Chapter 3 - Introduction to Robot languages and Programming

Robot Programming : The primary objective of robot programming is to make the robot understand its work cycle.

Two basic types :

1. Lead through methods.
2. Textual robot languages.

Lead through programming methods (Teach-by-showing) : These methods require the programmer to move the manipulator through the desired motion path and that the path to be committed to memory by the robot controller.

Teach-by-showing, which can be divided into:
1. Powered leadthrough or discrete point programming
2. Manual leadthrough or walk-through or continuous path programming

Powered leadthrough method :
- It 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.
- It is limited to point to point movements rather than continuous moments.
- Eg. Part transfer tasks, machine loading and unloading, spot welding.

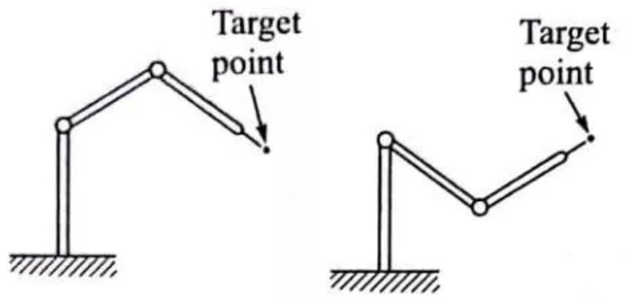
Manual leadthrough method :
- In this the programmer physically grasps the robot arm and manually moves it through the desired motion cycle.
- If the robot is large then a special programming apparatus is often substituted for the actual robot.
- It is used for continuous path programming.
- Eg. Spray painting, continuous arc welding.

In both lead through procedures operate in either

- Teach mode: is used to program the robot
- Run mode: is used to run or execute the program

A Robot program as a path in space

- Since the robot consists of several joints (axes) linked together, the definition of the path in space requires that the robot move its axes through various positions in order to follow that path.
- A point in space in the robot program as a position and orientation of the end effector, there is usually more than one possible set of joint coordinate values.
- We see in the figure that although the target point has been reached by both of the alternative axis configurations, there is a difference in the orientation of the wrist with respect to the point.

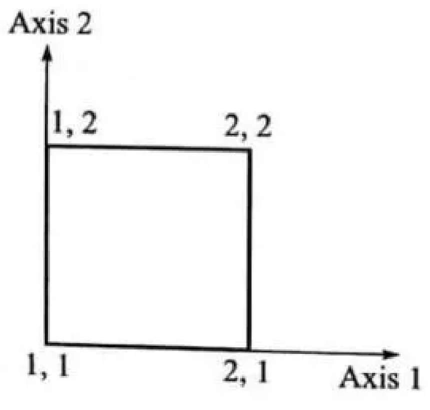


Two alternative axis configurations with end effrctor located at desired target point.

- We must conclude from this that the specification of a point in space does not uniquely define the joint coordinates of the robot.

Example on programming a point-to-point Cartesian robot with only two axes

- Figure shows the four possible points in the robot's rectangular workspace. A program for this robot to start in the lower left-hand corner and traverse the perimeter of the rectangle could be written as follows:

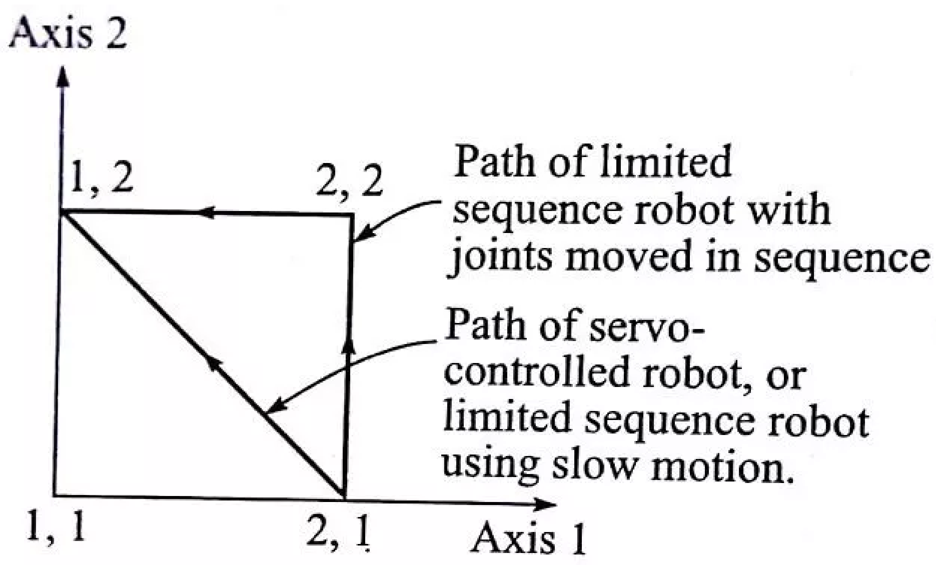


Step	Move	Comment
1	1,1	Move to lower left corner
2	2,1	Move to lower right corner
3	2,2	Move to upper right corner
4	1,2	Move to upper left corner
5	1,1	Move back to start position

- The point designation correspond to the x,y coordinate positions in the cartesian axis system, as illustrated in the figure. In our example, using a robot with two orthogonal slides and only two addressable points per axis, the definition of points in space corresponds exactly with joint coordinate values.

- Using the same robot, let us consider its behavior when performing the following program:

Step	Move	Comments
1	1,1	Move to lower left corner
2	2,1	Move to lower right corner
3	1,2	Move to upper left corner
4	1,1	Move back to start position


Axis 2
1, 2
2, 2
Path of limited sequence robot with joints moved in sequence
Path of servo-controlled robot, or limited sequence robot using slow motion.
1, 1
2, 1
Axis 1

Methods of Defining Positions in Space

There are three methods:

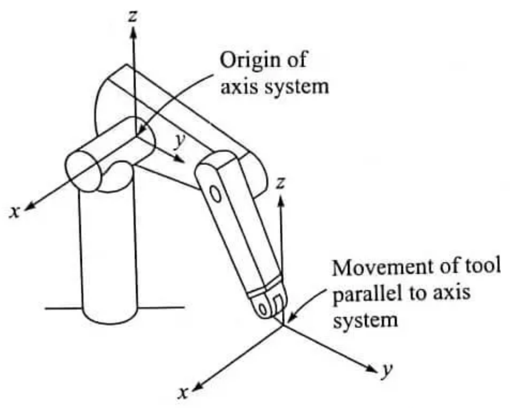
1. Joint movements
2. x-y-z coordinate motions (also called world coordinates)
3. Tool coordinate motions

Joint movements :

- This is the most basic and involves the movement of each joint, usually by means of a teach pendant.
- The teach pendant has a set of toggle switches (or similar control devices) to operate each joint in either of its two directions until the end effector has been positioned to the desired point.
- This method of teaching points is often referred to as the joint mode. Successive positioning of the robot arm in this way to define a sequence of points can be a very tedious and time-consuming way of programming the robot.
- To overcome this disadvantage, many robots can be controlled during the teach mode to move in x-y-z coordinate motions.

x-y-z coordinate motions

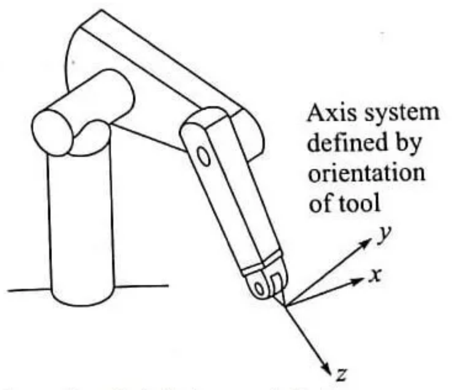
- This method, called the world coordinate system, allows the wrist location to be defined using the conventional Cartesian coordinate system with origin at some location in the body of the robot.
- This method, called the world coordinate system, allows the wrist location to be defined using the conventional Cartesian coordinate system with origin at some location in the body of the robot.
- To the programmer, the wrist (or end effector) is being moved in motions that are parallel to the x,y and z axes.
- The two or three additional joints which constitute the wrist assembly are almost always rotational, and while programming is being done in the x-y system to move the arm and body joints, the wrist is usually being maintained by the controller in a constant orientation.



Work mode or x-y-z method of defining points in space.

Tool coordinate motions

- Some robots have the capability for tool coordinate motions to be defined for the robot. This is a Cartesian coordinate system in which the origin is located at some point on the wrist and the xy plane is oriented parallel to the faceplate of the wrist..
- Accordingly, the z axis is perpendicular to the faceplate and pointing in the same direction as a tool or other end effector attached to the faceplate.
- Hence, this method of moving the robot could be used to provide a driving motion of the tool. Again, a significant amount of computational effort must be accomplished by the controller in order to permit the programmer to use tool motions for defining motions and points.



Tool mode of defining points in space

Motion Interpolation:
Interpolated motion is called for when the path that an object takes through space is important

The following interpolation schemes are available in most of the robot controllers.
1. Joint interpolation
2. Straight line interpolation
3. Circular interpolation
4. Irregular smooth motions (manual lead through programming).

Joint interpolation: The controller determines how far each joint must move to get from the first point defined in the programme to the next. It then selects the joint that requires the longest time. This determines the amount of movement for other axes such that all the axes start and stop at the same time. Joint interpolation is the default procedure for many commercial robots.

Straight-line interpolation: In this interpolation, the robot controller computes the straight-line path between two points and develops the sequence of addressable point along the path for the robot to pass through.

Circular interpolation: This requires the programmer to define a circle in the robot's workspace. This is done by specifying three points that lie along the circle. The controller constructs the circle by selecting a series of points that lie closer to the circle. These movements are actually small straight lines. If the addressable points are dense then the linear approximation becomes very much like circle.

Manual lead through Programming: When the manipulator wrist is moved by the programmer to teach, the movements consist of combination of smooth motion segments. These segments are sometimes approximately straight lines or curves or back and forth motions. These movements are referred as irregular smooth motions and an interpolation is involved to achieve them.

WAIT, SIGNAL and DELAY commands
All industrial robots are instructed to send signals or wait for signals

- These signals are called interlocks
- Common form is to actuate end effectors
- In grippers, its on or off or Binary
- Grippers involve 2 interlocks – open & close
- Feedback might be added to verify actuation
- Communication and coordination is important, so to accomplish this there are commands

1. 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). T

2. 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).

3. 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 assuring that the gripper is either opened or closed.

BRANCHING (Use of Subroutine programs)

- Most controllers for industrial robots provide a method of dividing a program into one or more branches.
- Branching allows the robot program to be subdivided into convenient segments that can be executed during the program.
- A branch can be thought of as a subroutine that is called one or more times during the program.
- The subroutine can be executed either by branching to it at a particular place in the program or by testing an input signal line to branch to it.
- A frequent use of the branch capability is when the robot has been programmed to perform more than one task. In this case separate branches are used for each individual task.

CAPABILITIES AND LIMITATIONS OF LEADTHROUGH PROGRAMMING

- Robot cannot be used in production while programming is carried out
- Slow movement of the robot while programming
- Jogging a robot using teach pendant is not intuitive as many coordinate systems
- are usually defined in a robotic system
- The operator must always track which coordinate frame the robot is set while jogging
- Program logic and calculations are hard to program
- Difficult to incorporate sensor data
- Cost equivalent to production value
- Online programming is not yet readily compatible with CAD/CAM technologies,
- Data communications, networking, and other computer-based technologies.

THE TEXTUAL ROBOT LANGUAGES

- The first textual robot language was WAVE, developed in 1973 as an experimental language for research at the Stanford Artificial Intelligence Laboratory.
- Research involving a robot interfaced to a machine vision system was accomplished using the WAVE language. The research demonstrated the feasibility of robot hand-eye coordination.
- Development of a subsequent language began in 1974 at Stanford. The language was called AL and it could be used to control multiple arms in tasks requiring arm coordination.
- Many of the concepts of WAVE and AL went into the development of the first commercially available robot textual language, VAL.
- VAL was introduced in 1979 by Unimation, Inc., for its PUMA robot series. This language was upgraded to VAL II and released in 1984.
- Two of the IBM languages were AUTOPASS and AML (A Manufacturing Language), the second of which has been commercially available since 1982 with IBM’s robotic products. Both of these languages are directed at assembly and related tasks.

- Some of the other textual languages for robots that should be mentioned include RAIL, MCL (Manufacturing Control Language), APT (Automatically Programmed Tooling), HELP.

GENERATION OF ROBOT PROGRAMMING LANGUAGES

The textual robot languages have a variety of structures and capabilities.

First Generation Languages:

- These languages use a combination of command statements and teach pendant procedures for developing robot programs.
- They were developed largely to implement motion control.
- Typical features include ability to define manipulator motions, straight line interpolation, branching and elementary sensor commands.
- Limitations include inability to specify complex arithmetic computations, and also cannot readily be extended for future enhancements.
- Eg. VAL language.

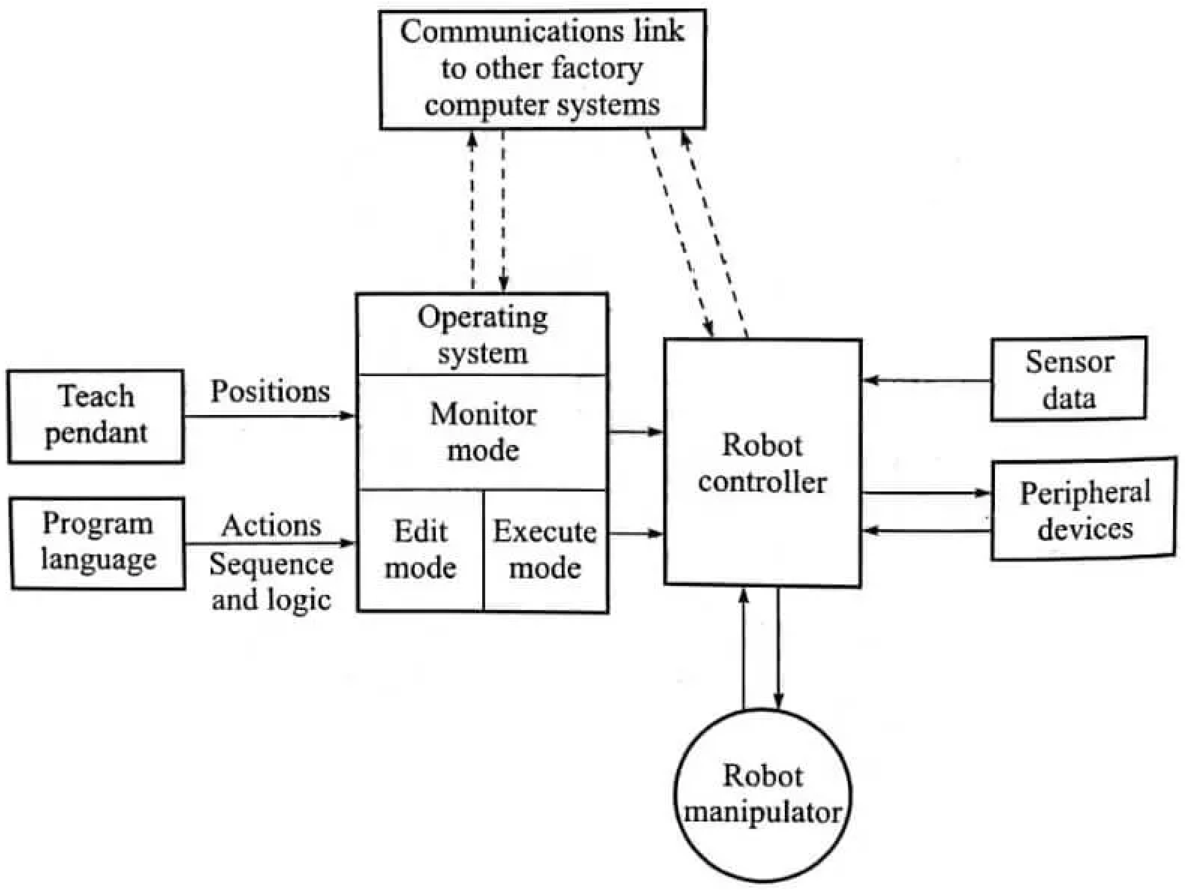
Second Generation Language:

- This language makes robot more intelligent.
- These languages are also known as structured programming languages.
- Features include Motion control, advance sensor capabilities, Limited intelligence, communication and data processing.
- Eg. AML, RAIL, MCL and VAL II.

Future Generation Language:

- Future generation robot language will involve a concept called “world modeling”
- In a programming scheme based on world modeling, the robot possesses knowledge of the three-dimensional world and is capable of developing its own step-by-step procedure to perform a task.
- This robot develops its own three-dimensional model in the workspace.
- In this the system develops its own program of actions required to accomplish the objective.

ROBOT LANGUAGE STRUCTURE



Communication link:

- The language must be designed to operate with a robot system as illustrated in Fig. It must be able to support the programming of the robot, control of the robot manipulator, and interfacing with peripherals in the work cell (e.g., sensors, and equipment).
- It should also support data communications with other computer systems in the factory.

Operating Systems:

- The purpose of the operating system is to facilitate the operation of the computer by the user and to maximize the performance and efficiency of the system and associated peripheral devices.

A robot language operating system contains the following three basic modes of operation:

1. Monitor mode
2. Run mode
3. Edit mode

- **The monitor mode** is used to accomplish overall supervisory control of the system.
- **The run mode** is used for executing a robot program. In other words, the robot is performing the sequence of instructions in the program during the run mode.
- **The edit mode** provides an instruction set that allows the user to write new programs or to edit existing program.

CONSTANTS, VARIABLES AND OTHER DATA OBJECTS

Constants: A constant is a value used in the program that does not change during the execution of the program.

Variable: A variable in computer programming is a symbol or symbolic name that can change in value during execution of the program.

MOTION COMMANDS

1. **MOVE :** This causes the end of the arm (end effector) to move from its present position to the point (previously defined)

   Eg.

   a) **MOVE A1** : This causes the end of the arm (end effector) to move from its present position to the point (previously defined), named A1.

   b) **MOVE A1 VIA A2 :** This command tells the robot to move its arm to point A1 via point A2.

c) **APPRO A1, 50 :** This 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 by a distance of 50mm.

d) **DEPART 50 :** This command causes the robot to move away from the pickup point along the tool Z-axis to a distance 50mm.

e) **DMOVE** : This command is used for an incremental move.
Eg.
i) **DMOVE (1,10) :** In this command the Joint 1 moves linearly by 10 inches.
ii) **DMOVE (<4,5,6> , <30, -60, 90>) :** In this command an incremental move of axes 4, 5, and 6 by 30°, -60°, and 90° respectively.

f) **MOVE ARM2 TO A1 :** The robot is instructed to move its arm number 2 from the current position to point A1.

2. SPEED : This command is used to define the velocity of the arm, which is moved.

Eg.

a) **SPEED 60 IPS :** This commands indicates that the speed to the end-effector during the program execution shall be 60 in./sec.
b) **SPEED 75 :** This command indicates that the robot should operate at 75 percent of normal speed during the program execution.

3. PATH : Several points can be connected together to define a path in the workspace.

Eg.

a) **DEFINE PATH1 = PATH (A1, A2, A3, A4)** : Accordingly, the path PATH consists of the connected series of point A1, A2, A3, A4.

4. **END EFFECTOR:** OPEN and CLOSE commands are used to operate the gripper.

Eg.

a) **OPENI and CLOSEI** : This command cause the action to occur immediately, without waiting for the next motion to begin.
b) **CLOSE 40 or CLOSE 1.575 IN** : When this command is applied to the gripper that has servocontrol over the width of the finger opening would close the gripper to an opening of 40mm (1.575 in)
c) **CLOSE 3.0 LB** : This type of command might be used to apply a 3-lb gripping force against the part.
d) **CENTER** : 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 closing making contact with the object.
e) **OPERATE** : This statement might be used to control the powered tool.
Eg.
i) OPERATE TOOL (SPEED = 125RPM)
ii) OPERATE TOOL (TORQUE = 5 IN LB)
iii) OPERATE TOOL (TIME = 10 SEC)
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 third statement indicates that after 10 seconds the operation will terminate.

5. **SENSOR** : These commands are used to turn on an off an output signal.
Eg.
a) **SIGNAL** : This command can be used for turning on or off an output signal.
b) **WAIT** : This command is used to verify that the device has been turned on before permitting the program to continue.

COMPUTATIONS AND OPERATIONS

- The need arises in many robot programs to perform arithmetic computations and other types of operations on constants, variables and other data objects.
- The standard set of mathematical operators in second generation languages are.

+	Addition

-	Subtraction
*	multiplication
/	Division
**	Exponentiation
=	Equal to

- Some of the languages also have the capability to calculate the common trigonometry, logarithmic, and similar functions.

SIN(A)	Sine of an angle A
COS(A)	Cosine of an angle A
TAN(A)	Tangent of an angle A
COT(A)	Cotangent of an angle A
ASIN(A)	Arc sine of an angle A
ACOS(A)	Arc cosine of an angle A
ATAN(A)	Arc tangent of an angle A
ACOT(A)	Arc cotangent of an angle A
LOG(X)	Natural logarithm of an X
EXP(X)	Exponential function e**X
ABS(X)	Absolute value of X
INT(X)	Largest integer less than or = X
SQRT(X)	Square root of X

- In addition to the arithmetic and trigonometric operators, relational operators are also used.

EQ	Equal to
NE	Not equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to

- Logical operators are also sometimes useful in robot programming

AND	Logical AND operator
OR	Logical OR

NOT Logical complement

PROGRAM CONTROL AND SUBROUTINES

- There are a variety of instructions available in the textual robot languages to control the logic flow of the program and to name and use subroutines in the program.

Program Sequence control :

- Following are the instructions that can be used to control logic flow in the program.
  GOTO 10 : This indicates an unconditional branch to statement 10.

Compare Online and Offline Programming

Sr. No.	Online Programming	Offline Programming
1	No extra hardware or software is required	Requires additional software and a computer
2	Most technicians are familiar with teach pendants	May require additional training for programmers
3	Best for simpler applications	Best for complex programs as saves time from manually entering steps
4	Requires robot to be taken out of production, increasing downtime	Reduces robot downtime since programming is completed offline without the robot present
5	Lead through programming lacks precision	Simulation ensures accurate programming

VAL COMMANDS WITH DESCRIPTION

Definition	*Command Statement*	*Explanation*
1. Motion control	APPRO P1, Z1	Command to approach the point P1 in the z-direction by Z1 distance above the object.
	MOVE P1	Command to move the arm from the present position to point P1.
	MOVE P1 VIA P2	Asks the robot to move to point P1 through point P2.
	DMOVE (J1, ΔX)	Moves the joint J1 by an increment of ΔX (linear)
	DMOVE (J1, J2, J3) ($d\alpha$, $d\beta$, $d\theta$)	Command to move joints J1, J2 and J3 by incremental angles of $d\alpha$, $d\beta$, $d\theta$ respectively.
2. Speed control	SPEED V IPS	The speed of the end effector is to be V inch per second at the time of program execution.
	SPEED R	Command to operate the arm end effector at R percent of the normal speed at the time of program execution.
3. Position control	HERE P1	Defining the name of a point as P1.
	DEFINE P1 = POINT ($x, y, z, w_\alpha, w_\beta, w_\theta$)	The command defines the point P1 with x, y, z co-ordinates and w_α, w_β, w_θ the wrist rotation angles.
	Path control : DEFINE PATH 1 = PATH (P1, P2, P3)	The path of the end effector is defined by the connection between points P1, P2 and P3 in series.
	MOVE PATH 1	Movement of the end effector along path 1.
	Frame definition: DEFINE FRAME 1 = FRAME (P1, P2, P3)	— Assings variable name to FRAME 1 defined by points P1, P2 and P3. P1-origin, P2-point along x-axis and P3-point along xy plane.
	MOVE ROUTE: FRAME 1.	— Defines the movement in the path for frame 1.
4. End effector operation	OPEN	— Opens the gripper fingers.
	CLOSE 50 MM	— In forms gripper to close keeping 50 mm width between the fingers.

	CLOSE 5 LB.	— Applies 5 Lb gripper force.
	CENTER	— Closes the gripper slowly till the establishment of contact with the object to be gripped.
	OPERATE TOOL (SPEED N RPM)	— Positioning and operating the powered tool. Here the EE is replaced by servo powered tool.
5. Operation of the sensors	SIGNAL 4, ON	The command actuates the output port 4 and turns on at certain stage of the program.
	SIGNAL 5, OFF. WAIT 13, ON	The output port 5 is turned off. The device gives a feed back signal indicating that it is on.
	REACT 16, SAFETY.	The change in signal (if any), in the input line 16, should be deviated to the sub-routine SAFETY.

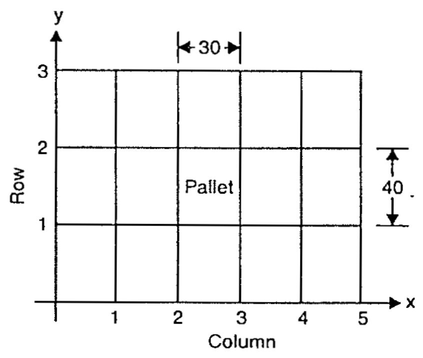
PROGRAM FOR PNP (PICK AND PLACE) ACTIVITY

VAL STATEMENT	*Statement Description*
BRANCH PICK	— The branch of program indicating part pick.
MOVE INTER	— Move to a intermediate position above chute.
WAIT 12	— Wait for a incoming part to the chute.
SIGNAL 5	— Open gripper fingers (sensor control).
MOVE PICK-UP	— Move gripper and pick-up the object.
SIGNAL 6	— Close the gripper to grab the object.
MOVE INTER	— Depart to intermediate position above chute.
END BRANCH	— End of pick-up activity.
BRANCH PLACE	— Start of placing activity.
MOVE Z (–50)	— Position part and gripper above the pallet.
SIGNAL 5	— Open gripper to release the part.
MOVE Z (50)	— Depart from the place (pallet) point.
END BRANCH	— End of place activity.

PROGRAM TO PALLETISE THE OBJECT

Given Data

Pallet variables	
ROW	integer ROW count.
COLUMN	integer COLUMN count.
X	co-ordinate value along x-axis.
Y	co-ordinate value along y-axis.
Locational constants	
PICK-UP	point of pick-up of objects.
CORNER	start of point in the pallet.
Locational variable	
DROP	point of dropping.



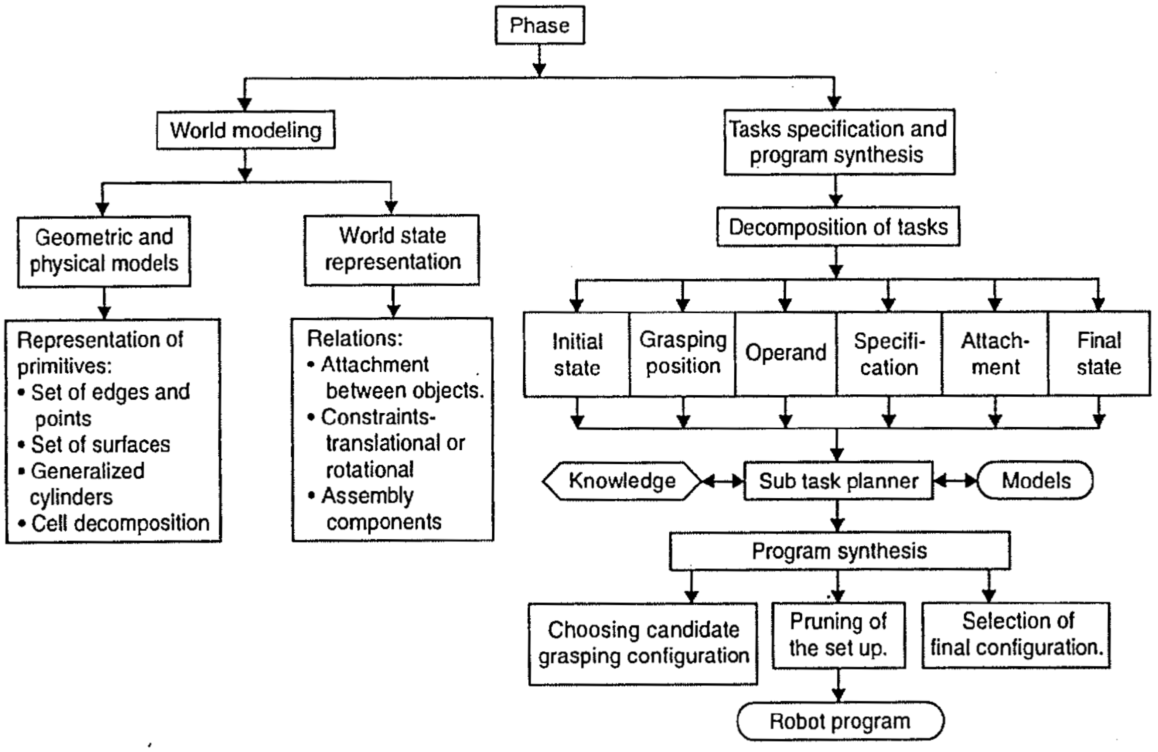
Palletising program.

VAL STATEMENT	*Description of Statement*
PROGRAM PALLETISE	— Start of program.
DEFINE PICK-UP = JOINTS (1, 2, 3, 4, 5)	
DEFINE CORNER = JOINTS (1, 2, 3, 4, 5)	— Definition of constants of location.
DEFINE DROP = CO-ORDINATES (X, Y)	— Definition of variable of pallets.
OPEN	— Gripper action control to open fingers.
ROW = 0.0	— Initialise the row count.
10 Y= ROW * 40.0	— Computation of dropping point (Y-co-ordinate)
COLUMN = 0.0	— Initialization of column count.
20 X = COLUMN * 30.0	— Computation of drop off point (X-co-ordinate)
DROP = CORNER + (X, Y)	— Defining drop for every iteration.
APPRO PICK-UP, 50	— Approach above pallet for pick-up.
MOVE PICK-UP	— Positioning for pick-up action.
CLOSE	— Gripper action control to close the fingers.
DEPART 50	— Action sequence for arm to move away from the pick-up point.
APPRO DROP, 50	— Approach command to position above drop point.
MOVES DROP	— Command to drop object.
OPEN	— Open the gripper fingers.
DEPART 50	— Move away from drop point.
COLUMN = COLUMN + 1.	— Column increment.
IF COLUMN LT 5 GO TO 20	— Check for the column limit (LT ⇒ less than)
ROW = ROW + 1	— Row increment.
IF ROW LT 3 GO TO 10	— Check for ROW limit.
END PROGRAM	— End of program.

AL PROGRAM FOR BOLT INSERTION TASK

```
BEGIN insertion
bolt_dia ← 12 * mm ;
bolt_height ← 25 * mm ;                                   } Set Variables.
tries ← 0 ;
grasped ← false
beam ← FRAME (ROT(Z, 90 * deg), VECTOR (500, 375,0) * mm) ;  } Def. Base frame.
feeder ← FRAME (nilrot, VECTOR(625, 500,0) * mm) ;
bolt_grasp ← feeder * TRANS (nilrot, nilvect) ;
bolt_tip ← bolt_grasp * TRANS (nilrot, VECTOR(0, 0, 12) * mm) ;   }
beam_bore ← beam * TRANS (nilrot, VECTOR(0, 0, 25) * mm) ;
                                                          Def. feature frame.
A ← feeder * TRANS (nilrot, VECTOR(0, 0, 125) * mm) ;
B ← feeder * TRANS (nilrot, VECTOR(0, 0, 200) * mm) ;      } Def. of via point frame.
C ← beam_bore * TRANS (nilrot, VECTOR(0, 0, 125) * mm) ;
D ← beam_bore * TRANS (nilrot, bolt_height * Z) ;
OPEN bhand To bolt_dia + 25 * mm ; }  Open hand.
MOVE barm To bolt_grasp VIA A.
WITH APPROACH = –Z WRT feeder ;       } Position hand above bolt.
CLOSE bhand To 0.9 * bolt_dia ;
IF bhand < bolt_dia THEN BEGIN ;
OPEN bhand To bolt_dia + 25 * mm ;
MOVE barm To 0 – 1 * Z * mm ;        } Bolt grasp activities statements.
END ELSE grasped ← true ;
tries ← tries + 1 ;
UNTIL grasped OR (tries > 4) ;
IF NOT grasped THEN ABORT ("bolt grasp failed") ;
MOVE barm To B
VIA A
WITH DEPARTURE = Z WRT feeder ;           } Movement of the arm to B and D.
MOVE barm To D VIA C
WITH APPROACH = –Z WRT beam_bore ;
MOVE barm to 0 – 0.1 * Z * mm ON FORCE (Z) > 200 * gms   } Check for Hole.
DO ABORT ("Hole absent") ;
MOVE barm To beam-bore DIRECTLY
WITH FORCE (Z) = –200 * gms
WITH FORCE (X) = 0 gms                  } Execution of insertion with compliance.
WITH FORCE (Y) = 0 gms
WITH DURATION = 5 * seconds.
END insertion.
```

ROBOT TASK PLANNING


Phase
World modeling
Tasks specification and program synthesis
Geometric and physical models
World state representation
Decomposition of tasks
Representation of primitives:
• Set of edges and points
• Set of surfaces
• Generalized cylinders
• Cell decomposition
Relations:
• Attachment between objects.
• Constraints- translational or rotational
• Assembly components
Initial state
Grasping position
Operand
Specifi- cation
Attach- ment
Final state
Knowledge
Sub task planner
Models
Program synthesis
Choosing candidate grasping configuration
Pruning of the set up.
Selection of final configuration.
Robot program

Compare Robot and Object Oriented Programming

Robot Oriented Programming	*Object Oriented Programming*
• The sequence of robot motion is programmed and robot is made to follow. • The robot motions are explicitly defined. • The commands and actions are correspondingly related.	• The program developed on task basis as a sequence of goal positions. • The explicit definition of robot motion is not required. • No relation between command and action but the task is executed.

EXPECTED QUESTIONS:

- List different robot languages.
- Compare robot oriented programming with object oriented programming.
- State any FOUR motion commands
- State the limitations of lead through programming.
- Explain lead through programming methods
- Write a VAL program to palletize an object. (Assume all necessary dimensions)
- State the use of subroutine function in robot programming
- State any four sensor commands.
- State the use of Teach pendant in robots
- Write a simple VAL program for Pick and Place operation.
- Write a program for Bolt inspection task.
- State any FOUR End effector commands.
- Compare online and offline programming
- Explain any 6 VAL commands.