UNIT – VI

ROBOT PROGRAMMING – A robot program may be defined as a path in space to be followed by the manipulator, combined with the peripheral actions that support the work cycle.

Peripheral actions include:
• Opening and closing the gripper
• Performing the logical decision making
• Communicating with other equipment in the robot cell

Current approaches to programming are classified as into two major categories:

   • In this type of programming, an assembly task is explicitly described as sequence of robot motions.
   • The robot is guided and controlled by the program through the entire task with each statement of programme corresponding to one action of the robot.

2. *Object-Oriented or task-level programming*:
   • It describes the assembly task as sequence of positional goals of the objects rather than the motion of the robot needed to achieve the goals.
   • No explicit robot motion is specified.

METHODS TO PROGRAM THE ROBOTS WORK CYCLE: Robots can be programmed by the following methods;

1. Leadthrough methods
2. Textual robot languages
3. Off-line programming
1. Leadthrough programming methods:

There are two ways of accomplishing leadthrough programming:

1. Power leadthrough
2. Manual leadthrough

The powered leadthrough method makes use of a teach pendant to control the various joint motors, and to power drive the robot arm and wrist through a series of points in space. Each point is recorded into memory for subsequent playback during the work cycle. The teach pendant is usually a small handheld control box with combinations of toggle switches, and buttons to regulate the robot’s physical movements and programming capabilities. Among the various robot programming methods, the powered leadthrough method is probably the most common today. It is largely limited to point-to-point motions rather than continuous movement because of the difficulty in using the teach pendant to regulate complex geometric motions in space.

The manual leadthrough method (also sometimes called the “walk-through” method) is more readily used for continuous-path programming where the motion cycle involves smooth complex curvilinear movements of the robot arm. The most common example of this kind of robot application is spray painting, in which the robot wrist, with the spray painting gun attached as the end effector, must execute a smooth, regular motion pattern in order to apply the paint evenly over the entire surface to be coated. Continuous arc welding is another example in which continuous-path programming is required and this is sometimes accomplished with the manual leadthrough method.

In the manual leadthrough method, the programmer physically grasps the robot arm (and end effector) and manually moves it through the desired motion cycle. If the robot is large and difficult to physically move, a special programming apparatus is often substituted for the actual robot. This apparatus has basically the same geometry as the robot, but it is easier to
manipulate during programming. A teach button is often located near the wrist of the robot (or the special programming apparatus) which is depressed during those movements of the manipulator that will become part of the programmed cycle.

2. **Textual Robot Languages:** Non computer controlled robots are programmed by manual method or by the walk through or lead through methods. Only computer controlled robots require a programming language.

Programming language in robotics comprises the generation of all data required to move the robot end effector along a required path in order to perform a specific task.

3. **Off-line Programming:**

Off-line programming method involves the preparation of the robot program off-line, in a manner similar to NC part programming. Off-line robot programming is typically accomplished on a computer terminal. After the program has been prepared, it is entered into the robot memory for use during the work cycle.

The advantage of this programming method is that programming can be done while the robot is still in production on the preceding job, thus production time of the robot is not lost to delays in teaching the robot a new task. This ensures higher utilisation of the robot.

**MOTION INTERPOLATION**

On many robots, the programmer can specify which type of interpolation scheme to use. The possibilities include:

1. Joint interpolation.
2. Straight line interpolation
3. Circular interpolation.
4. Irregular smooth motions (manual leadthrough programming).
For many commercially available robots, joint interpolation is the default procedure that is used by the controller. That is, the controller will follow a joint interpolated path between two points unless the programmer specifies straight line (or some other type of) interpolation.

*Circular interpolation* requires the programmer to define a circle in the robot’s workspace. This is not conveniently done by specifying three points that lie along the circle. The controller then constructs an approximation of defined circle. The movements that are made by the robot actually consist of short-straight-line segments. Circular interpolation therefore produces a linear approximation of the circle. If the grid work of addressable points is dense enough, the linear approximation looks very much like a real circle. Circular interpolation is more readily programmed using a textual programming language than with lead through techniques.

In manual lead through programming, when the programmer moves the manipulator wrist to teach spray painting or arc welding, the movements typically consists of combinations of smooth motion segments. These segments are sometimes approximately straight, sometimes curved (but not necessarily circular), and sometimes back-and-forth motions. We are referring to these movements as *irregular smooth motions*, and an interpolation process is involved in order achieve them. To approximate the irregular smooth pattern being taught by the programmer, the motion path is divided into a sequence of closely spaced points that are recorded into the controller memory. These positions constitute the nearest addressable points to the path followed during programming. The interpolated path may consist of thousands of individual points that the robot must play back during subsequent program execution.
**WAIT, SIGNAL, AND DELAY COMMANDS**

Nearly all industrial robots can be instructed to send signals or wait for signals during execution of the program. These signals are sometimes called interlocks. The most common form of interlock signal is to actuate the robot’s end effector. In case of a gripper, the signal is to open or close the gripper. Signals of this type are usually binary; that is, the signal is on/off or high-level/low-level. A binary valve to actuate the gripper is controlled by means of two interlock signals, one to open the gripper and the other to close it. In some cases, feedback signals can be used to verify that the actuation of the gripper had occurred, and interlocks could be designed to provide this feedback data.

In addition to control of the gripper, robots are typically coordinated with other devices in the cell also. For example, let us consider a robot whose task is to unload a press. It is important to slowdown the robot gripper entering the press before the press is open, and even more obvious, it is important that the robot remove its hand from the press before the press closes.

To accomplish this coordination, we introduce two commands that can be used during the program. The first command is

**SIGNAL M**

which instructs the robot controller to output a signal through line M (where M is one of several output lines available to the controller). The second command is

**WAIT N**

which indicates that the robot should wait at its current location until it receives a signal on line N (where N is one of several input lines available to the robot controller).

Let us suppose that the two-axis robot is to be used to perform the unloading of a press. The layout of the work cell is illustrated in Figure below.
The platten of the press (where the parts are to be picked up) is located at (8,8). The robot must drop the parts in a drop-off bin located at (1,8). One of the columns of the press is in the way of an easy straight line move from (8,8) to (1,8). Therefore, the robot must move its arm around the near side of the column in order to avoid colliding with it. This is accomplished by making use of points (8,1) and (1,1). Point (8,1) will be our position to wait for the press to open before entering the press to remove the part, and the robot will be started from point (1,1), a point in space known to be safe in the application. We will use controller ports 1 to 10 as output (SIGNAL) lines and ports 11 through 20 as input (WAIT) lines. Specifically, output line 4 will be used to actuate (SIGNAL) the press, and output lines 5 and 6 will be used to close and open the gripper, respectively. Input line 11 will be used to receive the signal from the press indicating that is has opened (WAIT). The following is the program to accomplish the press unloading task.
<table>
<thead>
<tr>
<th>S.No.</th>
<th>Move or signal</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,1</td>
<td>Start at home position</td>
</tr>
<tr>
<td>1</td>
<td>8,1</td>
<td>Move to wait position</td>
</tr>
<tr>
<td>2</td>
<td>WAIT 11</td>
<td>Wait for press to open</td>
</tr>
<tr>
<td>3</td>
<td>8,8</td>
<td>Move to pickup point</td>
</tr>
<tr>
<td>4</td>
<td>SIGNAL 5</td>
<td>Signal gripper to close</td>
</tr>
<tr>
<td>5</td>
<td>8,1</td>
<td>Move to safe position</td>
</tr>
<tr>
<td>6</td>
<td>SIGNAL 4</td>
<td>Signal press to actuate</td>
</tr>
<tr>
<td>7</td>
<td>1,1</td>
<td>Move around press column</td>
</tr>
<tr>
<td>8</td>
<td>1,8</td>
<td>Move to drop-off bin</td>
</tr>
<tr>
<td>9</td>
<td>SIGNAL 6</td>
<td>Signal gripper to open</td>
</tr>
<tr>
<td>10</td>
<td>1,1</td>
<td>Move to safe position</td>
</tr>
</tbody>
</table>

Each step in the program is executed in sequence, which means that the SIGNAL and WAIT commands are not executed until the robot has moved to the point indicated in the previous step.

An alternative way to address this problem is to cause the robot to delay before proceeding to the next step. In this case, the robot would be programmed to wait for a specified amount of time to ensure that the operation had taken place. The command

**DELAY X SEC**

indicates that the robot should wait X seconds before proceeding to the next step in the program. Below, we show a modified version of the above example, using time as the means for ensuring that the gripper is either opened or closed.
### GENERATIONS OF ROBOT PROGRAMMING LANGUAGES

Following are the three major classes into which robot languages can be broadly grouped:

1. **First Generation Language:**

   - This type of language provides an off-line programming in combination with the programming through the robot teach pendant.

   - Its capability is limited in handling of sensory data and communication with other components. The programming instructions can be used to define the motion sequence of the manipulator (MOVE), they have input/output capabilities (WAIT, SIGNAL) and they can be used to write subroutines (BRANCH).

Example: VAL (Versatile Algorithmic Language)
2. Second Generation Language:

- These are structured programming languages performing complete tasks.

- They can generate complex motions; can handle both analog and digital signals besides the binary signals.

- These languages have the added advantage of better interfacing facilities with other computers. Data processing, file management and keeping all the records of events happening in the work cell can be done more efficiently.

  Example: AML (A Manufacturing Language), RAIL (Robotic Automatix Incorporated Language), RCL, VALII etc.

3. Word Modeling and task-oriented object level language:

- A more advanced future language is word modeling. Here, a task is defined through a command (Say “TIGHTEN THE NUT”). In such a case intelligence is required and the robot should be capable of making decision.

- Future generation robot languages involve technology of artificial intelligence and hierarchical control system
ROBOTIC PROGRAMMING LANGUAGES

**AL:** The AL (Assembly Language) was developed at the robotic research centre of Stanford University. Its characteristics are:

- High level language with features of ALGOL and PASCAL.
- It is compiled into low-level language and interpreted on a real time control machine.
- It could be used to control multiple arms in tasks requiring arm coordination.
- It supports for word modeling

**AML:** A Manufacturing Language (AML) was developed by IBM. It is the control language for the IBM RS-1 robot. RS-1 robot is a Cartesian manipulator with 6 degrees of freedom. Its first three joints are prismatic and the last three joints are rotary. Its characteristics are:

- Provides an environment where different user interface can be built.
- Supports features of LISP like and APL-like constructs.
- Supports data aggregation
- Supports joint space trajectory planning subject to position and velocity constraints.
- Provides absolute and relative motions
- Provides sensor monitoring
RAIL: Robotic Automatix Incorporated Language (RAIL) was developed by Automatix for the use of robots and vision system.

- It is an interpreter loosely based on PASCAL.
- Several constructs have been incorporated into RAIL to support inspection and arc-welding systems, which are a major product of Automatix.
- The central processor of RAIL is Motorola 68000.
- Peripherals include a terminal and a teach box.
- RAIL is being supplied with three different systems:
  i. Vision only, no arm
  ii. A custom designed Cartesian arm for assembly tasks
  iii. A Hitachi process robot for arc welding

VAL:

- It is a robot programming language and control system originally designed for use with Unimation robots.
- Its stated purpose is to provide the ability to define robot tasks easily.
- The intended user of VAL will typically be the manufacturing engineer responsible for implementing the robot in a desired application.
- It has the structure of BASIC, with many new command words added for robot programming. It also has its own operating system, called VAL monitor, which contains the user interface, editor and file manager.
- It has been released for use with all PUMA robots and with the Unimate 2000 and 4000 series.
BASIC MODES OF OPERATIONS IN A ROBOT

The program and control methods are actuated through software running on an operating system in which manipulation of data takes place. The control functions are activated through monitors.

The three basic modes of operations in a robot are:

1. **Monitor Mode:**

In this mode the programmer can carry out the following functions/activities

- Define locations
- Load a particular piece of information in a particular register.
- Store information in the memory
- Save, transfer programs from storage into computer control memory

Monitor mode uses the following commands:

SPEED, STORE, LIST, PRINT, READ, DIRECTORY, ABORT or STOP etc.

2. **Edit Mode:**

In this mode, the following operations can be carried out

- The programmer can edit or change a set of instructions of existing programs or introduce a new set of information
- The user can erase some instructions and replace them by new lines.
- Any error, is if shown on the monitor, can be corrected.

In order to come out of the mode, and *end command* should be given.

Edit mode uses the following commands:

END, DELETE, or ERASE etc.
3. Run or Executive Mode:

a) During this mode, the sequential steps written by programmer are:

- Sometimes “dry run” can be tested by making the switch disable. After dry run, the switch may be made operational by the instruction “enable”.

- A program can be tested in “run mode” and by debugging, the errors in the program can be rectified.

b) For implementing robot language program, the operating system uses either an ‘interpreter’ or ‘compiler’.

- An ‘interpreter’ takes the source program one line at a time and generates equivalent code that is understood by the ‘compiler’.

- Compiler is a software in the operating system that converts source code into the object code (machine code) after compilation of the program. The robot controller can then read and process the machine codes.

MOTION COMMANDS

Move and Related Statements:

The principal feature that distinguishes robot languages from computer programming languages is manipulator motion control. The basic motion command, the MOVE statement is used as:

```
MOVE A1
```

This causes the end of the arm to move from the present position to the point A1. There are variations in the MOVE statements. For example, in the command 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.
In some cases, the trajectory must be controlled in such a way that, the end effector passes through some intermediate point as it moves from the present position to the next point defines in the statement. The intermediate point is referred as via point. The need of defining the via points is to overcome the obstacles present in the path. The move statement for this situation is

\[
\text{MOVE A1 VIA A2}
\]

This command tells the robot to move its arm to point A1, but it passes through via point A2 in making this move.

In addition to the absolute moves discussed above, sometimes incremental moves may also be used by the programmer. In the incremental move, the direction and distance of the move must be defined.

\[
\text{DMOVE (1, 10)}
\]

\[
\text{DMOVE (4,5,6), (30, -60,90)}
\]

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 by 10 inches. The second example commands an incremental move of the axes 4, 5, and 6 by 30°, -60° and 90° respectively.

**REACT statement**

This statement is used in continuous monitoring of an incoming signal and to respond to a change in the signal in an appropriate manner. It illustrates various types of commands that are used in order to interrupt regular execution of the robot program in case of high priority event i.e., when some form of error or safety hazard occurs in the work cell that is detected by the sensor. A typical form of the statement would be as follows.

\[
\text{REACT 17, SAFETY}
\]

The statement is interpreted as follows. Input line 17 is to be continuously monitored, and when a change in its signal value occurs, branch to a sub
routine called SAFETY. The input signal on line 17 is a binary signal which is at either of two levels (on/off). The change in the signal value which invokes the REACT statement is a change from one level to the other. If the signal is normally off, and suddenly comes on, then REACT transfers program control to the subroutine identified in the statement.