustrates the representation of the *initial step* of each sequence. The operation of control sequences can be visualized by placing a *token* in each initial step upon the initiation of system operation. A step is then said to be *active* while it possesses a token and *reset* when it does not possess a token. The actions associated with the step are performed while the step is active and are not performed when it is reset. The resetting of one or more steps and the activation of one or more successor steps can then be envisioned as the processes of token passing, consumption, and generation, as shown in Fig. 5 and Fig. 6. It should be noted that the selection and convergence of alternate paths within a sequence, as shown in Fig. 5, simply involve the passing of a single token. In contrast, divergence to multiple sequences involves the consumption of a single token and the generation of multiple tokens, as shown in Fig. 6*a*, with the converse operation for termination of multiple sequences shown in Fig. 6*b*.

When an action associated with a step is a Boolean variable, its association with the step may be expressed in an *action block,* as shown in Fig. 7. The qualifiers which can be used to specify the duration of the action are listed in Table 1. More complex actions can be specified via one of the programming languages described in Section 2.3; in this case, the action executes continuously while the associated "action control" shown in Table 1 has the Boolean value "1."

An example of the application of SFCs to the control and monitoring of a single motion, for example, in a robot control system, is given in Fig. 8. Here, the system waits until a motion command is received via the Boolean variable CMD_IN. It then initiates the appropriate motion by asserting the Boolean variable CMD. If a feedback signal DONE is not

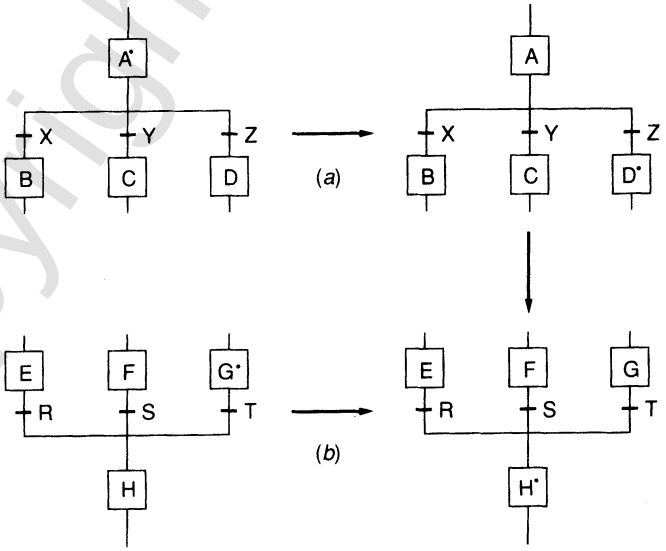


Figure 5 Sequence selection and convergence.

Copyrighted material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

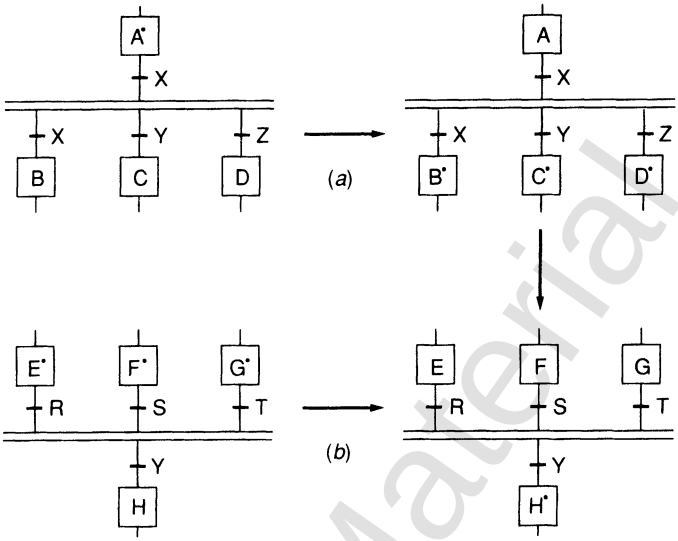


Figure 6 Parallel sequence initiation and convergence.

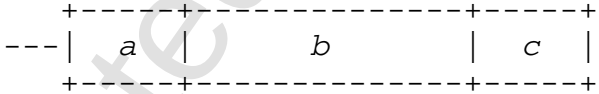


Figure 7 Action block: *a*, action qualifier; *b*, action name; *c*, feedback name.

Table 1 Action Block Qualifiers

Qualifier	Explanation
None	Nonstored (null qualifier)
N	Nonstored
R	Overriding reset
S	Set (stored)
L	Time limited
D	Time delayed
P	Pulse
SD	Stored and time delayed
DS	Delayed and stored
SL	Stored and time limited
P1	Pulse (rising edge)
P0	Pulse (falling edge)

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

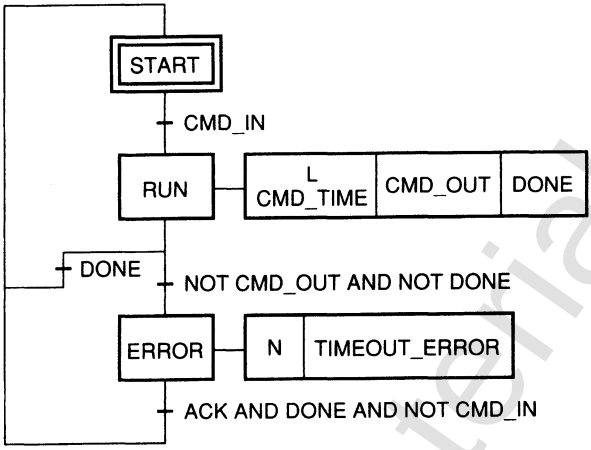


Figure 8 Sequential function chart example.

received within a time interval specified by the variable CMD, the system enters an error step and issues the error message TIMEOUT_ERROR. This error condition is cleared when acknowledged by the signal ACK, for example, from an operator's console, combined with the feedback (DONE) that the motion has been accomplished. The system then reenters the initial step and waits for another command.

1.5 Path Control

General-purpose control devices must often perform *path control,* that is, the coordinated control of several variables at once along a continuous path through time. Typical kinds of path control include:

- The path of a metal-cutting tool or a robot manipulator
- The trajectory of a set of continuous process variables such as temperature, pressure, and composition
- The startup of a set of velocity and tension variables in a paper or steel processing line

The numerical control of metal-cutting tools and robot manipulators is discussed in more detail in Sections 3 and 4, respectively.

A typical application to path control of the general-purpose architecture shown in Fig. 3 has the control processor planning the motions to be accomplished and issuing commands to the closed-loop controllers to perform the required motions. Coordination between the closed-loop controllers may be performed by the control processor, or by direct interaction among the closed-loop controllers, using the I/O bus or special interconnections.

In addition to performing path planning, the control processor also performs sequencing of individual motions and coordination of the motions with other control actions, typically using programming mechanisms such as the SFCs discussed in Section 1.4.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

2 PROGRAMMABLE CONTROLLERS (PLCs)

2.1 Principles of Operation

PLC is defined by the IEC as

> A digitally operating electronic system, designed for use in an industrial environment, which uses a programmable memory for the internal storage of user-oriented instructions for implementing specific functions such as logic, sequencing, timing, counting, and arithmetic, to control, through digital or analog inputs and outputs, various types of machines or processes. Both the PLC and its associated peripherals are designed to be easily integrable into an industrial control system and easily used in all their intended functions.[7]

The hardware architecture of almost all programmable controllers is the same as that for the GPCD shown in Fig. 3.

As illustrated in Fig. 9, the operation of most programmable controllers consists of a repeated cycle of four major steps:

1. All inputs from interfaces and closed-loop controllers on the I/O bus, and possibly from other GPCDs, are scanned to provide a consistent "image" of the inputs.
2. One "scan" of the user program is performed to derive a new "image" of the desired outputs, as well as internal program variables, from the image of the inputs and the internal and output variables computed during the previous program scan. Typically, the program scan consists of:
   a. Determining the currently active steps of the SFC (see Section 1.4), if any, contained in the program.
   b. Scanning the program elements or computing the outputs contained in the active actions of the SFC, if any (if the user program does not contain an SFC, then all program elements are scanned). Scanning of program elements in ladder diagrams or function block diagrams (see Section 2.3) typically proceeds from left to right

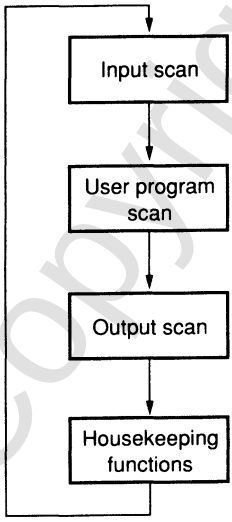


Figure 9 Basic operation cycle of a programmable controller.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

and from top to bottom. Programming elements are sometimes provided to enable skipping the evaluation of groups of program elements or to force the outputs of a group of elements to zero.

c. Evaluating transition conditions of the SFC (if any) at the end of the program scan, in preparation for step 2a in the next program scan.

3. The data from the updated output image are then transferred to the interfaces and closed-loop controllers on the I/O bus and possibly to other GPCDs as well.

4. Finally, ''housekeeping'' tasks are performed on a time-available basis. These typically include communication with the operator, a supervisory controller, a programming terminal, or other GPCDs.

After the performance of housekeeping tasks, the cyclic operation of the PLC begins again with the input scan. This may follow immediately upon execution of the housekeeping tasks or may be scheduled to repeat at a fixed execution interval.

Some programmable controller systems with separate I/O and/or communications processors provide for overlapping the scanning of the user program with the scanning of the inputs (step 1) and outputs (step 3) and communication functions (step 4). In these cases, special programming mechanisms may be needed to achieve concurrency and synchronization between the program and I/O scans and between the program and communications processing.

A further feature of some PLCs is the incorporation of a mutitasking operating system. As in general computing, the PLC operating system serves to coordinate the multitude of hardware and software resources and capabilities of the PLC. The incorporation of a multitasking operating system serves to allow the PLC to essentially execute multiple instances of the basic operation cycle shown in Fig. 9, rather than a single instance, at the same time. This affords much increased flexibility and capability over a single-tasking operating system PLC.

In PLCs incorporating multitasking operating systems, mechanisms may be needed to achieve appropriate levels of task coordination and synchronization between the multiple instances of the basic operating cycle. As an example, one task's results may affect another task's decisions, especially with regard to I/O.

2.2 Interfaces

The IEC has specified the standard voltage ratings shown in Table 2 for power supplies, digital inputs, and digital outputs of programmable controllers. The IEC standard[8] also defines additional parameters for digital inputs and outputs, shown in Tables 3–5; the parameters specified by the manufacturer should be checked against those defined in the IEC standard in order to assure the suitability of a particular input or output module for its intended use in the control system.

The IEC-specified signal ranges for analog inputs and outputs for programmable controllers are shown in Tables 6 and 7, respectively. The IEC standard lists a number of characteristics whose values are to be provided by the manufacturer, such as input impedance, maximum input error, and conversion time and method, and which must be checked against the requirements of the particular control application.

In addition to simple digital and analog inputs and outputs, closed-loop controllers which can reside on the I/O bus of the programmable controller system may be provided, as illustrated for GPCDs in Fig. 2. In this case, the programming languages for the programmable controller typically provide language elements, in addition to those described in Section 2.3, to support the configuration and supervisory control of these ''slave'' closed-loop controllers.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 2 Rated Values and Operating Ranges for Incoming Power Supplies and Digital I/O Interfaces of Programmable Controllers

	Recommended for	
Rated Voltage	Power Supply	I/O Signals
24 V dc[a]	Yes	Yes
48 V dc[a]	Yes	Yes
120 V rms ac[b]	Yes	Yes
230 V rms ac[b]	Yes	Yes

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

[a] Voltage tolerance for dc voltage ratings is −15 to +20%.
[b] Voltage tolerance for ac voltage ratings is −15 to +10%.

Table 3 Rated Values for dc and ac Digital Inputs of Programmable Controllers

Rated Voltage	Types
24 V dc	1–3
48 V dc	1–3
120 V ac	1–3
230 V ac	1–3

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

Table 4 Rated Values for dc Digital Outputs of Programmable Controllers

Rated Current (A)	Maximum Current (A)	Leakage Current (mA)	Input Type Compatibility
0.1	0.12	0.1	1–3
0.25	0.3	0.5	1–3
0.5	0.6	0.5	1–3
1	1.2	1	2, 3
2	2.4	1	2, 3

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

Table 5 Rated Values for ac Digital Outputs of Programmable Controllers

Rated Current (A)	Maximum Current (A)	Leakage Current (mA)
0.25	0.28	5
0.5	0.55	10
1	1.1	10
2	2.2	10

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 6 Rated Values for Analog Inputs of Programmable Controllers

Signal Range	Input Impedance
−10 to +10 V	≥ 10 kΩ
0 to +10 V	≥ 10 kΩ
+1 to +5 V	≥ 5 kΩ
4–20m A	≤ 300 Ω

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

Table 7 Rated Values for Analog Outputs of Programmable Controllers

Signal Range	Load Impedance
−10 to +10 V	≥ 1 kΩ
0 to +10 V	≥ 1 kΩ
+1 to +5 V	≥ 500 Ω
4–20 mA	≤ 600 Ω

Note: See the IEC Programmable Controller standard (Ref. 8) for additional notes and rating values.

Communication interfaces for programmable controllers provide many different combinations of connectors, signal levels, signaling rates, and communication services. The manufacturer's specifications of these characteristics should be checked against applicable standards to assure the achievement of the required levels of system performance and compatibility of all GPCDs in the system.

2.3 Programming

Programmable controllers have historically been used as programmable replacements for relay and solid-state logic control systems. As a result, their programming languages have been oriented around the conventions used to describe the control systems they have replaced, that is, relay ladder logic and function block diagrams. Since these representations are fundamentally graphic in nature, programmable controllers provide one of the first examples of the practical application of graphic programming languages.

Much of the programming of programmable controllers is done in the factory environment while the controlled system is being installed or maintained. Hence, programming was traditionally supported by special-purpose portable programming terminals. In recent years, most of the support for PLC programming has migrated to software packages for personal computers. However, some need still exists for specialized terminals for programming and debugging of programmable controllers in the industrial environment, although most of this functionality can also be supplied by ruggedized, portable personal computers. The selection of support environments is thus an important consideration in the selection and implementation of programmable controller systems.

The standard for PLC programming languages[3] is published by the IEC.[1] This standard specifies a set of mutually compatible programming languages, taking into account the dif-

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

ferent courses of evolution of programmable controllers in North America, Europe, and Japan and the wide variety of applications of programmable controllers in modern industry. These languages include:

- The SFC elements described in Section 1.4 for sequential control
- Ladder Diagrams (LD) for relay replacement functions
- Function Block Diagrams (FBD) for logic, mathematical, and signal-processing functions
- Structured Text (ST) for data manipulation
- Instruction List (IL) for assembly-language-level programming

The IL language will not be described in this book; for further details the IEC standard[3] should be consulted.

Figure 10 shows the application of the LD, FBD, and ST languages to implement a simple command execution and monitoring function. In general, a desired functionality can be programmed in any one of the IEC languages. Hence, languages can be chosen depending on their suitability for each particular application.

An exception to this portability is the use of iteration and selection constructs (`IF. . .THEN. . .ELSIF`, `CASE`, `FOR`, `WHILE`, and `REPEAT`) in the ST language.

The functionality shown in Fig. 10 can be encapsulated into a reusable *function block* by following the declaration process defined in the IEC language standard. An example of the graphical and textual declaration of this functionality is shown in Fig. 11.

In addition to providing mechanisms for the programming of mathematical functions and function blocks, the standard provides a large set of predefined standardized functions and function blocks, as listed in Tables 8 and 9, respectively. The intent is for these to be used as ''building blocks'' for user programs.

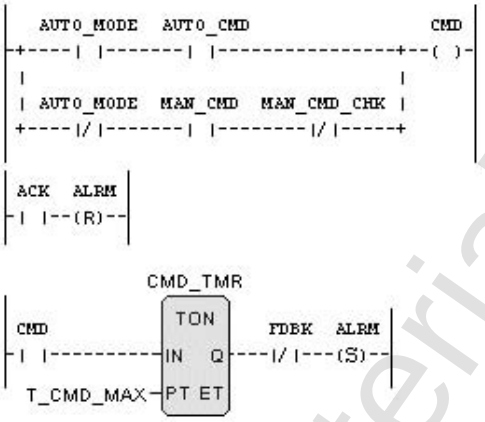
In addition to being used directly for building functions, function blocks, and programs, the LD, FBD, ST, and IL languages can be used to program the ''actions'' to be performed under the control of SFCs as described in Section 1.4. These SFCs can then be used to build programs and reusable function blocks using the mechanisms defined in the IEC language standard.[3]

It will be noted in Figure 11 that data types are defined for all variables. The IEC standard provides facilities for strong data typing, with a large set of predefined data types as listed in Table 10. In addition, facilities are provided for user-defined data types as listed in Table 11. The standard allows manufacturers to specify the language features that they support. Users should consult the standard to determine which language features are required by their application and check their language requirements carefully against the manufacturers' specifications when making their choice of a programmable controller system.

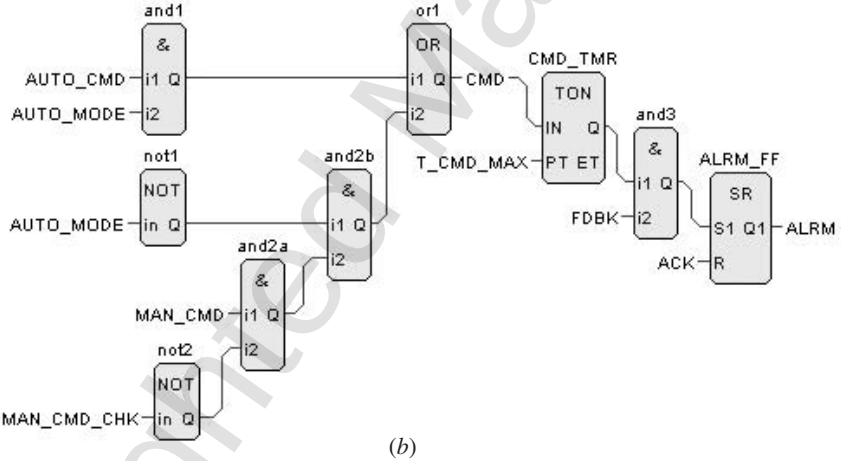
2.4 Programmable Controller Standard, IEC 61131

To propagate consistent characteristics and capabilities for PLCs in the marketplace, an international standard describing them has been developed under the auspices of the IEC.[1] It is impossible, in a chapter of this length, to cover all the hardware, software, and programming language characterisitics, features, and scope defined in the IEC standard for programmable controllers. The standard comprises seven parts[3,7–12] under the general title ''Programmable Controllers,'' covering various aspects of PLCs.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com



(*a*)



(*b*)

```
CMD := AUTO_CMD & AUTO_MODE OR MAN_CMD & NOT MAN_CMD_CHECK & NOT
AUTO_MODE;
CMD_TMR(IN := CMD, PT := T_CMD_MAX);
ALRM_FF(S1 := CMD_TMR.Q & NOT FBDK, R := ACK);
ALRM := ALRM_FF.Q1;
```

(*c*)

Figure 10 Programmable controller programming example.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

```
FUNCTION_BLOCK CMD_MONITOR (* Begin definition of FB CMD_MONITOR *)

  (* Definition of external interface *)

  VAR_INPUT

    AUTO_CMD : BOOL;      (* Automatic Command *)

    AUTO_MODE : BOOL;     (* AUTO_CMD Enable *)

    MAN_CMD : BOOL;       (* Manual Command *)

    MAN_CMD_CHK : BOOL;   (* Negated MAN_CMD for debouncing *)

    T_CMD_MAX : TIME;     (* Maximum time from CMD to FDBK *)

    FDBK : BOOL;          (* Confirmation of CMD completion by operative

    unit *)

    ACK : BOOL;           (* Acknowledgement/Cancel ALRM *)

  END_VAR

  VAR_OUTPUT

    CMD : BOOL;           (* Command to operative unit *)

    ALRM : BOOL;          (* T_CMD_MAX expired without FDBK *)

  END_VAR

  (* Definition of internal state variables *)

  VAR

    CMD_TMR : TON;  (* CMD-to-FDBK timer *)

    ALRM_FF : SR;   (* Note over-riding "S1" input,

                       Command must be cancelled before ACK can cancel

    alarm *)

  END_VAR

(* Definition of Function Block Body per
Figure (a), (b) or (c) *)

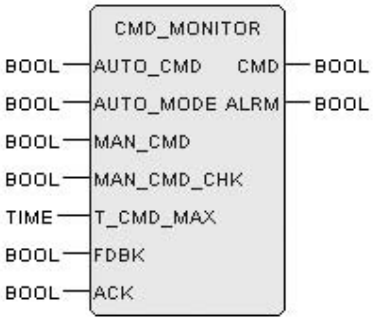
END_FUNCTION_BLOCK  (* End definition of FB CMD_MONITOR *)
```

(*a*)

Figure 11 Function block encapsulation: (*a*) textual declaration in ST language; (*b*) graphic representation.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com



(*b*)

Figure 11 (*Continued*)

Table 8 IEC Standard Functions for Programmable Controllers

Standard Name	Function	Standard Name	Function
Numeric Functions			
ABS	Absolute value	ACOS	Arc cosine
SQRT	Square root	ATAN	Arc tangent
LN	Natural logarithm	ADD or +	Addition
LOG	Logarithm base 10	SUB or −	Subtraction
EXP	Natural exponential	MUL or *	Multiplication
SIN	Sine	DIV or /	Division
COS	Cosine	MOD	Modulo
TAN	Tangent	EXPT or **	Exponentiation
ASIN	Arc sine	MOVE or :=	Assignment
Bit String Functions			
SHL	Shift left, zero filled	SHR	Shift right, zero filled
ROL	Rotate left circular	ROR	Rotate right circular
AND or &	Bitwise Boolean AND	OR or >=1	Bitwise Boolean OR
XOR or =2k+1	Bitwise Boolean exclusive OR	NOT	Bitwise Boolean complement
Selection and Comparison Functions			
SEL	Binary (1 of 2) selection	MUX	Multiplexer (1 of *N*) selection
MIN	Minimum of *N* inputs	MAX	Maximum of *N* inputs
LIM	Hard upper/lower limiter	GT or >	Greater than
GE or >=	Greater than or equal to	EQ or =	Equal to
LE or <=	Less than or equal to	LT or <	Less than
NE or <>	Not equal		
Character String Functions			
CONCAT	Concatenate *N* strings	INSERT	Insert one string into another
DELETE	Delete a portion of a string	REPLACE	Replace a portion of one string with another
NE or <>	Not equal		

Source: From Ref. 3.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 9 IEC Standard Function Blocks for Programmable Controllers

Standard Name	Function
Bistable function blocks	
SR	Flip-flop (set dominant)
RS	Flip-flop (reset dominant)
Edge detection function blocks	
R_TRIG	Rising edge detect
R_TRIG	Falling edge detect
Counter function blocks	
CTU	Up counter
CTD	Down counter
Timer function blocks	
TP	One-shot (pulse) timer
TON	On-delay timer
TOF	Off-delay timer
Message transfer and synchronization	
SEND	Messaging requester
RCV	Messaging responder

Source: From Ref. 3

Table 10 IEC Standard Data Types for Programmable Controllers

Keyword	Data Type	Bits
BOOL	Boolean	1
SINT	Short integer	8
INT	Integer	16
DINT	Double integer	32
LINT	Long integer	64
USINT	Unsigned short integer	8
UINT	Unsigned integer	16
UDINT	Unsigned double integer	32
ULINT	Unsigned long integer	64
REAL	Real number	32
LREAL	Long real	64
TIME	Duration	Implementation dependent
DATE	Date (only)	Implementation dependent
TIME_OF_DAY	Time of day (only)	Implementation dependent
DATE_AND_TIME	Date and time of day	Implementation dependent
STRING	String of 8-bit characters	8*n*
WSTRING	String of 16-bit characters	16*n*
BYTE	Bit string of length 8	8
WORD	Bit string of length 16	16
DWORD	Bit string of length 32	32
LWORD	Bit string of length 64	64

Source: From Ref. 3.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 11 Examples of User-Defined Data Types for Programmable Controllers

Direct derivation from elementary types, e.g.:

```
TYPE RU_REAL : REAL ; END_TYPE
```

Enumerated data types, e.g.:

```
TYPE ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL) ; END_TYPE
```

Subrange data types, e.g.:

```
TYPE ANALOG_DATA : INT (-4095..4095) ; END_TYPE
```

Array data types, e.g.:

```
TYPE ANALOG_16_INPUT_DATA : ARRAY [1..16] OF ANALOG_DATA ; END_TYPE
```

Structured data types, e.g.:

```
TYPE
 ANALOG_CHANNEL_CONFIGURATION :
  STRUCT
   RANGE : ANALOG_SIGNAL_RANGE ;
   MIN_SCALE : ANALOG_DATA ;
   MAX_SCALE : ANALOG_DATA ;
  END_STRUCT ;
 ANALOG_16_INPUT_CONFIGURATION :
  STRUCT
   SIGNAL_TYPE : ANALOG_SIGNAL_TYPE ;
   FILTER_PARAMETER : SINT (0..99) ;
   CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIGURATION ;
  END_STRUCT ;
END_TYPE
```

Source: Ref. 3.

Since technology is always advancing, some part(s) of the standard, at any given time, is (are) being updated. Copies of draft standards are normally available for review from the appropriate National Committees for the IEC. Additionally, experts are always welcome to participate in the standards generation effort on the recommendation of National Committees. Information about the IEC, its National Committees, and ordering of the various parts of the IEC 61131 standard for PLCs is available at the IEC website.[1]

3 NUMERICAL CONTROLLERS

3.1 Introduction and Applications

The century from 1760 to 1860 saw the development of a large number of machine tools for shaping cylindrical and flat surfaces, threads, grooves, slots, and holes of many shapes and sizes in metals. Some of the machine tools developed were the lathe, the planer, the shaper, the milling machine, drilling machines, and power saws. With increasing applications for metal machining, the cost in terms of manpower and capital equipment grew rapidly. The attempt at automation of the metal removal process gave birth to numerical controllers.

The history of numerical controllers dates back to the late 1940s, when John T. Parsons proposed a method to automatically guide a milling cutter to generate a smooth curve. Parsons proposed that successive coordinates of the tool be punched on cards and fed into the machine. The idea was to move the machine in small incremental steps to achieve a

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

desired path. In 1952, the U.S. Air Force provided funding for a project at the Massachusetts Institute of Technology (MIT) that developed the Whirlwind computer. In a subsequent project, the Servomechanisms Laboratory at MIT developed the concept of the first workable numerical control (NC) system. The NC architecture was designed to exploit the Whirlwind computer with emphasis on five-axis NC for machining complex aircraft parts.

The MIT NC architecture identified three levels of interaction with the numerical controller.[13–15] At the highest level is a machine-independent language, called APT (Automatically Programmed Tools). APT provides a symbolic description of the part geometry, tools, and cutting parameters. The next level, called the cutter location (CL) level, changes the symbolic specification of cutter path and tool control data to numeric data. The CL level is also machine independent. The lowest level, called the G-code level, contains machine-specific commands for the tool and the NC axis motions.

The conversion from APT to CL data involves the computation of cutter offsets and resolution of symbolic constraints. The conversion from CL data to G-code is called postprocessing.[13,15] Postprocessing transforms the tool center line data to machine motion commands, taking into account the various constraints of the machine tool such as machine kinematics and limits on acceleration and speed. The APT-to-CL data conversion and the compilation of CL data to G-code are computationally intensive; these computationally intensive functions were envisioned to be performed by the Whirlwind computer. The numerical controller works with simple G-codes to keep computational requirements low. The G-codes, punched on perforated paper tape, would be the input medium to the numerical controller.

Since their inception in the 1950s, numerical controllers have followed a similar pattern in the evolution of controller technology as the computer industry in the past 30 years. The first numerical controllers were designed with vacuum tube technology. The controllers were bulky and the logic inside the control was hard wired. The hard-wired nature of the controller made it very difficult to change or modify its functionality. Vacuum tubes were replaced by semiconductors in the early 1960s. In the early 1970s, numerical controllers started using microprocessors for control. The first generation of numerical controllers with microprocessor technology were mostly hybrid, with some hard-wired logic and some control functions in software. Today, most NC functionality is in the software. Microprocessor-based NC is also called computer numerical control (CNC).

The concept of distributed numerical control (DNC) was introduced in the 1960s to provide a single point of programming and interface to a large number of numerical controllers. Most NC users agree that the paper tape reader on a numerical controller suffers the most in reliability. DNC can transfer a program to NC through a direct computer link, bypassing the paper tape reader. DNC has two primary functions: (1) computer-assisted programming and storage of NC programs in computer memory and (2) transfer, storage, and display of status and control information from the numerical controllers. DNC can store and transfer programs to as many as 100 numerical controllers. Distributed numerical controllers commonly connect to numerical controllers through a link called Behind the Tape Reader (BTR). The name BTR comes from the fact that the connection between DNC and CNC is made between the paper tape reader and the control unit.

The use of paper tape is no longer the main form of storing or updating in CNC. With the advent of computer technology that is now incorporated into CNC, the CNC now has memory storage capability such that the programs are stored in the CNC as files (similar to a computer). Thus, loading new programs or storing old programs is now done via electronic connection through RS232, Ethernet, or memory storage devices such as flash memory cards. DNC is still used, especially in die-cutting machines where programs are too long and the

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

programs are transferred through a ''drip-feed'' technique where the CNC runs the program as the program is downloaded from the DNC.

Numerical controllers are widely used in industry today. The predominant application of NC is still for metal-cutting machine tools. Some of the basic operations performed by machine tools in metal machining are turning, boring, drilling, facing, forming, milling, shaping, and planing.[16] Turning is one of the most common operations in metal cutting. Turning is usually accomplished by lathe machines. The part or workpiece is secured in the chuck of a lathe machine and rotated. The tool, held rigidly in a tool post, is moved at a constant speed along the rotational axis of the workpiece, cutting away a layer of metal to form a cylinder or a surface of a more complex profile.

Applications other than metal machining for numerical controllers include flame cutting, water jet cutting, plasma arc cutting, laser beam cutting, spot and arc welding, and assembly machines.[17]

3.2 Principles of Operation

NC System Components

A block diagram of a NC system is shown in Fig. 12. The three basic components in a NC system are (1) a program input medium; (2) the controller hardware and software, including the feedback transducers and the actuation hardware for moving the tool; and (3) the machine itself.

The controller hardware and software execute programmed commands, compute servo commands to move the tool along the programmed path, read machine feedback, close the servocontrol loops, and drive the actuation hardware for moving the tool. The actuation hardware consists of servomotors and gearing.

The feedback devices on an NC servo system provide information about the instantaneous position and velocity of the NC axes. The servo feedback devices can be linear transducers or rotary transducers. The two most common rotary transducers are *resolvers* and *encoders*.[18] Resolvers consist of an assembly that resembles a small electric motor with a stator–rotor configuration. As the rotor turns, the phase relationship between the stator and rotor voltages corresponds to the shaft angle such that one electrical degree of phase shift corresponds to one mechanical degree of rotation. An optical encoder produces pulsed output that is generated from a disk containing finely etched lines that rotates between an exciter lamp and one or more photodiodes. The total number of pulses generated in a single revolution is a function of the number of lines etched on the disk. Typical disks contain 2000–10,000 lines. A resolver is an analog device with an analog output signal, whereas an encoder is a numerical device that produces a digital output signal.

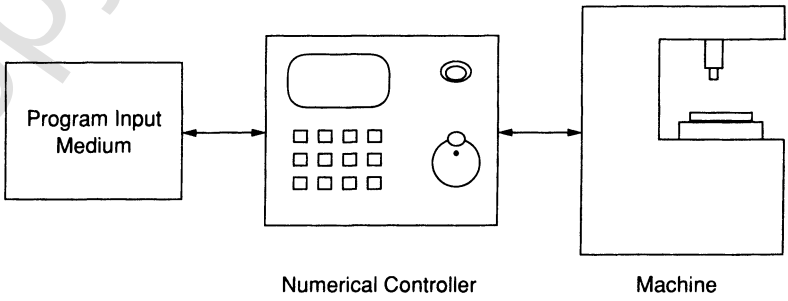


Figure 12 Components of NC system.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Linear transducers may include scales or distance coded markers. Linear transducers have the advantage that they do not introduce backlash errors that a rotary transducer may not be able to detect depending on its mechanical location.

Traditional input media for entering programs into an NC machine include (1) punched cards, (2) punched tape, (3) magnetic tape, and (4) direct entry of the program into the computer memory of the numerical controller.[19] More modern media include flash memory cards.

Originated by Herman Hollerith in 1887, the punched card as an input medium is almost obsolete. The standard ''IBM'' card's fixed dimensions are 3.25 in. wide, 7.375 in. long, and 0.007 in. thick. Each card contains 12 rows of hole locations with 80 columns across the card. To edit a part program, cards in the deck are replaced with new cards. With a deck of cards, it is easy to lose sequence or have missing blocks due to the loss of a card. Also, punched cards are a low-density storage medium with an input rate that is slower than most other media. As a result, punched cards are now regarded as obsolete for NC program media.

Punched tape was for many years the most popular input medium for a numerical controller. Although punched tape is mostly obsolete, many numerical controllers still provide a punched tape reader. The specifications of the punched tape are standardized by the EIA (Electronic Industries Association) and the AIA (Aerospace Industries Association). Tapes are made of paper, aluminum–plastic laminates, or other materials. Making editorial changes to the punched tape is difficult; only minor editing is possible by splicing new data into the tape. With the advent of online computer editing techniques, rapid editorial changes to a program can be made on a computer screen. At the end of an editing session, a tape can be automatically punched on command from the keyboard.

Magnetic tape is not used as much as punched tape because of its susceptibility to pollutants in the NC environment. Dust, metal filings, and oil can cause read errors on the tape. Sealed magnetic tapes overcome some of these problems.

Along with flash memory cards, direct entry of a part program into the controller memory is a common input medium for today's NC. The programmer can either type in the NC program from a keyboard and a video display terminal or generate the NC program from an interactive graphics environment.[20] Part programming with the aid of interactive graphics is discussed in more detail in Section 3.5.

Operation of a Numerical Controller: Machine Coordinate System

NC requires a point of reference and a coordinate system to express the coordinates of parts, tools, fixtures, and other components in the workspace of the machine tool. The commonly used coordinates are three orthogonal intersecting axes of a right-handed Cartesian coordinate frame, as shown in Fig. 13. The rotations *a, b,* and *c* about the *x, y,* and *z* axes are used for NC with more than three axes. In most older numerical controllers, a coordinate frame is marked on the machine and all coordinates are with respect to this fixed frame. In the newer numerical controllers, the machine tool user can program, or ''teach,'' a location for the origin of the reference coordinate system. Since such a reference coordinate system is not permanently attached to the machine, it is sometimes called a *floating coordinate system.*

To illustrate the use of an NC coordinate system, let us consider a simple example of drilling a hole in a rectangular plate with a numerical controller. This example will illustrate the steps in the initial setup and operation of an NC machine. Figure 14 shows the drawing of a simple rectangular plate to be drilled by a drilling machine.

The first step in the programming of any NC operation is getting a drawing of the part. The drawing is usually a blueprint with the dimensions and geometrical attributes of the part. Figure 14 shows a rectangular part and the location of the center of the hole to be drilled in the part. Let the lower left-hand corner of the part be the origin of a two-dimensional Cartesian coordinate frame with the *x* axis and the *y* axis as shown in Fig. 14.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

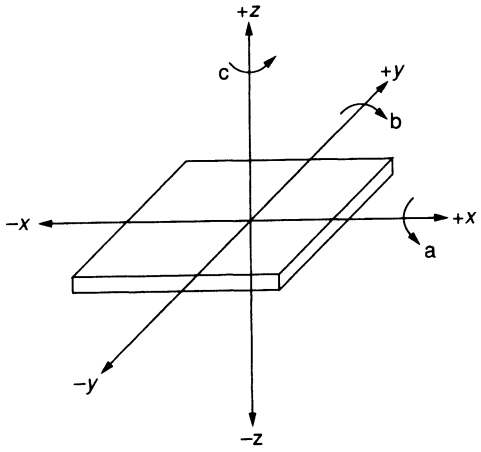


Figure 13 NC coordinate system.

The location of the center of the hole is specified by the coordinates $x = 1.500$ and $y = 2.500$.

The drilling machine must be told the location of the center of the hole. The first step is to establish the location of the NC coordinate system. With the part rigidly held in a fixture, the operator manually moves the machine tool to the lower left-hand corner of the part and presses a button on the machine control panel to teach this point as the origin of the Cartesian coordinate system. Now the machine can locate the center of the hole from the coordinates $x = 1.500$ and $y = 2.500$. In addition to specifying the location of the hole, a programmer can program the rotational speed of the drill, the direction of rotation of the drill (clockwise or counterclockwise), the feed rate, and the depth of cut. The *feed rate* is the distance moved by the tool in an axial direction for each revolution of the workpiece. The *depth of cut* is defined by the thickness of the metal removed from the workpiece, measured in a radial direction.

If the application were milling instead of drilling, the programmer also specifies a cutting speed and a rate of metal removal. The *cutting speed* in a turning or milling operation is the

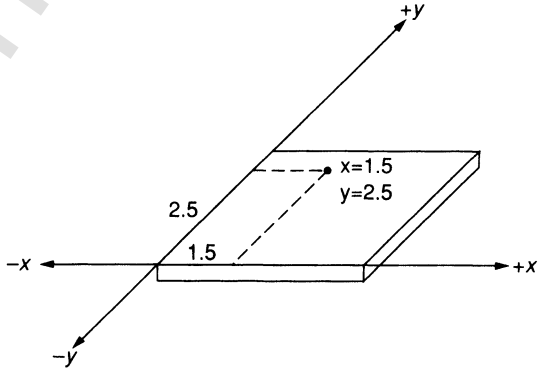


Figure 14 Rectangular plate for drilling.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

rate at which the uncut surface of the workpiece passes the cutting edge of the tool, usually expressed in millimeters per minute or inches per minutes. The *rate of metal removal* is given by the product of the cutting speed, the feed rate, and the depth of cut. The cutting speed and the feed rate are the two most important parameters that a machine operator can adjust to achieve optimum cutting conditions.

3.3 Point-to-Point and Contouring Numerical Controllers

In the example of the preceding section, the drill must be moved to the center of the hole before the drilling operation starts. The drill can start from the origin and traverse to the center of the hole. The machine tool may first move along the x axis by 1.5 in., followed by a movement along the y axis by 2.5 in. It may also simultaneously start moving along the x and y axes. A numerical controller that can position the tool at specified locations without control over the path of the tool between locations is called a *point-to-point* or *positioning controller*.[13] A positioning controller may be able to move along a straight line at 45° by simultaneously driving its axes.

A *contouring controller* provides control over the tool path between positions. For instance, the tool can trace the boundary of a complex part with linear and circular segments in one continuous motion without stopping. For this reason, contouring controllers are also called *continuous-path machines*. The controller can direct the tool along straight lines, circular arcs, and several other geometric curves. The user specifies a desired contour which is typically the boundary or shape of a complex part, and the controller performs the appropriate calculations to continuously drive two or more axes at varying rates to follow this contour.

The specification of a contour in point-to-point NC is tedious because it takes a large number of short connected lines to generate a contour such as a circular arc. The programmer must also provide the appropriate feed rates along these short straight-line segments to control the speed along the contour.

NC Interpolators

In multiaxis contoured NC, multiple independently driven axes are moved in a coordinated manner to direct the tool along a desired path. An interpolator generates the signals to drive the servoloops of multiple actuators along a desired tool path.[21] The interpolator generates a large number of intermediate points along the tool path. The spacing between the intermediate points determines the tool accuracy in tracking the desired path. The closer the points are, the better the accuracy. At each interpolated point along the tool path, the positions of the multiple axes are computed by a kinematic transformation. By successively commanding the NC axes to move from one intermediate point to the next, the tool traces a desired contour with respect to the part.

Older NC systems contained hardware interpolators such as integrators, exponential deceleration circuits, linear interpolators, and circular interpolators. Most of today's numerical controllers use a software interpolator. The software interpolator takes as input the type of curve (linear, circular, parabola, spline, etc.), the start and end points, parameters of the curve such as the radius and the center of a circle, and the speed and acceleration along the curve. The interpolator feeds the input data into a computer program that generates the intermediate target points along the desired curve. At each intermediate target point, the interpolator generates the commands to drive the servoloops of the actuators. The servoloops are closed outside of the interpolator. Several excellent references provide details on closed-loop control of servoactuators.[22,23]

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

3.4 NC Programming

The two methods of programming numerical controllers are *manual programming* and the use of a high-level, computer-assisted *part programming language.*[24] In both cases, the part programmer uses an engineering drawing or a blueprint of the part to program the geometry of the part. Manual programming typically requires an extensive knowledge of the machining process and the capabilities of the machine tool. In manual programming, the part programmer determines the cutting parameters such as the spindle speed and feed rate from the characteristics of the workpiece, tool material, and machine tool.

The most common programming medium for NC machines was formerly perforated tape. EIA standards RS-273-A and RS-274-B define the formats and codes for punching NC programs on paper tape. The RS-273-A standard is called the interchangeable perforated-tape variable-block format for positioning and straight-cut NC systems, while the RS-274-B is the interchangeable perforated-tape variable-block format for contouring and contouring/positioning NC systems. A typical line on a punched tape according to the RS-273-A standard is as follows:

```
n001 g08 x0.0 y1.0 f225 S100 t6322 m03 (EB)
```

where *n* is the sequential block number; *g* is a preparatory code used to prepare the NC for instructions to follow; *x*, *y* are dimensional words or coordinates; *f* is the feed rate; *s* is the code for the spindle rotation speed; *t* is the tool selection code; *m* denotes a miscellaneous function; and (EB) is the end-of-block character, typically a carriage return. The dimension words *x* and *y* can be expanded for machines with greater than two axes. The order of the dimension words for machines with more than two axes is given as *x, y, z, u, v, w, p, q, r, i, j, k, a, b, c, d, e.*

Punched tape can be created manually or by means of a high-level computer programming system. Manual programming is more suited to simple point-to-point applications. Complex applications are usually programmed by a computer-assisted part programming language. The most common computer-assisted programming system for NCs is the APT system.[24] APT programming is independent of the machine-tool-specific parameters such as tool dimensions. Some of the other programming systems are ADAPT, SPLIT, EXAPT, AUTOSPOT, and COMPACT II.

NC programming today is getting less specialized and less tedious with the help of interactive graphics programming techniques. The part programmer creates the NC program from a display of the part drawing on a high-resolution graphics monitor in a user-friendly, interactive environment. Graphical programming is discussed in Section 3.5.

Manual Programming

Manual programming of a machine tool is performed in units called *blocks*. Each block represents a machining operation, a machine function, or a combination of both. Each block is separated from the succeeding block by an *end-of-block code*.

The standard tape code characters are shown in Table 12. *Preparatory functions,* or G-codes, are used primarily to direct a machine tool through a machining operation. Examples are linear interpolation and circular interpolation. *Miscellaneous* functions, or M-codes, are generally on–off functions such as coolant-on, end-of-program, and program stop. Details on the G-codes and the M-codes and their corresponding functions are given in several excellent tutorials on NC programming.[19,24]

The *feed-rate function f* is followed by a coded number representing the feed rate of the tool. A common coding scheme for the feed-rate number is the inverse-time code, which is generated by multiplying by 10 the feed rate along the path divided by the length of the path. If FN denotes the feed-rate number in inverse-time code, FN is given as follows:

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 12 Manual Programming Codes

Character Digit or Code	Description
0	Digit 0
1	Digit 1
2	Digit 2
3	Digit 3
4	Digit 4
5	Digit 5
6	Digit 6
7	Digit 7
8	Digit 8
9	Digit 9
a	Angular dimension around x axis
b	Angular dimension around y axis
c	Angular dimension around z axis
d	Angular dimension around special axis or third feed function
e	Angular dimension around special axis or second feed function
f	Feed function
g	Preparatory function
h	Unassigned
i	Distance to arc center parallel to x
j	Distance to arc center parallel to y
k	Distance to arc center parallel to z
m	Miscellaneous function
n	Sequence number
p	Third rapid traverse direction parallel to x
q	Second rapid traverse direction parallel to y
r	First rapid traverse direction parallel to z
s	Spindle speed
t	Tool function
u	Secondary motion dimension parallel to x
v	Secondary motion dimension parallel to y
w	Secondary motion dimension parallel to z
x	Primary x motion dimension
y	Primary y motion dimension
z	Primary z motion dimension
.	Decimal point
,	Unassigned
Check	Unassigned
+	Positive sign
–	Negative sign
Space	Unassigned
Delete	Error delete
Carriage return	End of block
Tab	Tab
Stop Code	Rewind stop
/	Slash code

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

```
FN = 10*(feedrate along the path) / (length of the path)
```

For example, if the desired feed rate is 900 mm/min and the length of the path is 180 mm, the feed-rate number is 50 (f = 0050). The inverse-time coded feed rate is expressed by a four-digit number ranging from 0001 to 9999. This range of feed-rate numbers corresponds to a minimum interpolation time of 0.06 s and a maximum interpolation time of 10 min.

Spindle speeds are prefixed with the code "s" followed by a coded number denoting the speed. A common code for spindle speed is the *magic-three code*. To illustrate the steps in computing the magic-three code, let us convert a spindle speed of 2345 rpm to magic-three. The first step is to write the spindle speed as 0.2345×10^4. The next step is to round the decimal number to two decimal places and write the spindle speed as 0.23×10^4. The magic-three code can now be derived as 723, where the first digit 7 is given by adding 3 to the power of 10 (4 + 3), and the second two digits are the rounded decimal numbers (23). Similarly, a spindle speed of 754 rpm can be written as 0.75×10^3, and the magic-three code is 675.

Manual programming of the machine tool requires the programmer to compute the dimensions of the part from a fixed reference point, called the origin. Since the NC unit controls the path of the tool center point, the programmer must take into account the dimensions of the machine tool before generating the program. For instance, a cylindrical cutting tool in a milling operation will traverse the periphery of the part at a distance equal to its radius. The programmer also takes into account the limits on acceleration and deceleration of the machine tool in generating a program. For example, the feed rate for straight-line milling can be much higher than the feed rate for milling an inside corner of a part. The tool must be slowed down as it approaches the corner to prevent overshoot. Formulas and graphs are often provided to assist the programmer in the above calculations.

To illustrate manual part programming, let us consider an example of milling a simple part shown in Fig. 15. The programmer initially identifies the origin for the Cartesian co-

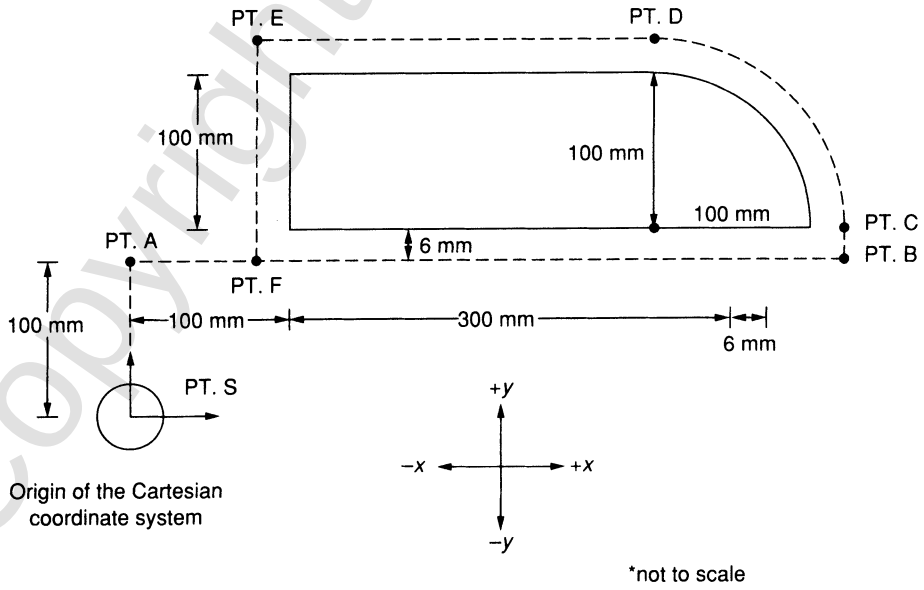


Figure 15 Drawing of a part for machining.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

ordinate system and a starting point for the tool. We will assume that the tool starts from the origin (point PT.S). The programmer builds each block of the program, taking into account the tool diameter and the dynamic constraints of the machine tool. Let us assume that the milling tool in our example is cylindrical with a diameter of 12 mm.

In the first block in the program, the tool starts from rest at point PT.S and moves to point PT.A. Let the desired feed rate be 3000 mm/min. Since the distance from PT.S to PT.A is 100 mm, the inverse-time code for the feed rate is 300. Let the spindle speed be 2000 rpm. The magic-three code for 2000 rpm is 720. We will use a preparatory function code, g 08, for exponential acceleration from rest to the desired feed rate. The first block of the program can now be written as follows:

BLOCK 1 (FROM PT.S TO PT.A)

n	0001	Block 1
g	08	Automatic acceleration
x	0.0	No displacement along the *x* axis
y	100	A displacement of 100 mm along the *y* axis
f	300	Feed rate in inverse-time code
s	720	Spindle speed in magic-three code
m	03	Clockwise spindle rotation
(EB)		End of block

Similarly, the remaining blocks for moving the tool center point along the dotted line of Fig. 15 can be derived as follows. Note that the feed rate and spindle speed need not be programmed for each block if there is no change.

BLOCK 2 (FROM PT.A TO PT.B)

n	0002	Block 2
x	406	Displacement of 406 mm along the *x* axis
y	0.0	No displacement along the *y* axis
m	08	Turn coolant on
(EB)		End of block

BLOCK 3 (FROM PT.B TO PT.C)

n	0003	Block 3
x	0.0	No displacement along the *x* axis
y	6	Displacement of 6 mm along the *y* axis
(EB)		End of block

BLOCK 4 (FROM PT.C TO PT.D)

n	0004	Block 4
g	03	Circular interpolation, counterclockwise
x	106	Displacement of −106 mm along the *x* axis
y	106	Displacement of 106 mm along the *y* axis
i	300	The *x* coordinate of the center of the circle
j	106	The *y* coordinate of the center of the circle
(EB)		End of block

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

BLOCK 5 (FROM PT.D TO PT.E)

n	0005	Block 5
x	206	Displacement of −206 mm along the *x* axis
y	0.0	No displacement along the *y* axis
(EB)		End of block

BLOCK 6 (FROM PT.E TO PT.F)

n	0006	Block 6
g	09	Automatic deceleration
x	0.0	No displacement along the *x* axis
y	112	Displacement of −112 mm along the *y* axis
m	30	Turn off spindle and coolant
(EB)		End of block

These six blocks are entered on a sheet called the *process sheet,* as shown in Fig. 16. The data from the process sheet are punched on a paper tape and input to the NC machine. Notice that each line on the process sheet represents one block of the program. Manual programming as illustrated above can be both tedious and error prone for an inexperienced programmer. In the next section, we illustrate the use of a computer to enter the part program.

Computer-Assisted Programming: Programming in APT

APT was designed to be the common programming language standard for all numerical controllers.[25] APT programs are machine independent. An APT program is a series of English-like statements with a precise set of grammatical rules. Hundreds of keywords embody the huge expanse of NC knowledge into one language. An APT program typically contains process- or part-oriented information. The program does not contain control- and machine-tool-oriented information. APT provides three-dimensional programming for up to five axes.

An APT program containing a description of the part geometry and tool motions is input to a computer program called the *postprocessor*. The postprocessor checks for errors, adds machine-tool-specific control information, transforms the geometric description of the part into tool motion statements, and produces the proper codes for running the machine

PART NUMBER Sample						TAPE NUMBER 2003A					DATE 6/1/88	
PART NAME Example						MACHINE Allen-Bradley 8200					PROGRAMMER S. Chand	
N	G	X	Y	Z	I	J	K	F	S	T	M	COMMENTS
001	08	0.0	100					300	720		03	From PT. S to PT. A
002		406	0.0								08	From PT. A to PT. B
003		0.0	6									From PT. B to PT. C
004	0.3	-106	106		300	106						From PT. C to PT. D
005		-206	0.0									From PT. D to PT. E
006	0.9	0.0	-112								30	From PT. E to PT. F

Figure 16 Process sheet for manual programming.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

tool. Each special machine has its own postprocessor. Most statements in APT are composed of two parts separated by a slash. The word to the left of the slash is called the major word or the keyword and the one to the right is called the minor word or the modifier. For example, COOLNT/OFF is an APT statement for turning the coolant off. Comments can be inserted anywhere in an APT program following the keyword REMARK.

An APT program is generated in two steps. The first step is to program the geometric description of the part. The second step is to program the sequence of operations that defines the motion of the tool with respect to the part. The programmer defines the geometry of the part in terms of a few basic geometric shapes such as straight lines and circles. The geometric description comprises a sequence of connected shapes that defines the geometry of the part. No matter how complex a part is, it can always be described in terms of a few basic geometric shapes.

Step 1: Programming Part Geometry in APT. The programmer programs the geometry of a part in terms of points, lines, circles, planes, cylinders, ellipses, hyperbolas, cones, and spheres. APT provides 12 ways of defining a line; 2 of the 12 definitions are (1) a line defined by the intersection of two planes and (2) a line defined by two points. There are 10 ways of defining a circle—for instance, by the coordinates of the center and the radius or by the center and a line to which the circle is tangent. The first part of an APT program typically contains the definitions of points, lines, circles, and other curves on the part. For the part shown in Fig. 15, an APT program may start as shown in Fig. 17. Note that the line numbers in the leftmost column are for reference only and are not a part of the APT program.

Line numbers 3 and 4 define the inside and outside tolerances in millimeters.[13,15] Line 5 defines the tool diameter as 12 mm. Lines 6 and 7 define the start point and point PT1. Lines 8–12 define the lines and the circular arc in the part geometry. Note that a temporary line, LIN2, is introduced to guide the tool to the corner at point PT2 (Fig. 17). Line 13 turns the spindle on with a clockwise rotation. Line 14 specifies the feed rate, and line 15 turns the coolant on.

Step 2: Programming the Tool Motion Statements in APT. In programming the motion of the tool, the programmer usually starts at a point on the part that is closest to the origin. The programmer then traverses the geometry of the part as viewed from the tool, indicating directions of turn such as `GOLFT` and `GORGT`. Sometimes the analogy of the programmer ''riding'' or ''straddling'' the tool is used to get the sense of direction around the periphery of the part.

An initial sense of direction for the tool is established by the statement INDIRP (''in the direction of a point''). For the part of Fig. 17, let us assume that the tool starts from the point STPT and moves to point PT1. A sense of direction is established by the following statements:

```
FROM / STPT
INDIRP / PT1
```

The statement `GO` is used in startup of the motion. The termination of the motion is controlled by three statements: `TO`, `ON`, and `PAST`. The difference between the statements `GO / TO LIN1`, `GO / ON LIN1`, and `GO / PAST LIN1` is illustrated in Fig. 18.

For the example of Fig. 17, the first GO statement is as follows:

```
GO / TO, LIN1
```

The next statement moves the tool to `PT2`. The sense of direction is set by the previous motion from `STPT` to `PT1`. The tool must turn right at `PT1` to traverse to `PT2`. Also, the

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

```
1     PARTNO A345, Revision 2
2     REMARK Part machined in 3/4 inch Aluminum
3     INTOL/0.00005
4     OUTTOL/0.00005
5     CUTTER/12
6     STPT   =   POINT/0,0,0
7     PT1    =   POINT/0,106,0
8     LIN1   =   LINE/(POINT/0, 106, 0), (POINT/400, 106,
      0)
9     CIRC1  =   CIRCLE/CENTER, (POINT/300, 106, 0),
      RADIUS, 100
10    LIN2   =   LINE/(POINT/400, 206, 0), LEFT, TANTO,
      CIRC1
11    LIN3   =   LINE/(POINT/400, 206, 0), (POINT/100, 206, 0)
12    LIN4   =   LINE/(POINT/100, 206, 0), (POINT/100, 106, 0)
13    SPINDL/ON, CLW
14    FEDRAT/300
15    COOLNT/ON
```

Figure 17 APT programming of part for machining.

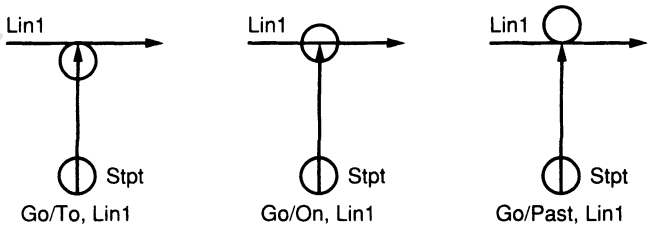


Figure 18 GO statement in APT.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

tool must remain to the right of the line LIN1 as seen from the tool in the direction of motion between PT1 and PT2. This move statement is written as follows:

```
TLRGT, GORGT/LIN1, PAST, LIN2
```

The next step is to move the tool from PT2 to PT3 along the circle CIRC1:

```
TLRGT, GOLFT/CIRC1, PAST, LIN3
```

The next two motion statements to complete the motion around the part are as follows:

```
GOFWD/LIN3, PAST, LIN4
TLRGT, GOLFT/LIN4, PAST, LIN1
```

The complete program is shown below:

```
PARTNO A345, Revision 2
REMARK Part machined in 3/4 inch Aluminum
INTOL/0.00005
OUTTOL/0.00005
CUTTER/12
STPT = POINT/0,0,0
PT1 = POINT/0,106,0
LIN1 = LINE/(POINT/0,106,0), (POINT/400, 106, 0)
CIRC1 = CIRCLE/CENTER, (POINT/300, 106, 0), RADIUS, 100
LIN2 = LINE/(POINT/400, 206, 0), LEFT, TANTO, CIRC1
LIN3 = LINE/(POINT/400, 206, 0), (POINT/100, 206, 0)
LIN4 = LINE/(POINT/100, 206, 0), (POINT/100, 106, 0)
SPINDL/ON, CLW
FEDRAT/300
COOLNT/ON
FROM/STPT
INDIRP/PT1
GO/TO, LIN1
TLRGT, GORGT/LIN1, PAST, LIN2
TLRGT, GOLFT/CIRC1, PAST, LIN3
GOFWD/LIN3, PAST, LIN4
TLRGT, GOLFT/LIN4, PAST, LIN1
STOP
FINI
```

3.5 Numerical Controllers and CAD/CAM

A computer-aided design (CAD) system is used for the creation, modification, and analysis of designs. A CAD system typically comprises a graphics display terminal, a computer, and several software packages. The software packages aid the designer in the creation, editing, and analysis of the design. For instance, in the analysis of a part design, the designer can access a library of routines for finite-element analysis, heat transfer study, and dynamic simulation of mechanisms.

Computer-aided manufacturing (CAM) refers to the use of a computer to plan, manage, and control the operations in a factory. CAM typically has two functions: (1) monitoring and control of the data on the factory floor and (2) process planning of operations on the factory floor.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Since the parts manufactured in a factory are usually designed on a CAD system, the design should be integrated with the programming of the machines on the factory floor. The process of integrating design and manufacturing is often referred to as CAD/CAM.

Today, CAD systems are commonly used to design the parts to be machined by NC. The NC programmer uses a CAD drawing of the part in the generation of the part program. Several numerical controllers provide an interactive graphic programming environment that displays the CAD drawing of the part on a graphics terminal. On the screen of the graphics terminal, a programmer can display the cross section of the part, rotate the part in three dimensions, and magnify the part. These operations help the programmer to better visualize the part in three dimensions; a similar visualization of the part from a two-dimensional blueprint is difficult.

In an interactive graphic programming environment, the programmer constructs the tool path from a CAD drawing. In many systems, the tool path is automatically generated by the system following an interactive session with the programmer. The output can be an APT program or a CLFILE, which can be postprocessed to generate the NC punched tape.[20]

There are two basic steps in an interactive graphics programming environment:

1. *Defining the Geometry of the Part.* The geometric definition of the part can be specified during the part design process in a CAD/CAM system. If the geometric definition does not exist, the part programmer must create it on the graphics terminal. This process is interactive, with the part programmer labeling the various edges and surfaces of the part on the graphics screen. After the labeling process is complete, the system automatically generates the APT geometry statements for the part. The definition of the part geometry is much more easily performed on the interactive graphic system than by the time-consuming, error-prone, step-by-step manual process.
2. *Generating the Tool Path.* The programmer starts by defining the starting position of the cutter. The cutter is graphically moved along the geometric surfaces of the part through an interactive environment. Most CAD/CAM systems have built-in software routines for many machining operations, such as surface contouring, profile milling around a part, and point-to-point motion. The programmer can enter feed rates and spindle speeds along each segment of the path. As the program is created, the programmer can visually verify the tool path with respect to the part surface on the graphics screen. The use of color graphics greatly aids in the visual identification of the part, tool, and program parameters.

The advantages of the CAD/CAM approach to NC programming are as follows[26]:

1. The geometric description of the part can be easily derived from the CAD drawing.
2. The programmer can better visualize the part geometry by manipulating the part drawing on the screen, such as rotating the part and viewing cross sections.
3. The interactive approach for the generation of part programs allows the user to visually verify a program on the graphics screen as it is being created.
4. The programmer has access to many machining routines on the CAD system to simplify tool path programming.
5. The CAD environment allows the integration of part design and tool design with programming.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

4 ROBOT CONTROLLERS

The robot controller is a device that, via a pretaught software program, controls the operation of the manipulator and peripheral equipment of a robot. This section explains the composition of a typical robot system and the procedure for developing its working program.

4.1 Composition of a Robot System

As shown in Fig. 19, a basic industrial robot system is composed of four components: tools, manipulators, the controller, and the teach pendant. The tool is attached at the end point of the manipulator. The type of tool depends on the work to be done by the manipulator. The manipulator maintains the position and the orientation of the tool. The controller controls the manipulator's motion, tools, and peripherals. The teach pendant is the human–machine interface for the robot controller. The movement of the manipulator and the attached tools are programmed by using the teach pendant.

Tools

The general industrial robot, which is without a dexterous humanlike hand, works by installing a suitable tool for each job in the arm end. Some applications require the use of peripheral equipment, such as power supply or parts supply for the tools. The robot controller can control the operation of peripheral equipment along with the movement of the manipulator. This subject will be described at length later in this section. Table 13 shows the relation between work and the tool. For multiprocess jobs such as assembly tasks, an auto-

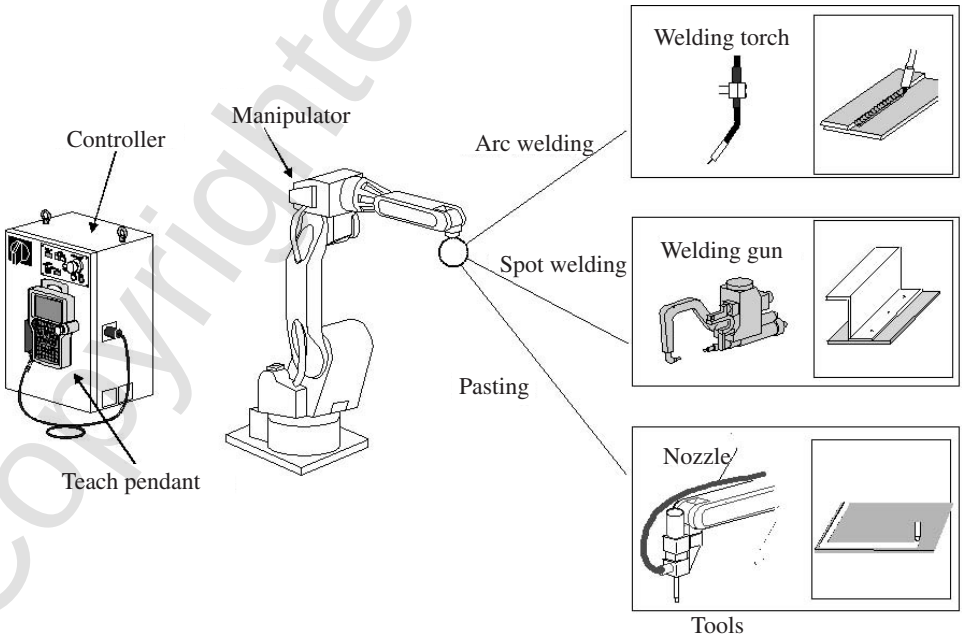


Figure 19 Basic industrial robot system.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 13 Relation between Work and Tool

Work	Tool	Peripheral Equipment	Main Controlled Items
Arc welding	Welding torch	Welding power supply, wire supply device	Welding voltage, welding current, arc on/off
Spot welding	Welding gun	Welding power supply	Welding condition, gun on/off, pressurizing power
Pasting/sealing	Nozzle	Paste (seal material) supply device	Gun on/off
Parts transportation	Gripper		Opening and closing
Assembly	Screw driver	Tool exchange device	Tool on/off

matic tool exchange device is necessary to change tools for each individual process. The robot controller also controls the tool exchange device.

Manipulator

The role of the manipulator is to maintain the tool position and orientation. Most industrial robot arms now are very similar to a human arm in that they have a joint. The joint is driven by an actuator in the form of a servomechanism. Actuators include ac servomotors, dc servomotors, and oil pressure motors, for example. In general, industrial robots use six actuators (thereby have six degrees of freedom), but there are some cases where an industrial robot will require less than six. The manipulator needs tools and transfer devices in order to hold and transport payloads.

Controller

As explained earlier, the controller is a device which, using preprogrammed (or ''taught'') control programs, controls the position and orientation of the manipulator and attached tools. This will be explained in more detail later. A general controller includes the functions to interpret and execute the control program, I/O functions (digital I/O and analog I/O) to control the tools and peripheral equipment, and functions to control servodrives for each actuator.

As shown in Fig. 20, the configuration of a robot controller follows the general scheme of all GPCDs illustrated in Fig. 3, with specialized interfaces as required for the robot control and programming functions. Of particular note is the increasing use of Internet or local-area network (LAN) communication interfaces on the controller, because the robot is frequently working simultaneously with many robots in the factory and also because of the use of remote operation and remote maintenance.

Teach Pendant

The teach pendant is the controller's human–machine interface for making the robot's control program. The screen on which the status of the manipulator and the control program are displayed and the button to operate the manipulator and to teach the position of each joint into the memory of the controller are arranged on the pendant.

4.2 Control Program

The control program is programmed using the teach pendant to make the robot work as described above. The control program is composed of commands, such as those to drive the

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

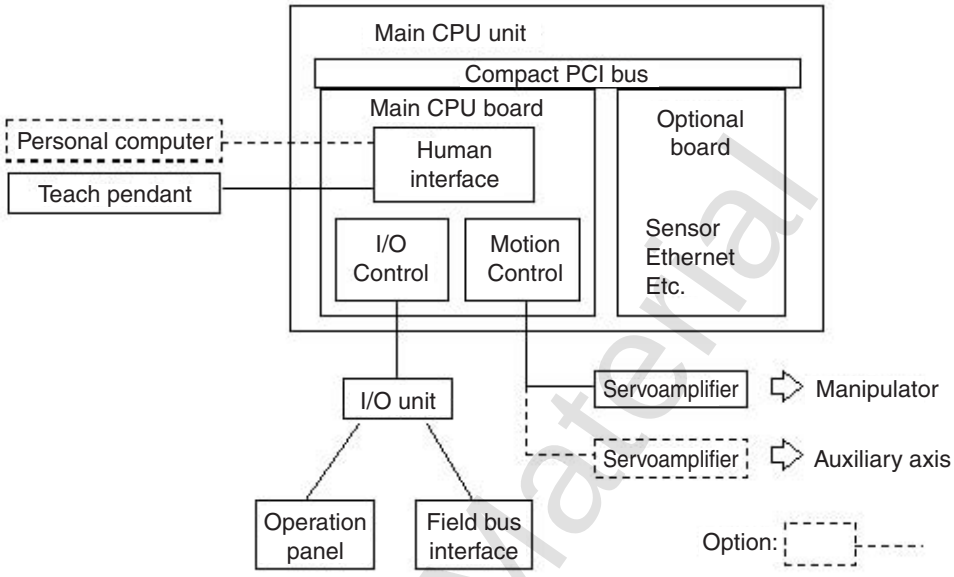


Figure 20 Composition of a robot controller.

controller itself or the manipulator. The program saved in the memory of the controller controls the manipulator and executes the software processing, such as external I/O processing or communication processing, by a set of output signals.

Just as with a computer program, very difficult tasks are made possible by control programs that can be changed through either a change in commands or the creation of subroutines via a series of commands. The basic programming method is to teach the position and the posture of the tool according to the work to be done. For jobs such as arc welding, pasting, and sealing, it is necessary to teach the trajectory of the movement of the tool. In such a case, the controller has a function (interpolation function) to automatically generate any trajectory while the control program is executed by teaching only the starting point and ending point for a straight line and/or the starting point, ending point, and intermediate point for any curve, instead of teaching the entire trajectory directly. Through this function the position and orientation of the tool can be moved. When teaching the trajectory, the speed at which the tool will be moved through the trajectory is very important; therefore, the speed of the tool is taught along with the position and orientation. For example, consider the simple arc welding of two parts shown in Fig. 21. The tool is an arc welding torch, and the peripheral equipment is a welding power supply.

A series of jobs begin with the positioning of the torch at the start point. Next, the arc welding power supply is turned on. To get the best value from the welding, the torch end matches the parts and welds them together while controlling the welding voltage and the current. The arc welding power supply is turned off when reaching the position where the welding is finished. To create this program, the manipulator is moved while the welding torch position and orientation are changed by the teach pendant. The teaching is done as follows:

1. Standby position (P1)
2. Welding start point vicinity (P2)
3. Welding start point (P3)

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

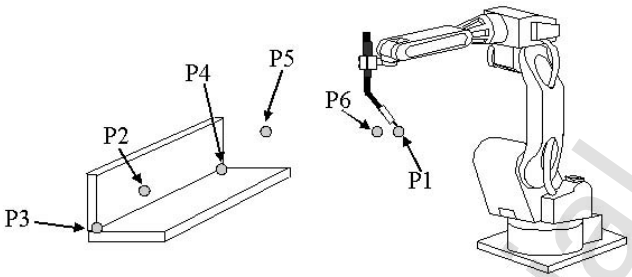


Figure 21 Programming an arc welding robot.

4. Welding end point (P4)
5. Welding end point vicinity (P5)
6. Standby position (P6)

Two teaching points, the start point (P3) and the end point (P4), are necessary for the arc welding. However, supplementary teaching points should be included in order to assure the manipulator does not interfere with the change in the work object or to assure that there is not interference between the work object and the tool. In an actual work situation, it is necessary to teach more supplementary points according to the work environment. The result of the teaching is shown in Table 14 as commands of a robot language. In step 4, the straight-line trajectory is generated automatically, at 1200 mm/min. In an actual situation commands are added using the teach pendant.

After the motion of the manipulator is taught, the control of the peripheral equipment is taught. In this case, when moving from P3 to P4, it is necessary to control the arc welding power supply. Table 15 shows the robot language program.

In the example above, a robot language was used to explain the procedure for creating a robot control program. The teaching method of actually moving the manipulator with the teach pendant is called *online programming*. On the other hand, the method by which the robot is not actually used in making the program but via computer graphics and a robot simulation is called *off-line programming*.

An off-line program can make the control program when an actual manipulator cannot be used, such as the production line design period, and is effective, for example, in the examination of the line arrangement. Online programming and off-line programming are the same as the basic instruction commands shown previously. Figure 22 shows an example screen of a PC-based off-line programming system.

Table 14 Initial Robot Program

```
Step commands
1: MOVEP1
2: MOVEP2
3: MOVEP3
4: MOVE LINEAR P4 SPEED=1200mm/min
5: MOVE P5
6: MOVE P6
```

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

Table 15 Robot Program with Peripheral Equipment Control

```
Step commands
01: MOVE P1
02: MOVE P2
03: MOVE P3
04: OUT ANALOG PORT=0 VALUE=<some value>
05: OUTANALOGPORT=1VALUE=<some value>
06: OUT DIGITAL PORT=0 VALUE=ON
07: MOVE LINEAR P4 SPEED=600mm/sec
08: OUT DIGITAL PORT=0 VALUE=OFF
09: MOVE P5
10: MOVE P6
```

Copyrighted Material

Acknowledgments

The authors gratefully acknowledge the contributions of Steven Tourangeau of Rockwell Automation to Section 3 as well as the contributions to the previous edition of this chapter by the late Odo J. Struger of Rockwell Automation.

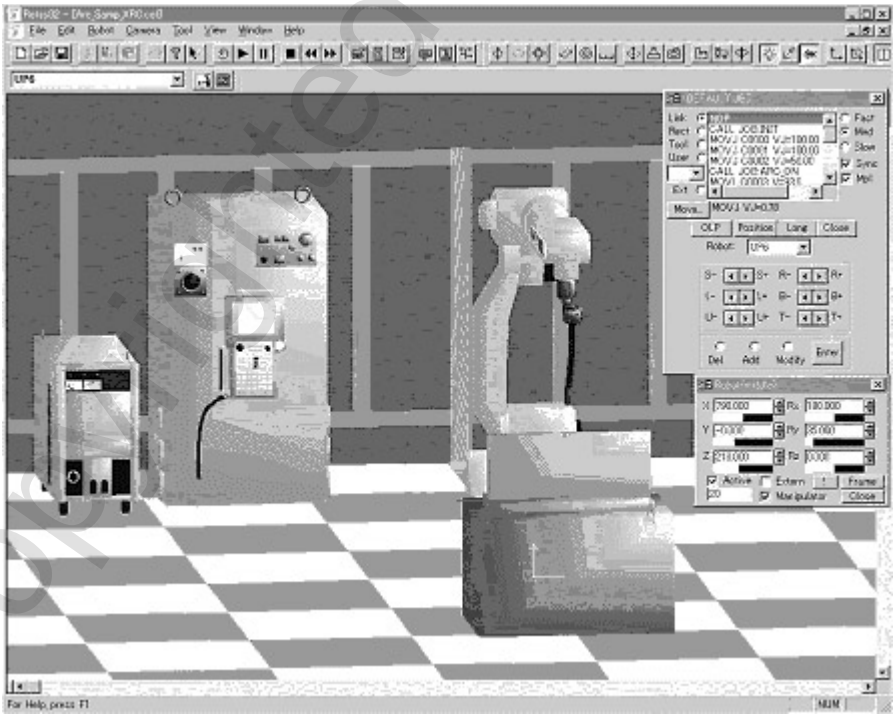


Figure 22 Example screen of an off-line programming system.

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com

REFERENCES

1. http://www.iec.ch.
2. IEC 60848, *GRAFCET Specification Language for Sequential Function Charts,* 2nd ed., International Electrotechnical Commission, Geneva, 2002.
3. IEC 61131-3, *Programmable Controllers—Part 3: Programming Languages,* 2nd ed., International Electrotechnical Commission, Geneva, 2003.
4. IEC 61512-1, *Batch Control—Part 1: Models and Terminology,* 1st ed., International Electrotechnical Commission, Geneva, 1997.
5. IEC 61499-1, *Function Blocks for Industrial-Process Measurement and Control Systems—Part 1: Architecture,* 1st ed., International Electrotechnical Commission, Geneva, 2004.
6. B. P. Douglass, *Real-Time UML,* Addison-Wesley, Reading MA, 1998.
7. IEC 61131-1, *Programmable Controllers—Part 1: General Information,* 2nd ed., International Electrotechnical Commission, Geneva, 2003.
8. IEC 61131-2, *Programmable Controllers—Part 2: Equipment Requirements and Tests,* 2nd ed., International Electrotechnical Commission, Geneva, 2004.
9. IEC 61131-4, *Programmable Controllers—Part 4: User Guidelines,* 2nd ed., International Electrotechnical Commission, Geneva, 2004.
10. IEC 61131-5, *Programmable Controllers—Part 5: Communications,* 1st ed., International Electrotechnical Commission, Geneva, 2000.
11. IEC 61131-7, *Programmable Controllers—Part 7: Fuzzy Control Programming,* 1st ed., International Electrotechnical Commission, Geneva, 2000.
12. IEC 61131-8, *Programmable Controllers—Part 8: Guidelines for the Application and Implementation of Programming Languages,* 2nd ed., International Electrotechnical Commission, Geneva, 2003.
13. R. S. Pressman and J. E. Williams, *Numerical Control and Computer-Aided Manufacturing,* Wiley, New York, 1977.
14. W. Leslie, *Numerical Control Users Handbook,* McGraw-Hill, New York, 1970.
15. Y. Koren, *Computer Control of Manufacturing Systems,* McGraw-Hill, New York, 1983.
16. E. M. Trent, *Metal Cutting,* Butterworths, Woburn, MA, 1977.
17. G. Boothroyd, C. Poli, and L. E. Murch, *Automatic Assembly,* Marcel Dekker, New York, 1982.
18. A. Fitzgerald and C. Kingsley, *Electric Machinery,* 2nd ed., McGraw-Hill, New York, 1958.
19. J. Childs, *Numerical Control Part Programming,* Industrial Press, New York, 1973.
20. M. Groover, and E. Zimmers, *CAD/CAM Computer-Aided Design and Manufacturing,* Prentice-Hall, Englewood Cliffs, NJ, 1984.
21. Y. Koren, A. Shani, and J. Ben-Uri, ''Interpolator for a CNC System,'' *IEEE Transactions on Computers,* **C-25**(1), 32–37 (1976).
22. General Electric, *Pulse Width Modulated Servo Drive,* GEK-36203, March 1973.
23. J. Beckett and G. Mergler, ''Analysis of an Incremental Digital Positioning Servosystem with Digital Rate Feedback,'' *ASME Journal of Dynamic Systems, Measurement and Control,* **87,** March 1965.
24. A. Roberts and R. Prentice, *Programming for Numerical Control Machines,* 2nd ed., McGraw-Hill, New York, 1978.
25. Illinois Institute of Technology Research Institute (IITRI), *APT Part Programming,* McGraw-Hill, New York, 1967.
26. D. Grossman, ''Opportunities for Research on Numerical Control Machining,'' *Communications of the ACM* **29**(6), 515–522 (1986).

BIBLIOGRAPHY

Hughes, T. A., *Programmable Controllers,* 4th ed., ISA, Raleigh, NC, 2004.

Lewis, R., *Modelling Distributed Control Systems Using IEC 61499,* IEE, London, 2001.

Lewis, R. W., *Programming Industrial Control Systems Using IEC 1131-3, rev. ed.,* IEE, London, 1998.

Copyrighted Material

Copyright © 2006 John Wiley & Sons Retrieved from: www.knovel.com