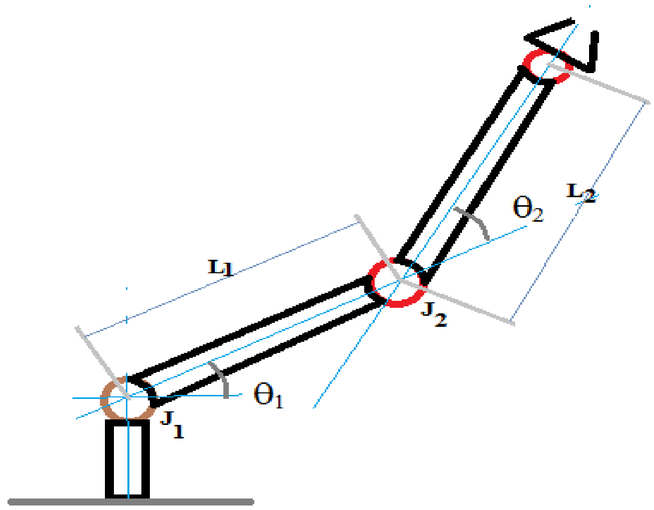
UNIT 4 : ROBOT KINEMATICS & ROBOT PROGRAMMING

FORWARD KINEMATICS: It is a scheme to determine joint angles of a robot by knowing its position in the world coordinate system.

REVERSE KINEMATICS: It is a scheme to determine the position of the robot in the world coordinate system by knowing the joint angles and the link parameters of the robot.

THE FORWARD AND REVERSE TRANSFORMATION OF 2-DEGREE OF FREEDOM

Forward transformation of a 2- degree of freedom arm



We can determine the position of the end of the arm in world space by defining a vector for link 1 and another for link 2.

$$r_1 = [L_1 \cos \theta_1, L_1 \sin \theta_1] \text{ -----eq1}$$

$$r_2 = [L_2 \cos (\theta_1 + \theta_2), L_2 \sin (\theta_1 + \theta_2)] \text{ ------eq2}$$

Vector addition of eq1 and eq2 yields the coordinates x and y of the end of the arm point P_W in world space

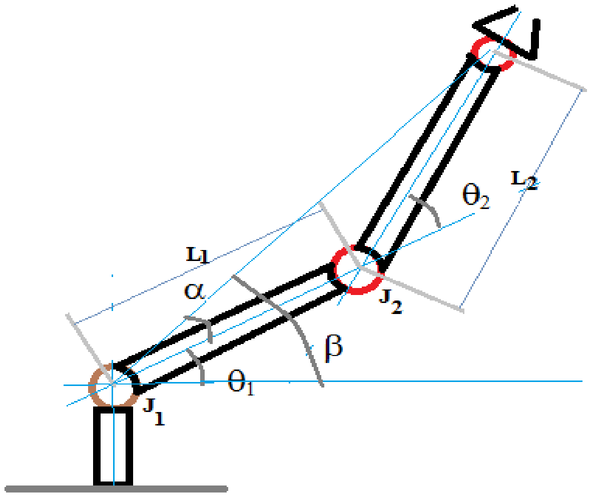
$$X = L_1 \cos \theta_1 + L_2 \cos (\theta_1 + \theta_2) \text{ ----eq3}$$

$$Y = L_1 \sin \theta_1 + L_2 \sin (\theta_1 + \theta_2) \text{ ----eq4}$$

Reverse transformation of a 2- degree of freedom arm

It is more important to be able to derive the joint angles given the end of the arm position in the world space. The typical situation is where the robot's controller must compute the joint angles required to move its end of arm to a point in space defined by the points coordinates.

Consider RR robot



We can find the angles θ_1 and θ_2 by using trigonometric identities
Cos (A+B) = cos A cos B – sin A sin B
Sin (A+B) = sin A cos B + sin B cos A

$$\tan (A\text{-}B) = \frac{TAN\ A + TAN\ B}{1 - TAN\ A\ TAN\ B}$$

We can re write the equations 3 and 4

$$X = L_1 \cos \theta_1 + L_2 \cos \theta_1 \cos \theta_2 - L_2 \sin \theta_1 \sin \theta_2 \text{ ----eq5}$$

$$Y = L_1 \sin \theta_1 + L_2 \sin \theta_1 \cos \theta_2 + L_2 \sin \theta_2 \cos \theta_1) \text{ ----eq6}$$

Squaring and adding the equations 5 and 6

$$cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}$$

Defining α and β

$$\text{Tan } \beta = \frac{y}{x}$$

$$\text{Tan } \alpha = \frac{L_2\, SIN\ \theta_2}{L_1 + L_2\, COS\theta_2}$$

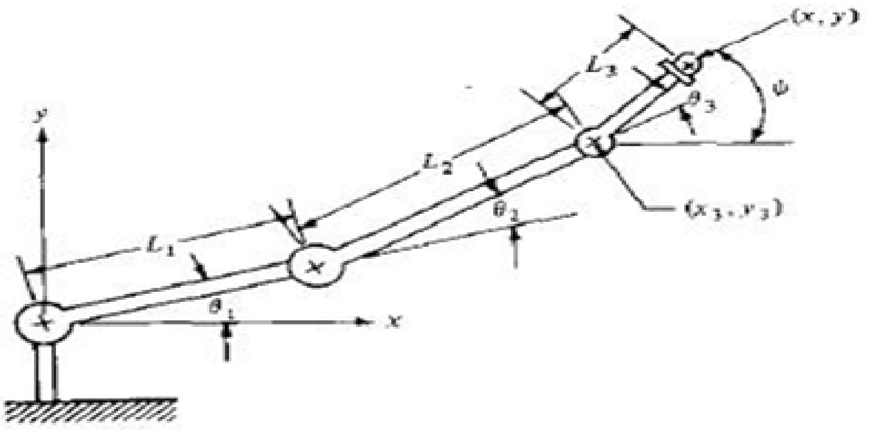
$$\text{Tan } \theta_1 = \text{Tan } (\beta\text{-}\alpha) = \frac{TAN + TAN}{1 - TAN\ \ TAN} = \frac{\frac{Y}{X} + \frac{L_2\, SIN\ \theta_2}{L_1 + L_2\, COS\theta_2}}{1 - \frac{Y}{X}\frac{L_2\, SIN\ \theta_2}{L_1 + L_2\, COS\theta_2}}$$

$$= \frac{Y(L_1 + L_2 COS\theta_2) + X(L_2 SIN\ \theta_2)}{X(L_1 + L_2 COS\theta_2) - Y(L_2 SIN\ \theta_2)}$$

We know the link lengths L_1 and L_2 and arm end position P(x, y) can able to calculate the joint angles

A 3-DEGREE OF FREEDOM ARM IN TWO DIMENSIONS

The arm we have been modeling is very simple; a two-jointed robot arm has little practical value except for very simple tasks.
Let us add to the manipulator a modest capability for orienting as well as positioning a part or tool. Accordingly, we will incorporate a third degree of freedom into the previous configuration to develop the RRR manipulator shown in Figure.



This third degree of freedom will represent a wrist joint. The world space coordinates for the wrist end would

$$X = L_1 \cos \theta_1 + L_2 \cos (\theta_1 + \theta_2) + L_3 \cos (\theta_1 + \theta_2 + \theta_3)$$

$$Y = L_1 \sin \theta_1 + L_2 \sin (\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

$$\psi = \theta_1 + \theta_2 + \theta_3$$

The reverse transformation of three degree of freedom of arm . when defining the position of end of the arm p(x, y) and ψ is the orientation angle for the wrist.
We can solve the joint angles θ_1, θ_2, θ_3 using

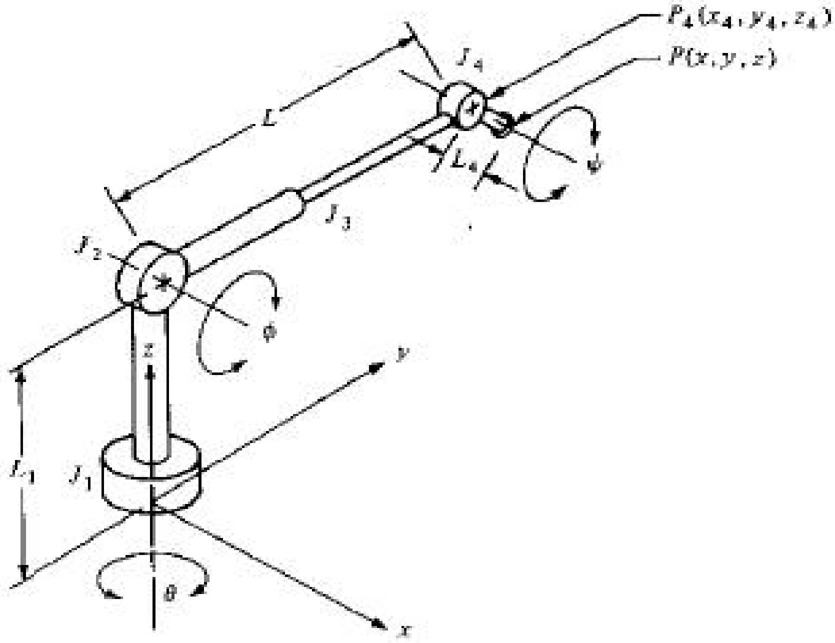
$$X_3 = X - L_3 \cos \psi$$

$$Y_3 = Y - L_3 \sin \psi$$

A 4-DEGREE OF FREEDOM MANIPULATOR IN THREE DIMENSIONS:

The configuration of a manipulator in three dimensions. The manipulator has 4 degrees -of freedom: joint 1 (type T joint) allows rotation about the z axis; joint 2 (type R) allows rotation about an axis that is perpendicular to the z axis; joint 3 is a linear joint which is capable of sliding over a certain range; and joint 4 is a type R joint which allows rotation about an axis that is parallel to the joint 2 axis. Thus, we have a TRLR manipulator.

Let us define the angle of rotation of joint j_1 about z axisθ; the angle of rotation of joint 2 will be called the elevation angleΦ; the length of linear joint 3 will be called the extension L (L represents a combination of links 2 and 3); and the angle that joint 4 makes with the x y plane will be called the pitch angleψ. These features are shown in figure.



Forward kinematics: The position of the end of the wrist, P, defined in the world coordinate system for the robot, is given by

$$X = \cos\theta\,(L\cos\Phi + L_4\cos\psi)$$

$$Y = \sin\theta\,(L\cos\Phi + L_4\cos\psi)$$

$$Z = L_1 + L\sin\Phi + L_4\sin\psi)$$

Reverse kinematics: Given the specification of point P(x, y, z) and pitch angle ψ we can find any of the joint positions relative to the world coordinates system. Using P_4 (x_4, y_4, z_4), which is the position of joint 4.

$$X_4 = x - \cos\theta\,(L_4\,\mathrm{Cos}\psi)$$

$$Y_4 = y - \sin\theta\,(L_4\,\mathrm{Cos}\psi)$$

$$Z_4 = z - L_4\sin\psi$$

The values of L, Φ and θ can be computed

$$L = (X_4^2 + Y_4^2 + (Z_4 - L_1)^2)^{-1}$$

$$\mathrm{Sin}\,\Phi = \frac{z_4 - L_1}{L}$$

$$\mathrm{COS}\,\theta = \frac{Y_4}{L}$$

ROBOT PROGRAMMING TECHNIQUES

A number of different techniques are used to program robots. The principal task of robot programming is to control the motions and actions of the manipulator. A robot is programmed by entering the programming commands into its controller memory. The methods of entering the commands are:

1. Online programming
2. Leadthrough programming
3. Walk-through programming
4. Offline programming
5. Task programming

LEADTHROUGH PROGRAMMING

Leadthrough programming requires the operator to move the robot arm through the desired motion path during a teach procedure, thereby entering the program into the controller memory for subsequent playback. There are two methods of performing the leadthrough teach procedure:

- Powered leadthrough
- Manual leadthrough

The difference between the two is the manner in which the manipulator is moved through the motion cycle.

Powered leadthrough is commonly used as the programming method for playback robots with point-to-point control. It involves the use of a teach pendant (hand-held control box) which has toggle switches or contact buttons for controlling the movement of the manipulator joints. Using the toggle switches or buttons, the programmer drives the robot arm to the desired positions, in sequence, and records the positions into memory. During subsequent playback, the robot moves through the sequence of positions under its own power.

Manual leadthrough is convenient for programming playback robots with continuous path control in which the continuous path is an irregular motion pattern such as in spray painting. This programming method requires the operator to physically grasp the end-of-arm or tools attached to the arm and manually move through the motion sequence, recording the path into memory. Because the robot arm itself may have significant mass and would therefore be difficult to move, a special programming device often replaces the actual robot for the teach procedure. The programming device has a similar joint configuration to the robot, and it is equipped with a trigger handle (or other control switch), which is activated when the operator wishes to record motions into memory. The motions are recorded as a series of closely spaced points. During playback, the path is recreated by controlling the actual robot arm through the same sequence of points.

Powered leadthrough is the most common programming method in industry at this time.

Advantages

- Easy to program: shop personnel can readily learn it and does not require deeper programming experience.

Disadvantages

- Interruption in production.
- Teach pendant have limitations in the amount of decision making logic that can be incorporated in the program.
- No interface to other computer subsystems in the factory.

ROBOT LANGUAGES

A language is a system of communication, which usually is connected to human spoken language and which is based on an arbitrary system of symbols. The most important feature of a language is its ability to produce messages. In a computer, the executable control program is formed of a sequence of machine-language commands. A machine-language command consists of a numerical code, which contains the type of the command and the source and destination addresses of the information. To make programming easier, several high-level programming languages have been developed. Instead of numbers and addresses, the developer can now use words and names. Before the use of such a high-level control program, it must be compiled to machine-language code. This is done by compilers, which have been developed for each language.

Different languages have different aims and are suitable for different purposes. For example, MATLAB is a mathematical language, which has been developed to solve mathematical problems. It has built-in functions for powerful mathematical analysis, but it is not suitable for real-time control of a mobile robot. HTML is a markup language to describe how information appears in web browsers, but it is not suitable to solve mathematical problems. Some of the basic types of commands in programming languages are:

1. Motion and sensing functions (*e.g.*, MOVE, MONITOR)
2. Computation functions (*e.g.*, ADD, SORT)
3. Program flow control functions (*e.g.*, RETURN, BRANCH)

TYPES OF ROBOT LANGUAGES

The earliest methods for training a robot like mechanical setup, point-to-point path recording, and task lead through did not use word-based languages. Some of the high-level computer languages now used to program robots are: Wave, AL, ACL, AML, APT, ARCL, ZDRL, HELP, Karel, CAP 1, MML, RIPL, MCL, RAIL, RPL, ARMBASIC, Androtext, VAL, IBL, and Ladder Logic.

Wave

Wave was the first high-level language created for programming a robot. Standford Artificial Laboratory developed it in 1973.

AL

The AL (Arm Language) high-level programming language was developed at the robotics research center of Stanford University.

ACL

The Advanced Command Language (ACL) is a robot language that employs a user-friendly conversational command environment. Yaskaua robots use it.

AML

AML is the programming language used for the control of robots produced by IBM. AML is intended to provide a complete interpreted computer language along with all of the programming support typically associated with high-level programming languages.

APT

The Automatically Programmed Tools (APT) language is a computer language dealing with motion. Electronic Systems Laboratory of MIT developed it in 1956.

ARCL

ARCL (A Robot Control Language) was based on Pascal-like syntax. It was a compiled language and the developed cross-compiler required three passes before the executable code was ready to be downloaded and executed in a robot. This language has a Pascal-like syntax with sensory-control and motion control commands. An example of an ARCL-language command is MOVA (GRIP, HI, CONT, MED), which opens the gripper on the robot. ARCL emphasized sensory-based programming rather than planned trajectory motion and was designed for educational robot.

HELP

HELP is a high-level programming language developed for use with General Electric's Allegro assembly robot.

Karel

Karel, Karel 2, and Karel 3 are robot control languages used by some FANUC robot controllers.

CAP 1

The Conversational Auto Programming 1 (CAP 1) robot language is used by the FANUC 32-18-T Robot Controller.

MML

MML was a model-based mobile robot language that was developed at the University of California. It is a high-level offline programming language, which contains functions for high-level sensor functions, geometric model description and path planning, and others. This language contains an important concept of slow and fast functions, which architecture is essential for real-time control of robots. A slow function is executed sequentially, while a fast function is executed immediately. The second important concept is the separation of the reference and current posture, which makes precise and smooth motion control and dynamic posture correction possible.

RIPL

RIPL (Robot Independent Programming Language) is based on an object-oriented Robot Independent Programming Environment [RIPE]. The RIPE computing architecture consists of a hierarchical multiprocessor approach, which employs distributed general and special-purpose processors. This architecture enables the control of diverse complex subsystems in real time while co-coordinating reliable communications between them.

MCL

MCL is short for Manufacturing Control Language and was developed by McDonnell Douglas for the U.S. Air Force's ICAM project.

RAIL

RAIL is a high-level programming language developed by Automatix for use with robots and vision systems.

RPL

RPL is a high-level programming language developed by SRI and is used to configure automated manufacturing systems.

ARMBASIC

ARMBASIC is an extension of the hobbyist computer language BASIC. It was used with the Microbot Mini-Mover 5 educational robot.

Androtext

Androtext is a high-level computer language developed by Robotronic Corporation to make commanding a personal robot easier.

VAL

VAL stands for Victor's Assembly Language. VAL is a high-level programming language developed for PUMA lines of robots. The programming language is similar to BASIC. It has a complete set of vocabulary words for writing and editing robot programs.

IBL

IBL (Instruction Based Learning) is a method to train robots using natural language instructions. IBL uses unconstrained language with a learning robot system. A robot is equipped with a set of primitive sensory-motor procedures such as turn left or follow the road that can be regarded as an execution-level command language. The user's verbal instructions are converted into a new procedure and that procedure becomes a part of the knowledge that the robot can use to learn increasingly complex procedures. With this procedure, the robot should be capable of executing increasingly complex tasks. Because errors are always possible in human-machine communication, IBL verifies whether the learned subtask is executable. If it is not, then the user is asked for more information.

EXAMPLE OF A ROBOT PROGRAM USING VAL

The VAL language is the most advanced commercial language designed for use with Unimation, Inc. industrial robots. VAL stands for Victor's Assembly Language. It is basically an offline language in which the program defining the motion sequence can be developed off line, but the various point locations used in the work cycle are most conveniently defined by leadthrough. To demonstrate the VAL language, let us assume that the robot must pick up objects from a chute and place them in successive boxes. One possible sequence of robot activity is as follows:

1. Move to a location above the part in the chute.
2. Move to the part.
3. Close the gripper jaws.
4. Remove the part from the chute.
5. Carry the part to a location above the box.
6. Put the part into the box.
7. Open the gripper jaws.
8. Withdraw from the box.

The corresponding VAL program is as follows:

EDIT DEMO. 1

- PROGRAM DEMO. 1
  1. ? APPRO PART, 50 @
  2. ? MOVES PART @
  3. ? CLOSE I @
  4. ? DEPARTS 150@
  5. ? APPROS BOX, 200@
  6. ? MOVE BOX@
  7. ? OPENI @
  8. ? DEPART@

The exact meaning of each line is:

1. Move to a location 50 mm above the part in the chute.
2. Move along a straight line to the part.
3. Close the gripper jaws.
4. Withdraw the part 150 mm from the chute along a straight-line path.
5. Move along a straight line to a location 200 mm above the box.
6. Put the part into the box.
7. Open the gripper jaws.
8. Withdraw 75 mm from the box.

When the program is executed, it causes the robot to perform the steps which describe the task.

MOTION COMMANDS:

Among the most important functions in a robot language are those which control the movement of the manipulator arm. This section describes how the textual languages accomplish these functions.

Move and Related Statements:

One of the most important functions of the language, and the principal feature that distinguishes robot languages from computer programming languages, is manipulator motion control. we defined the basic motion command, the MOVE statement

MOVE A1

This causes the end of the arm (end effector) to move from its present position to the point (previously defined), named A1. A point is defined in terms of the robot's joint positions, and so A1 defines the position and orientation of the end effector. This MOVE statement generally causes the arm to move with a joint-interpolated motion. There are variations on the MOVE statement. For example, the VAL II language provides for a straight line move with the statement:

MOVES A1

The suffix S stands for straight line interpolation. The controller computes a straight line trajectory from the current position to the point A1 and causes the robot arm to follow that trajectory.

In some cases, the trajectory must be controlled so that the end effector passes through some intermediate point as it moves from the present position to the next point defined in the statement. This intermediate point is referred to as a via point. The need for the via point arises in applications in which there are obstacles and clearances to be considered along the motion path. For example, in removing a part from a production machine, the arm trajectory would have to be planned so that no interference occurs with the machine. The move statement for this situation might read like the following:

MOVE A1 VIA A2

This command tells the robot to move its arm to point A1, but to pass through via

point A2 in making the move.

A related move sequence involves an approach to a point and departure from the point. The situation arises in many material-handling applications, in which it is necessary for the gripper to be moved to some intermediate location above the part before proceeding to it for the pick up. This is what is called an approach, and the robot languages permit this motion sequence to be done in several different ways. We will use VAL II to illustrate. Suppose the robot's task is to pick up a part from a container. We assume that the gripper is initially open. The following sequence might be used:

APPRO A1, 50
DOVES A1
SIGNAL (to close gripper)
DEPART 50

The APPRO command causes the end effector to be moved to the vicinity of point A1, but offset from the point along the tool z axis in the negative direction (above the part) by a distance of 50 mm. From this location the end effector is moved straight to the point A1 and closes its gripper around the part. The DEPART statement causes the robot to move away from the pickup point along the tool z axis to a distance of 50 mm. The provision is available in VAL II for the APPRO and DEPART statements to be performed using straight line interpolation rather than joint interpolation. These commands are APPROS and DEPARTS, respectively.

In addition to absolute moves to a defined point in the workspace, incremental moves are also available to the programmer. In the incremental move, the direction and distance of the move must be defined. This is commonly done by specifying the particular joint(s) to be moved and the distance of the move. Move distances for linear joints are defined in inches or millimeters, while rotational joint moves are specified in degrees of rotation. The following examples from AML illustrate the possibilities:

DMOVE(1,10)
DMOVE(<4,5,6>, <30,-60,90>)

DMOVE is the command for an incremental or 'Delta' move. In parenthesis, the joint and the distance of the incremental move are specified. The first example moves joint 1 (assumed to be a linear joint) by 10 in. The second example commands an incremental move of axes 4, 5, and 6 by 30°, - 60°, and 90°, respectively.

In the AL language, which is designed for multiple arm control, the MOVE statement can be used to identify which arm is to be moved. Robots of the future might possess more than a single arm, and we present the AL statement to illustrate how this might be done.

MOVE ARM2 TO A1

The robot is instructed to move its arm number 2 from the current position to point Al.

Speed control:

The SPEED command is used to define the velocity with which the robot's arm is moved. When the SPEED command is given in the monitor mode (preparatory to executing a program), it indicates some absolute measure of velocity available for the robot. This might be specified as

SPEED 60 IPS

which indicates that the speed of the end effector during program execution shall be 60 in./sec unless it is altered to some other value during the program. If no units are given, the speed command usually indicates some value relative to the robot designer's concept of 'normal' speed. For instance,

SPEED 75

indicates that the robot should operate at 75 per cent of normal speed during program execution (unless altered during the program).
When the speed command is included as a statement in the robot program, it can be used either to specify the actual speed (e.g., 60 IPS), or it can indicate that the robot should operate at a certain per cent of the speed that was specified under monitor mode before program execution. For example, if SPEED 60 IPS was specified under monitor command, and the following statement appeared in the program SPEED 75

it would mean that the subsequent statements should be performed at a speed that is 75 per cent of 60 IPS (45 in./sec).

Definition of Points in the Workspace:

Our motion control programs have made use of points in the workspace. The locations of these points must be defined for the program. As indicated earlier, the definition of point locations is usually done by means of a teach pendant. The pendant is used to drive the robot arm to the desired position and orientation. Then, with a command typed into the keyboard such as

HERE A1

that point location is named A1. (The HERE statement is used in the VAL language.) The position and orientation of each joint are captured in control memory as an aggregate such as

<50.526, 236.003, 14.581, 25.090, 125.750>

As indicated previously, the first three values are the x-y-z coordinates in world space and the remaining values are wrist rotation angles. An alternative way of specifying points in space is to name the point and designate its coordinate values by typing them into control memory directly without using the teach pendant. We have used the following method for specifying these coordinate values

DEFINE A1= POINT <50.526, 236.003, 14.581, 25.090, 125.750> There are, of course, problems in defining points in this way because of the programmer's difficulty in knowing the coordinates in the work cell of the desired position for the robot end effector.

Paths and Frames:

Several points can be connected together to define a path in the workspace. The following statement might be used to specify a path

DEFINE PATH1 PATH(A1, A2, A3, A4)

Accordingly, the path PATH1 consists of the connected series of points A1, A2, A3, and A4, defined relative to the robot's world space. The beginning of the path is A1 and the end of the path is the last point that is specified in the series. A path can consist of two or more points. All points specified in the path statement must have been previously defined. The manner in which the robot moves between the points in the path is determined by the motion statement. For example, the statement

MOVE PATH1

would indicate that the robot arm would move through the sequence of positions defined in PATHI using a joint-interpolated motion between the points. If the statement is used, this indicates that straight line interpolation must be used to move between the points in the path.

MOVES PATH1

A reference frame is a Cartesian coordinate system that may have other points or paths defined relative to it. For example, suppose a part has several identical features replicated in its design, each with a different position and orientation. Furthermore, suppose that the robot must be programmed to process each of the identical features on the part. An example of this kind of situation might be the routing of the same pattern at several locations around the contour of a curved plate. The pattern would consist of a path which contains several moves, and although all paths are identical. their positions and orientations in space vary around the part. In this situation, it would be convenient to program the routing path relative to a reference frame and then redefine the reference frame for each location on the part. The following statement conveys the concept of the frame definition in robot programming

DEFINE FRAME1 = FRAME(A1, A2, A3).

The variable name given to the frame is FRAME1. Its position in space is defined using the three points, A1, A2, and A3. A1 becomes the origin of the frame, A2 is a point along the x axis, and A3 is a point in the xy plane. Accuracy is improved in the internal calculations as the separation between points is increased. The three points uniquely define the Cartesian coordinate system of the new frame. The z axis is perpendicular to the xy plane, with its positive direction pointing to form a right- hand coordinate system.

In our illustration of the series of routing operations around the contoured surface of a curved plate, let us assume that there are nine identical patterns to be made, each one with a different reference frame. We can define the nine frames as FRAME1, FRAME2,..., FRAME9. A routing path, called ROUTE, can now be defined relative to one of these frames (any frame will do) by means of a statement such as the following

```
DEFINE ROUTE:FRAME1 = PATH(P1, P2, P3, P4, P5, P6, P7)
```

where the series of points P1 through P7 defines the routing pattern at the first position on the part identified by FRAME1. Instead of the seven points being defined relative to the world space coordinate system, they are defined relative to the new coordinate system FRAME1. When the robot is commanded to follow the path, the statement must include the definition of the reference frame, as follows

```
MOVES ROUTE:FRAME1
```

To repeat the same path, only relocating and reorienting it relative to successive reference frames, we would use the following commands as required to execute the sequence of routing operations

```
MOVES ROUTE:FRAME2
     .
     .
     .
MOVES ROUTE:FRAME3
     .
     .
     .
MOVES ROUTE:FRAME9
```

Using the computational methods for compound transformations , the path ROUTE is transformed in the robot space into each new frame that is specified in the move command. Each of the points in ROUTE is transformed into the new frame, and the straight line segment path is executed accordingly.

If the programmer were to use the statement MOVES ROUTE, that is, without specifying the particular frame, then the default condition would require the robot to interpret the command so that the world space coordinate system were used as the reference frame. The path would be transformed into the robot's world cartesian coordinate system.

END EFFECTOR AND SENSOR COMMANDS

we made use of the SIGNAL and WAIT statements to initiate output signals or await input signals. The signals were binary (on-off) which imposed limitations on the level of control that could be exercised. The second generation languages have more advanced input-output capabilities.

End Effector Operation:

One of the uses of the SIGNAL commands in the previous chapter was to operate the gripper: SIGNAL 5 to close the gripper and SIGNAL 6 to open the gripper. In most robot languages, there are better ways of exercising control over the end effector operation. The most elementary commands are

OPEN and CLOSE

VAL II distinguishes between differences in the timing of the gripper action. The two commands OPEN and CLOSE cause the action to occur during execution of the next motion, while the statements
OPENI and CLOSEI

cause the action to occur immediately, without waiting for the next motion to begin. This latter case results in a small time delay which can be defined by a parameter setting in VAL II.

The preceding statements accomplish the obvious actions for a non-servoed gripper. Greater control over a servoed gripper operation can be achieved in several ways. For instance, the command

CLOSE 40 MM

or

CLOSE 1.575 IN

when applied to a gripper that has servocontrol over the width of the finger opening would close the gripper to an opening of 40 mm (1.575 in.). Similar commands would control the opening of the gripper.

Some grippers also have tactile and/or force sensors built into the fingers. These permit the robot to sense the presence of the object and to apply a measured force to the object during grasping. For example, a gripper servoed for force measurement can be controlled to apply a certain force against the part being grasped. The command

CLOSE 3.0 LB

indicates the type of command that might be used to apply a 3-lb gripping force

against the part. Force control of the gripper can be substantially more refined than the preceding command. For a properly instrumented hand, the AL language statement

CENTER

provides a fairly sophisticated level of control for tactile sensing. Invoking this command causes the gripper to slowly close until contact is made with the object by one of the fingers. Then, while that finger continues to maintain contact with the object, the robot arm shifts position while the opposite finger is gradually closed until it also makes contact with the object. The CENTER statement allows the robot to center its arm around the object rather than causing the object to be moved by the gripper closure. This could be useful in determining the position of an object whose location is only approximately known by the robot.

For end effectors that are powered tools rather than grippers, the robot must b able to position the tool and operate it. An OPERATE statement (based roughly on a command available in the AL language) might be used to control the powered tool For example, consider the following sequence of commands:

OPERATE TOOL (SPEED=125 RPM)
OPERATE TOOL (TORQUE = 5 IN LB)
OPERATE TOOL (TIME=10 SEC)

We are assuming a powered rotational tool such as a powered screwdriver All three statements apply to the operation. However, the first two statements are mutually exclusive; either the tool can be operated at 125 r/min or it can be operated with a torque of 5 in.-lb. The driver would be operated at 125 r/min until the screw began to tighten, at which point the torque statement would take precedence. The third statement indicates that after 10 sec the operation will terminate.

Sensor Operation:

Let us consider some additional control features of the SIGNAL, WAIT, and similar statements beyond those described in Chap. 8. The SIGNAL command can be used both for turning on or off an output signal. The statements

SIGNAL 3, ON
.
.
SIGNAL 3, OFF

would allow the signal from output port 3 to be turned on at one point in the program and turned off at another point in the program. The signal in this illustration is assumed to be binary. An analog output could also be controlled with the SIGNA command. We will reserve for analog signals the input/output ports numbered greater than 100. The statement could be written as follows:

SIGNAL 105, 4.5

This would provide an output of 4.5 units (probably volts) within the allowabl range of the output signal.

The on-off conditions can also be applied with the WAIT command. In th following sequence, the robot provides power to some external device. The WA is used to verify that the device has been turned on before permitting the program to continue. Later in the program, the robot turns off the device and the device signals back that it has been turned off before the program continues. The relevant commands are as follows:

SIGNAL 5, ON Robot turns on the device
WAIT 15, ON Device signals back that it is on
.
.
.
SIGNAL 5, OFF Robot turns off the device

WAIT 15, OFF Device signals back that it is off

The WAIT statement can be used for analog signals as well as binary digital signals in the same manner as the SIGNAL command.

Instead of identifying input and output signals by their I/O port number, it is often more convenient to define a variable name for the signal. This is usually easier for the programmer to remember. It also permits the variable to be used in the program, so that its value might be changed within the program by means of some logical operators (to be covered in Sec. 9.7). The variables could be defined as follows

DEFINE MOTOR1 = OUTPORT 5
DEFINE SENSR3 = INPORT 15

This would permit the preceding input output statements to be written in the following way:

SIGNAL MOTOR1, ON
WAIT SENSR3, ON

SIGNAL MOTOR1, OFF

WAIT SENSR3, OFF

It is also possible to define an analog signal, either input or output, as a variable that is used during program execution. The statement

DEFINE VOL T1 = OUTPORT 105

specifies that the variable VOLT1 will be used with output port 105. At some point in the program, that variable could be computed to be a particular value (e.g., 4.5 V). and that value could be sent to the designed device in the cell by the statement

SIGNAL VOLT1

The value of VOLTI would be signaled to the external device through output port 105. Similar use can be made of the WAIT command for an analog input signal. Specification of the variable name and associated input port is done by

DEFINE VOLTS3 = INPORT 115

Subsequent use of the variable can be made in a WAIT statement as follows

WAIT VOLT3

which indicates that the program execution should wait for the value of the signal on input port 115 to have a value that is greater than or equal to VOLT3. The programmer must keep in mind what the normal signal level is likely to be (whether it is normally greater than or less than VOLT3) since this may influence the logic of the program