



# ROBOT LANGUAGES & PROGRAMMING

## **WAVE and AL**


WAVE, developed at Stanford, demonstrated a robot hand–eye coordination while it was implemented in a machine vision system. Later a powerful language AL was developed to control robot arms. WAVE incorporated many important features. Trajectory calculations through coordination of joint movements, end-effector positions and touch sensing were some of the new features of WAVE. But the algorithm was too complex and not user-friendly. They could not be run in real-time and on-line. On the other hand, trajectory calculations are possible at compile time and they can be modified during run time. AL has a source language, a translator to generate runnable code and a run time system for effecting various motions of the robot manipulator. The syntax of the language can implement various subroutines, involving activities between the robot and its surroundings, various statements concerning SIGNALS and WAFT to carry on tasks in sequence. Different sensors can be incorporated and programming can take care of some condition monitoring statements. The robot manipulator movement-commands include various motions, velocities, forces, torques, etc. POINTY is another interactive system and a part of AL system.

## **VAL**

VAL is a popular textual robot language developed by Unimation Inc. for the PUMA series of robots. VAL has been upgraded to VAL II system with more interlocking facilities. Victor Sheinman developed VAL language. VAL is very user-friendly. It provides arm movement in joint, world and tool coordinates, gripping and speed control. WAIT and SIGNAL commands can be given to implement a specific task. The commands are subroutines written in BASIC and translated with the aid of an interpreter. Compiled BASIC has more flexibility.

## **AML**

A manufacturing language, AML was developed by IBM. AML is very useful for assembly operations as different user-robot programming interfaces are possible. The programming language AML is also used in other automated manufacturing systems. The advantage of using AML is that integers, real numbers and strings can be specified in the same aggregate which is said to be an ordered set of constants or variables. An aggregate can be used to specify coordinate values of the robot's joints or wrist position and orientation.



## **MCL**

US Air force ICAM project led to the development of another manufacturing control language known as MCL by McDonnell-Douglas. This is a modification of the popular APT (Automatically Programmed Tooling) language used in CNC machine tools as many similar commands are used to control machine tools in CAM applications. Lines, circles, planes, cylinders and many other complex geometrical features can be defined in MCL.

## **RAIL**

RAIL was developed by Automatix for robotic assembly, inspection, arc welding and machine vision. A variety of data types as used in PASCAL can be used. An interpreter is used to convert the language into machine language commands. It uses Motorola 68000 type microcomputer system. It supports many commands and control of the vision system.





## **HELP**

HELP was developed by General Electric Company. It acts more or less like RAIL. It has the capability to control two robot arms at the same time. The structure of the language is like PASCAL.

## **JARS**

JARS was developed by NASA's JPL. The base of the language is PASCAL. JARS can be interfaced with PUMA 6000 robot for running robotic programs.

## **RPL**

RPL was developed at SRI International. A compiler is used to convert a program into the codes that can be interpreted by an interpreter. Unimation PUMA 500 can be controlled with the help of RPL. The basic ideas of LISP (an AI language) have been organized into a FORTRAN-like syntax in RPL. It is modular and flexible.



Different textual robot languages have different attributes. For example, VAL, HELP and MCL though powerful for many simple tasks, do not have the same structured modular programming capability like AL, AML, JARS and ADA or VAL II. In a manufacturing cell, multiple robots or robotic equipment work in unison. Control of two or more operations done by the robots in a coordinated manner is complex. Synchronizing the motions of the robots requires necessary software commands. AL, ADA, AML, MCL have the capability of controlling multiple arms. The programming language must be capable of expressing various geometric features like joint angles, coordinate transformations such as rotation, translation, and vector quantities. Homogeneous matrices are used to specify the rotation. Rotation can also be specified by Euler angles. AML, RAIL and VAL use Euler angles while AL manipulates homogeneous matrix for control. AL is very suitable for assembly tasks wherein many sensors are employed, though other languages like AML and HELP are flexible enough to run various subroutines. Slewing and straight line motions control are available with most of the languages.



## AUTOPASS

AUTOPASS is a language proposed by IBM to facilitate a human assembler to perform the activities. This is a sophisticated world modeling system helpful to keep track of the objects. Visual location and identification of parts by using tactile sensors are the facilities taken by the system. Program debugging proceeds interactively with the user to avoid ambiguities in robot's interpretation of the statements given by the user.

However, in all the robot languages, features like editor, interpreter, compiler, data management, and debugging are common.





# CLASSIFICATION OF ROBOT LANGUAGES

---

Robot languages can be grouped broadly into three major classes:

1. First generation language
2. Second generation language
3. World modelling and task-oriented object level language

The first generation language provides an off-line programming in combination with the programming through robot pendant teaching. VAL is an example of a first generation robot programming language. The capability of a first generation language is limited to the handling of sensory data (except ON/OFF binary signals) and communication with other computers. However, branching, input/output interfacing and commands leading to a sequence of movements of arm and body, and opening and closing of the end-effectors are possible.






The second generation languages include AML, RAIL, MCL, VAL II etc. They are structured programming languages performing complex tasks. Apart from straight line interpolation, complex motions can be generated with the second generation languages. They can handle both analog and digital signals besides the binary signals. Force, torque, slip and other sensors can be incorporated on the joints, wrist or the gripper fingers and the robot controller is capable of communicating with such sensory devices so that better motion control can be effected.

In case of faults or errors, the first generation robot will probably stop functioning if it cannot cope with the situation. But robots with second generation language programming can recover in the event of malfunction, probably by activating some other programs. This is of course, low level intelligence. Robots programmed with second generation languages behave somewhat intelligently because of enhanced sensory capabilities. For example, with the aid of expert systems (knowledge-based programs), some decision-making programs can be kept in the robot's memory to find some means of getting out of the malfunctioning.

Second generation languages have the added advantage of better interacting 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.



A more advanced future language is 'world' modelling. In this case, a task is defined through a command, say 'TIGHTEN THE NUT'. The robot should be capable of performing step by step functions to accomplish the objective of tightening the nut. This is possible only if the robot is capable of seeing the world around it. The three-dimensional model of the work environment around the robot must be understood. The robot must find the nut and the spanner, pick them up and place them in a sequential manner and finally tighten the nut with the aid of the spanner. Intelligence is required in this case and the robot should be capable of making decisions. Future generation robot languages involve technology of artificial intelligence and hierarchical control systems. It is hoped that it will be possible to develop complete off-line robot programming through world modelling and a high level object oriented command (say TIGHTEN THE NUT) will be obeyed by the robots.



# COMPUTER CONTROL AND ROBOT SOFTWARE

---

For performing a specific task, a robot is required to move in proper sequence. The robot can be taught and programmed through teach pendant. But it is difficult to have close control through pendant teaching. So textual programming is attempted and the computer instructions are given following the syntax of a certain robot language. The program and control methods are actuated through software running on an operating system in which manipulation of data takes place. Monitors are used to activate control functions.

In a robot, there are three basic modes of operation:

1. Monitor mode
2. Run mode or execute mode
3. Editor mode





## **Monitor Mode**

In the monitor mode, the programmer can 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, enable or disable, and move back and forth into its edit and run mode.

## **Edit Mode**

In the edit mode, 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 can replace them by new lines. In this mode, any error if shown on the monitor can be corrected. However, to come out of the edit mode, an end command, say (E) should be given.





## Run or Execute Mode

The programs to carry out a predefined task can be executed in the run mode. The sequential steps as written by the programmer are followed during the run mode. Sometimes dry run can be tested by making the switch disable. For example, when arc welding is done, the trajectory can be tested by dry run after the weld signals are made non-operational. The signals are made non-operational by the disable switch. 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. Suppose the robot has been programmed to describe a path beyond the defined envelope, the robot cannot move and the monitor will exhibit some error message. The path or the coordinate points of locations are to be redefined and corrected in the edit mode. Then after ending the edit mode, the run mode may be actuated. The robot will run following the correct trajectory.



## **Introduction to VAL**

VAL is an example of a robot programming language and an operating system which regulates and controls a robotic system by following the user commands and instructions. It has been designed to be highly interactive to minimize programming time.

The VAL operating system, or monitor is stored in ROM in the controller, and thus VAL is immediately available when the controller is powered ON. This system monitor is responsible for the control of the robot, and it can accept commands either from the manual control unit, or, the system terminal, or from user programs stored in RAM. Its versatility and flexibility are further increased by the fact that it can perform most of its commands even while a user program is being executed.



The instructions of VAL language can be combined to describe complete robot tasks. Normally, the motions and actions of the robot are controlled by a program stored in RAM. Such robot control programs are created by the user and are called 'user programs' or 'VAL programs'. VAL also contains an editor allowing the user to create or modify (edit) robot control programs. In addition, the editor has a simple mode of operation whereby a program step and location definition are automatically generated each time a button, RECORD on the manual control unit (teach pendant) is pressed.

VAL programs can also include subroutines, which are separate programs. Each subroutine can call other subroutines with up to ten such levels possible. Once a user program has been entered into the VAL system, it is possible to execute it any number of times.



## Locations

Robot locations refer to the data that represents the position and orientation of the robot tool (end point). VAL has got two ways of representing robot locations. One way is to express a location in terms of the positions of the individual robot joints, which is then called a *precision point*. Another way is to express in terms of the cartesian coordinates ( $x$ ,  $y$ ,  $z$ ) and orientation angles of the robot tool relative to a reference frame fixed in the robot base. Such locations are called *transformations* and provide a more intuitive representation of locations than do precision points, since the components are easily related to the workspace.

Compound transformations are also available in VAL which define a location relative to other locations and are written as strings of transformation names separated by colons.





## Trajectory Control

VAL uses two different methods to control the path of a robot from one location to another. The methods either (i) interpolate between the initial and final position of each joint, producing a complicated tool-tip curve in space, or (ii) move the line instruction is a complete statement. The robot follows each line instruction sequentially until the program commands a return or jump to another instruction line identified by a level.



## Monitor Commands

To enter and execute a program, one has to give certain VAL commands, called monitor commands. VAL commands also can define locations, list program and location data, store and retrieve user programs and location data on diskettes and can also determine and control the current state of the VAL system. A complete listing of all such commands can be seen in 'User's Guide to VAL'. Here, only some specific commands are given. VAL commands can be divided into the following categories: (i) defining and determining locations (ii) editing programs (iii) listing program and location data (iv) storing and retrieving program and location data and (v) program control.



## Other Monitor Commands

Additional monitor commands are available in VAL II for storing programs onto disks (STORE), copying programs (COPY), loading files from disk to computer memory (LOAD), listing the file names contained on a disk (FLIST), renaming (RENAME) or deleting (DELETE) files, and so on.

One final monitor command that will be mentioned is the DO command. The DO command causes the robot to execute a specified programming instruction. For example,

DO (programming instruction)

causes the robot to carry out the particular programming instruction indicated. We will give examples of the use of this command in the following section.



## ***Defining and Determining Locations***

Location variables can be set equal to the current location or a previously defined location by HERE and POINT command. The current location can be displayed using WHERE command. TEACH command records a series of location values under the control of RECORD button on manual control unit. The following example shows how to teach a position PART by HERE Command, and the resulting positions displayed on the screen.

HERE		PART	or	HERE	P1
X/JT1	Y/JT2	Z/JT3	O/JT4	A/JT5	T/JT6
248.63	592.97	148.53	141.141	30.822	1.225

To define a position, a command like

POINT PART = P1

may be given where a location variable PART is equal to the value P1.





A command, say,

TEACH P1

is used to record a location variable P1 when the record button on the teach pendant is pressed. Successive location variables can be assigned P1, next P2, next P3 and so forth by teaching new locations on the path and pressing the record button each time. The motion path is taught by the command TEACH.



There are several commands in VAL II for defining locations and determining the current location of the robot. In the following commands all distances and coordinate values are in millimeters. One of the methods of defining point locations is the HERE command. The operator uses the teach pendant to move the robot manipulator to the position to be defined and uses the command as follows:

#### HERE P1

This command, given in the monitor mode, defines the variable P1 to be the current robot arm location. A related command is

#### WHERE

This command queries the system to display the current location of the robot in Cartesian world coordinates and wrist joint variables. It also displays the current gripper opening if the robot is equipped with a position-servoed hand.

The TEACH command is used to record a series of location values under the control of the record button on the teach pendant. Each time the record button is pressed, a location variable is defined and given the value corresponding to the location of the robot at the instant the record button is pressed. Each successive location variable is automatically assigned a new name. The assigned name is derived from the name specified in the command. For example, if the command

#### TEACH P1

is typed into the monitor, the first recorded location variable is P1, the next is P2, the third is P3, and so forth. It is possible to teach a complete motion path by successively positioning the robot using powered leadthrough with the teach pendant and pressing the record button.

A third command for defining positions is the POINT command. For example, the command

#### POINT PA = P1

sets the value of PA (a location variable) equal to the value of P1.



## ***Motion Control***

*MOVE* moves the robot to a specified location.

*MOVES* moves the robot in a straight line path.

*DRAW* moves the robot along a straight line through specified distances in X, Y and Z directions.

*APPRO* moves the robot to a location which is at an offset (along tool Z-axis) from a specified point.

*DEPART* moves the tool along the current tool Z-axis.

*APPROS* or *DEPARTS* do the same as *APPRO* or *DEPART* instructions, but along straight line paths.

*CIRCLE* moves the robot through circular interpolation via three specified point locations.



# MOTION COMMANDS

There are a number of programming instructions used to define the motion path in VAL II. The basic instruction is

**MOVE P1**

which causes the robot arm to move by a joint-interpolated motion to the point P1. A related motion command is

**MOVES P1**

which causes the movement to be along a straight line path from the current location to the point P1.





VAL II has approach and depart commands which are written as follows

APPRO P1, 50

MOVE P1

DEPART 50

The first command moves the tool to position and orientation specified by P1, but offset from the point along the tool  $z$  axis by a distance of 50 mm. The MOVE command moves the tool to point P1. The DEPART instruction moves the tool away from P1 by 50 mm along the tool  $z$  axis. In this sequence, the tool  $z$  axis direction is defined by the orientation of the tool in the P1 specification. The APPRO and DEPART statements are executed in joint-interpolated motions. Both commands can be carried out with straight line motions by specifying APPROS and DEPARTS statements, respectively.



The speed setting can be changed during the program by means of the SPEED command. The command applies to all subsequent motions until the speed is again altered. There are two ways to specify speed in a VAL II program. The first method is to specify a numerical value without a units argument.

**SPEED 60**

This is interpreted as a percentage of the speed specified in the monitor SPEED command, as explained above. The alternative way is to specify

**SPEED 15 IPS**

which indicates that the subsequent motion commands are to be executed at a speed of 15 in./s (IPS). Units can also be specified as millimeters per second (MMPS).

A single joint can be changed by a certain amount with the statement

**DRIVE 4, -62.5, 75**

This command changes the angle of joint 4 (assumed to be a rotational joint) by driving it 62.5° in the negative direction at a speed of 75 per cent of the monitor speed.



Execution of the program requires specification of speed. The SPEED command in the VAL II monitor mode is given prior to program execution in the following way

**SPEED 50**

This specifies that the robot execution speed is 50 units on a scale which ranges between 0.39 (very slow) and 12800 (very fast), where 100 is the 'normal' speed. Thus, a setting of 50 would be below normal speed. The actual speed at which the motion sequence is executed is related to the product of the speed setting defined by this monitor command and the speed specified within the program by the SPEED instruction. For example, if the monitor speed setting is 50 (as above), and the SPEED is specified in the program as SPEED 60, then the actual speed of the motions following the command will be 30 per cent of normal speed.



Another command in VAL II used for motion control is to align the tool or end effector for subsequent moves. The statement is simply

### ALIGN

This causes the tool to be rotated so that its z axis is aligned parallel to the nearest axis of the world coordinate system. It is useful for lining up the tool before a series of locations are taught in which it is important that the tool be properly oriented. This is done in the monitor mode with the DO command as follows

### DO ALIGN

The DO command can also be used with other motion commands, as in DO MOVE P1 and DO DRIVE 4, -62.5, 75.





## **Hand Control**

*OPEN* and *CLOSE* indicate respectively the opening and closing of the gripper during the next instruction.

*OPENI* and *CLOSEI* carry on the same functions, but immediately.

*CLOSEI 75* In VAL II, if a servo-controlled gripper is used, then this command causes the gripper to close immediately to 75 mm. A gripper closing command may also be given by

GRASP 20, 15

The above command causes the gripper to close immediately and checks whether the opening is less than the amount of 20 mm. If the opening is less than the amount of 20 mm, the program, branches to the statement 15.

*MOVEST PART, 30* indicates that the servo controlled end-effector causes a straight line motion to a point defined by PART and the gripper opening is changed to 30 mm.

*MOVET PART, 30* causes the gripper to move to position, PART with an opening of 30 mm by joint-interpolated motion.



The basic commands are OPEN and CLOSE. With a pneumatic binary gripper, these commands cause the gripper to assume fully open and fully closed positions. These commands are executed during the next robot motion. The commands OPENI and CLOSEI cause the commands to be executed immediately rather than during the next motion.

With a servocontrolled gripper, the finger opening can be specified as follows

OPEN 75

.

.

.

CLOSEI 50

The first statement causes the gripper to open to 75 mm during the next motion of the robot, while the second statement causes the gripper to close immediately to 50 mm.



A related statement is GRASP. This allows a check to be made to determine if the gripper has closed by an expected amount. The command would work as follows:

GRASP 12.7, 120

The command causes the gripper to close immediately, and checks if the final opening is less than the specified amount of 12.7 mm. If it is, the program branches to the statement 120 in the program. If the statement number is not specified, for example,

GRASP 12.7

then an error message is displayed at the operator's console. The GRASP statement provides a convenient method of grasping an object and testing to make sure that contact has been made.



The control of the gripper and simultaneous movement of the robot arm can be accomplished with a single statement rather than two separate commands as described above. The required statement is

**MOVET P1, 75**

This command generates a joint-interpolated motion from the previous position to the point P1, and during the motion, the hand opening is changed to 75 mm. The corresponding straight line motion command is

**MOVEST P1, 75**

These statements assume a servocontrolled gripper capable of responding to the 75-mm opening specification. For a pneumatically operated hand, the command is interpreted to mean 'open' if the value specified is greater than zero, and 'close' if





## ***Editing Programs***

EDIT permits to create or modify (edit) a user program. There are several subcommands in EDIT to properly edit a program. A typical command for editing is,

### ***Storing and Retrieving Program and Location-data***

The command that displays the file directory of the diskette is

LISTF

The specified programs, location and both programs and locations can be stored respectively in a program file and a location file by entering the commands

STOREP  
STOREL  
and STORE

The commands that can be used for loading the programs, locations and both programs and locations respectively contained in a specified disk into the system memory are

LOADP  
LOADL  
and LOAD



In VAL II, an additional command can be constructed as

FLIST      – for listing the file names kept on a disk.

Besides, VAL and VAL II can accept commands

COPY      – for copying the program

RENAME – for renaming the files

DELETE – for deleting the files.



## Program DEMO. A

1. APPRO PART, 50
2. MOVES PART
3. CLOSEI
4. DEPARTS 150
5. APPROX BOX, 200
6. MOVE BOX
7. OPENI
8. DEPART 75

. END



The name of the program is DEMO. A

1. Move to a location, 50 mm above the location PART (PART is a location to be defined).
2. Move along a straight line to PART
3. Close the gripper jaws to grip the object immediately
4. Withdraw 150 mm from PART along a straight line path
5. Approach along a straight line to a location 200 mm above the location, BOX (BOX is to be defined later)
6. Move to BOX
7. Open the hand (and drop the object)
8. Withdraw 75 mm from BOX

In the above example, steps 1, 6 and 7 are examples of joint-interpolated motions, while steps 2, 4 and 5 are examples of straight line motions. Steps 3 and 7 contain hand control instructions.





## ***Robot Configuration Control***

Any robot configuration change is accomplished during the execution of the next motion instruction other than a straight line motion.

*RIGHTY* or *LEFTY* change the robot configuration to resemble a right or left human arm respectively.

*ABOVE* or *BELOW* commands make the elbow of the robot to point up or down respectively.



## ***Location Assignment and Modification***

The instructions that do the same as the corresponding monitor commands are

SET      and      HERE

## ***Program Control, Interlock Commands and Input/Output Controls***

*SETI* sets the value of an integer variable to the result of an expression.

*TYPEI* displays the name and value of an integer variable.

*PROMPT* In VAL II, this command often helps the operator to respond by typing in the value requested and pressing the return key.

For example,

PROMPT "Enter the Value:", Y1

indicates the quotations on the CRT and the system waits for the operator to respond by assigning some value to variable name Y1 and there the program is executed.



*GOSUB* and *RETURN* are necessary to transfer control to a subroutine and back to the calling program respectively.

A *VAL* subroutine is a separate user program which is considered as a subroutine when *GOSUB* instruction is executed.

*IF... THEN* transfers control to a program step depending on a relationship (conditions) being true or false.

*PAUSE* terminates the execution of a user program.

*PROCEED* The user program that is held back by *PAUSE* Command can be resumed from the point by entering this command.

*SIGNAL* turns the signals ON or OFF at the specified output channels.

*IFSIG* and *WAIT* test the states of one or more external signals.


*SIGNAL* command is helpful to communicate with the peripheral equipment interfaced with robot in the work-cell.

*RESET* turns OFF all external output signals.

The command, say,

*SIGNAL 2, -3*

indicates that output signal 2 (positive) is to be turned ON and output signal 3 (negative) is to be turned OFF.



The command, say,

SIGNAL 2, -3

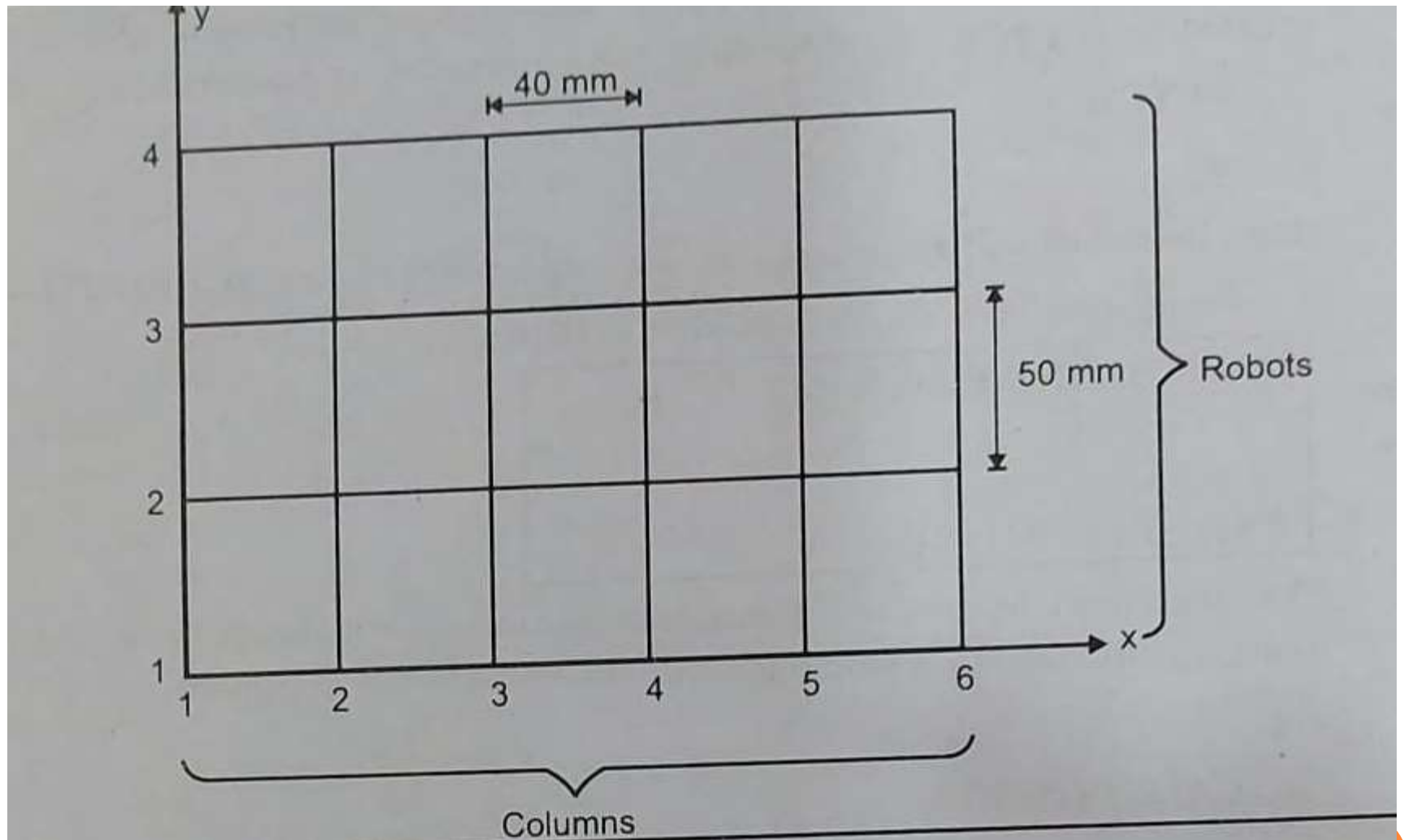
indicates that output signal 2 (positive) is to be turned ON and output signal 3 (negative) is to be turned OFF.

WAIT SIG (-1, 2)

will prevent the program execution until external input signal 1 is turned OFF (negative) and external input signal 2 is turned ON (positive).







### Some additional parameters :

MAXCOL	–	The number of columns on the pallet
MAXROW	–	The number of rows on the pallet
YSPACE	–	The spacing in the Y-direction (i.e. spacing between rows)
XSPACE	–	The spacing in the X-direction (i.e. spacing between columns)
Subroutine	–	PALLET

The subroutine uses the above parameters as arguments that must be identified when the subroutine is called.

SIGNAL 1 (i.e. output line 1) is used to initiate the delivery of an empty pallet to the loading position.

WAIT 11 (i.e. input line 11) is used in conjunction with a sensor device to ensure the pallet delivery has been accomplished.

SIGNAL 2 (i.e. output line 2) is used to initiate the removal of the filled pallet from the loading position.

WAIT 12 (i.e. input line 12) tests to make sure that it has been removed.



**Program :**

```
PROGRAM PALLETIZE
DEFINE PICKUP = JOINTS (1, 2, 3, 4, 5)
DEFINE CORNER = JOINTS (1, 2, 3, 4, 5)
DEFINE DROP = CO-ORDINATES (X, Y)
OPEN I
5  SIGNAL 1
   WAIT 11
   CALL PALLET (MAX COL = 6, MAXROW = 4, XSPACE = 40, YSPACE = 50)
   SIGNAL 2
   WAIT 12
```



```
GOTO 5
END PROGRAM.
SUBROUTINE PALLET (MAXCOL, MAXROW, XSPACE, YSPACE)
  ROW = 0
10  Y = ROW * YSPACE
    COLUMN = 0
20  X = COLUMN * XSPACE
    DROP = CORNER + <X, Y>
    APPRO PICKUP, 50
    MOVES PICKUP
    CLOSE I
    DEPART 50
    APPRO DROP, 50
    MOVES DROP
    OPEN I
    DEPART 50
    COLUMN = COLUMN + 1
    IF COLUMN LT MAXCOL GOTO 20
    ROW = ROW + 1
    IF ROW LT MAXROW GOTO 10
  END SUBROUTINE
```



The additional command

### REACT-VAR 2, SUB TRAY

indicates that the reactions are invoked if the external binary signal identified is a negative variable, VAR 2. If the current state of signal is OFF, signal is monitored for a transition from OFF to ON and then again OFF. When the reaction or specified signal transition is detected, the program control is transferred to the subroutine named TRAY.

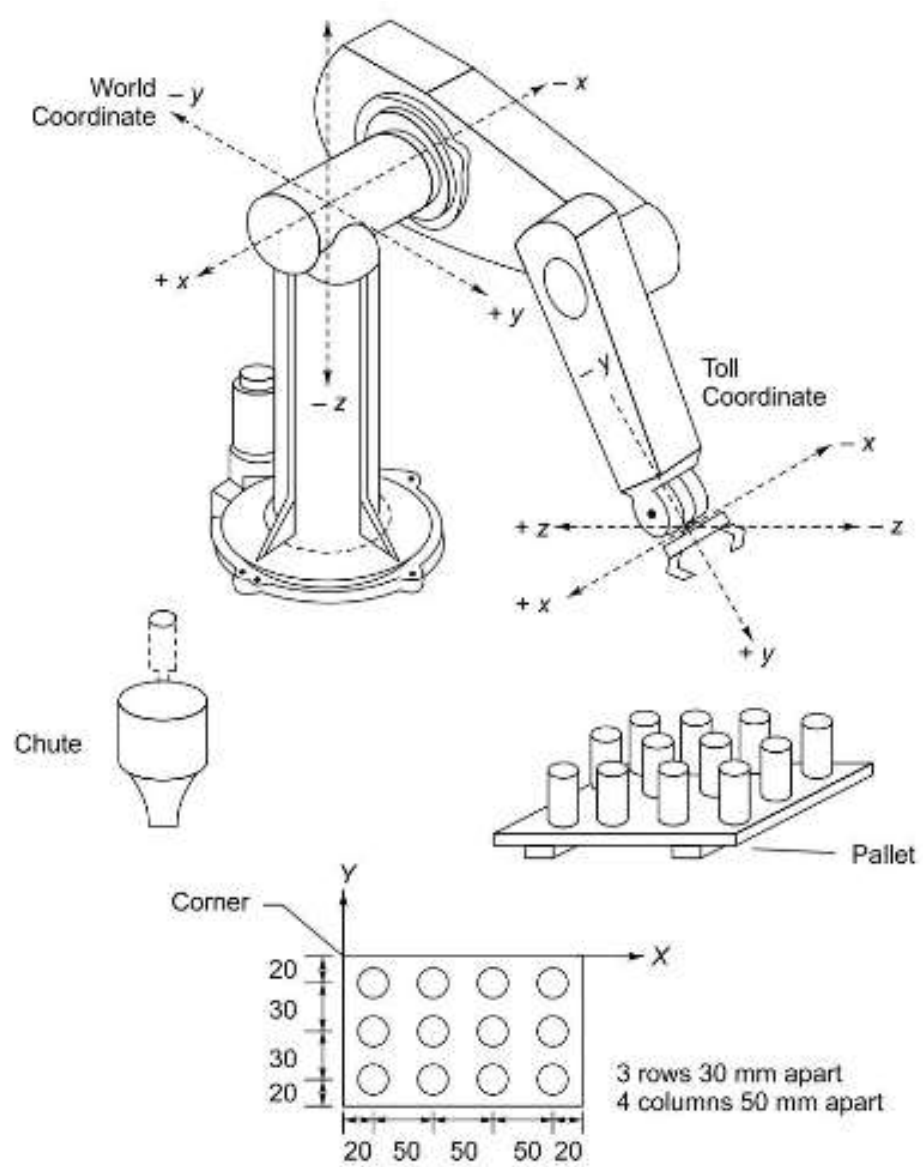
REACTI interrupts robot motion immediately.

VAL II may communicate with either digital or analog signals through input/output modules.

### IOPUT and IOGET

are the commands that are used either to send or receive output respectively to a digital I/O module.





```
. PROGRAM DEPALLET 1
  REMARK PROGRAM TO PICK OBJECTS FROM A PALLET
  REMARK CORNER AND CHUTE LOCATIONS ARE TAUGHT
    SETI MAXCOL = 4
    SETI MAXROW = 3
    SETI ROW = 1
    SETI COLUMN = 1
    SET PICK = CORNER
    SHIFT PICK BY 20.00, -20.00, 60.00
    OPENI
10    MOVE PICK
    DRAW 0, 0, - 25.00
    CLOSEI
    DRAW 0, 0, 25.00
    MOVE CHUTE
    OPENI
    GOSUB PALLET
    IF ROW LE MAXROW THEN 10
. END
. PROGRAM PALLET
```



```
REMARK SUBROUTINE FOR LOCATIONS
      SETI COLUMN = COLUMN + 1
      IF COLUMN GT MAXCOL THEN 20
      SHIFT PICK BY 50.00,0.00, 0.00
      GO TO 10
20     SETI ROW = ROW + 1
      IF ROW GT MAXROW THEN 30
      SHIFT PICK BY -150.00, -30.00, 0.00
      SETI COLUMN = 1
30     RETURN
. END
```





## Welding Instructions

Robot welding provides good weldment through controlled weld speed, welding voltage and welding current. Optimal weld conditions can be set to have better welded joints. Complex curves can be generated by welding programs. Straight bead, three point weave and trapezoidal (five point) weave patterns can be generated for multipass and multilayered welding. A few welding commands are illustrated in this section.

WSET instructions sets the speed, welding voltage and current as a welding condition identified by a number (1–4). For example,

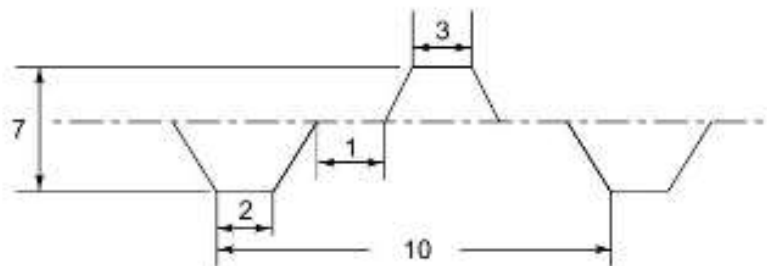
$$\text{WSET } 1 = 13, 54.3, 63$$

sets a welding speed of 13 mm/s, welding voltage of 54.3% and welding current of 63% as welding condition 1.

WVSET sets a weaving pattern, setting some or all of the following parameters: cycle distance, amplitude, right end stop distance, right end stop time, center stop distance, left end stop distance and left end stop time. All except the first two are optional. The weaving pattern for the instructions

$$\text{WVSET } 1 = 10, 7, 2, 0, 1, 3, 0$$

is illustrated in Fig. 6.3.

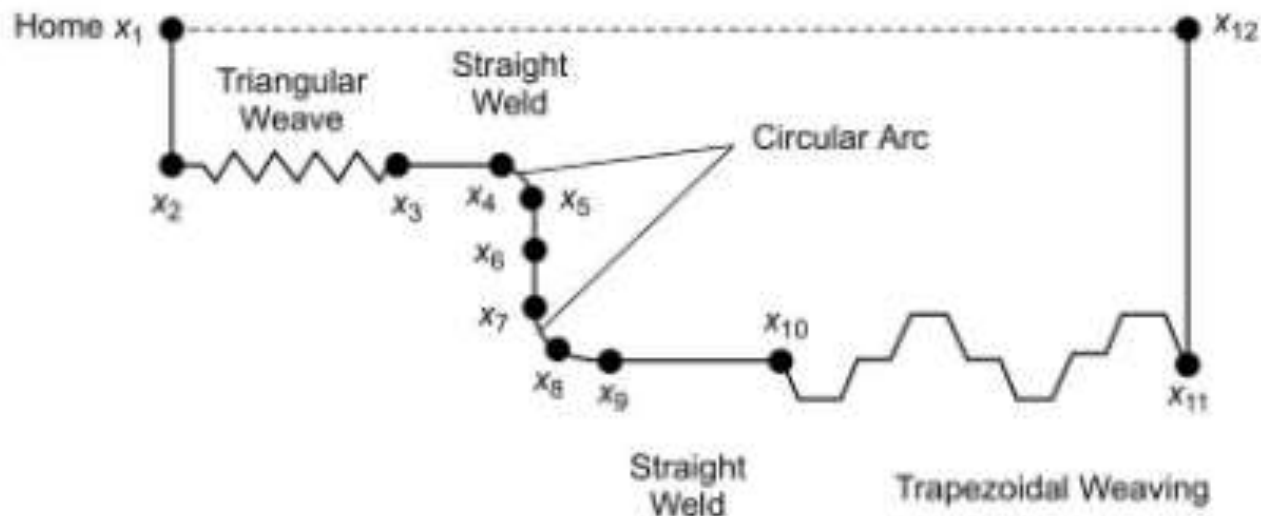


*The trapezoidal weave pattern*

WSTART starts welding under present welding condition and weaving condition (set by WSET and WVSET respectively). WEND inactivates a welding start signal. CRATERFILL instruction is used when a crater filler is required at a welding end.

### Example

A weldment is to be made. The weld trajectory is a continuous path arc welding along the paths  $X_2 - X_3$  with triangular weaving,  $X_3 - X_4$  with straight weld,  $X_4 - X_5 - X_6$  with circular interpolation,  $X_6 - X_7$  with straight weld,  $X_7 - X_8 - X_9$  with circular arc,  $X_9 - X_{10}$  with straight weld and  $X_{10} - X_{11}$  with five point weaving. The weld torch begins its movement from home position  $X_1$  and departs to location  $X_{12}$ . Craterfilling is done at the end of trapezoidal weaving. Write a VAL program for suitable arc welding.

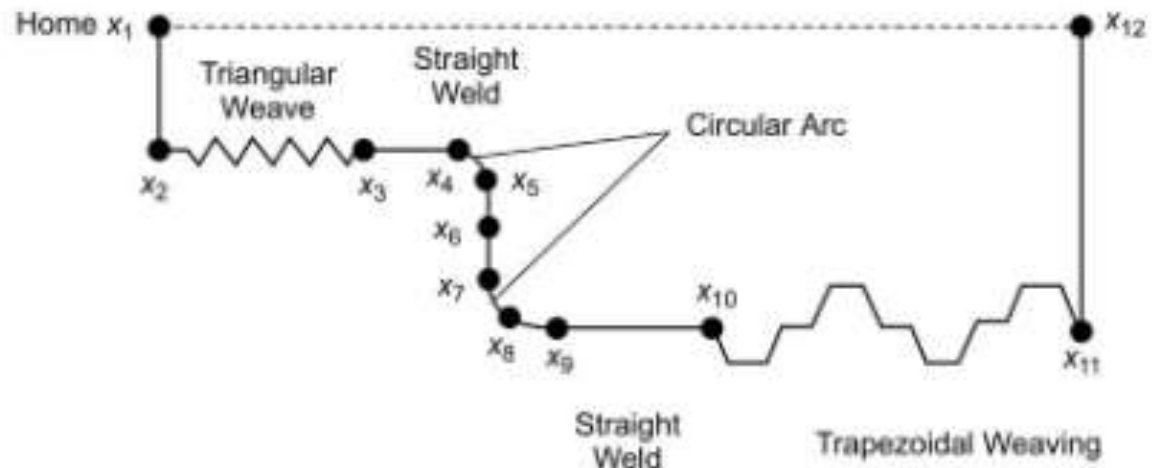


# PROGRAM WELD CURVE

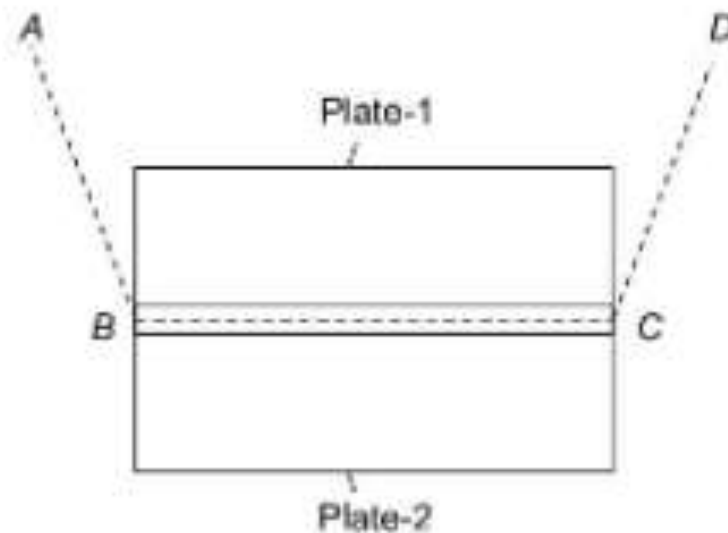
```

1 WSET 1 = 10, 40, 50
2 WSET 2 = 8, 35, 60
3 WSET 3 = 12, 40, 55
4 WVSET 1 = 5, 5
5 WVSET 2 = 10, 7, 2, 0, 1, 2, 0
6 MOVE XI
7 MOVEX2
8 WSTART 1, 1
9 MOVES X3
10 WEND 0.5
11 WSTART 2
12 MOVES X4
13 CIRCLE X4, X5, X6
14 MOVES X7
15 CIRCLE X7, X8, 9
16 MOVES X10
17 WEND 0.5
18 WSTART 3, 2
19 MOVES XI1
20 CRATERFILL 0.8, 3
21 WEND 0.5
22 MOVE X12
END

```



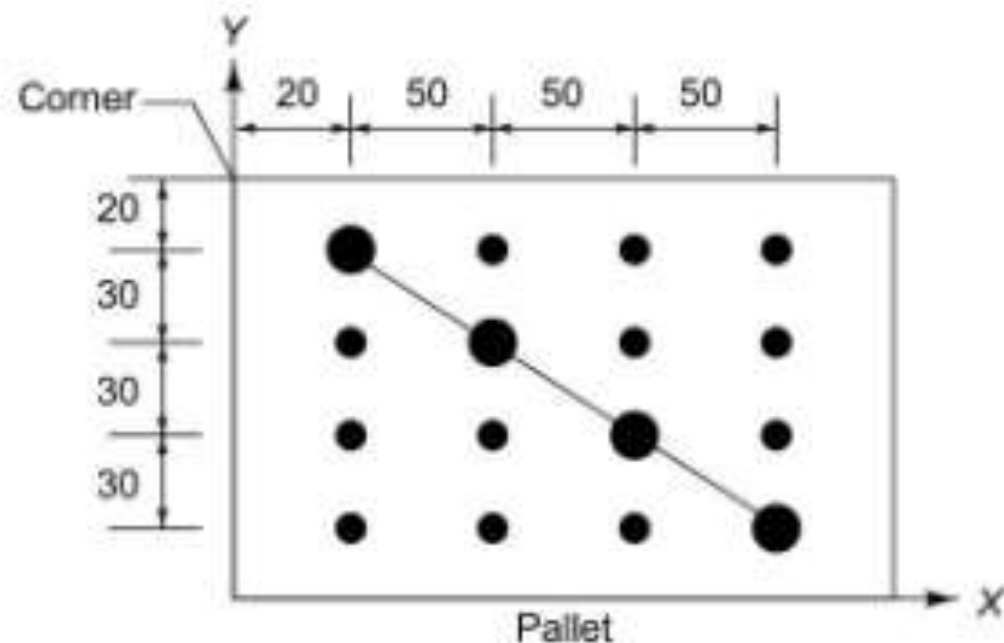
- 6.1 Two plates of 5 mm thickness are to be welded with square butt joint as shown in Fig. 6.5. The welding is straight weld. The welding torch should start from Position A, move to B, continue with continuous arc welding along BC in a straight line and then move to position D. Write a VAL program in world-coordinates.



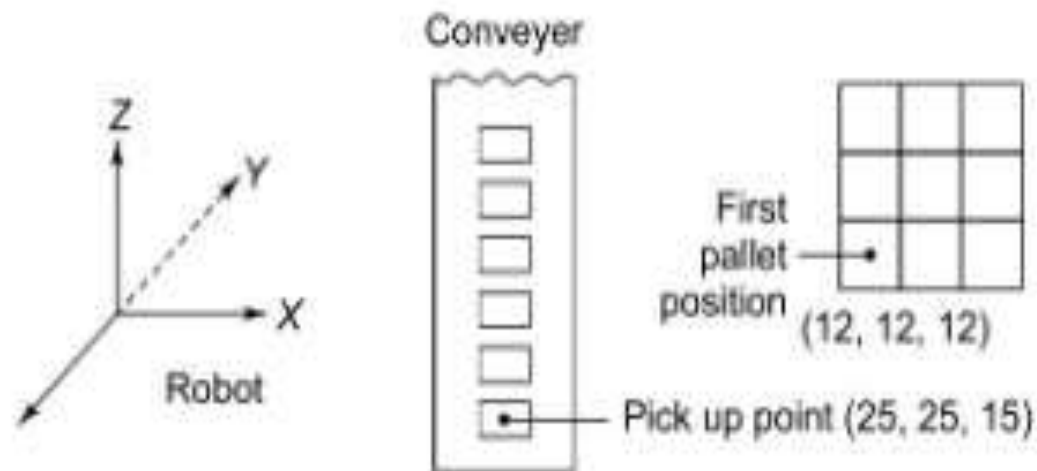
**FIG. 6.5** *Figure of exercise Problem 6.1*



6.5 A robot is engaged to unload the objects from the pallet along the diagonal as shown in Fig. 6.8. Assume that there are 4 rows and 4 columns. Rows are 30 mm apart and columns are 50 mm apart. The plane of the pallet is assumed to be parallel to  $x$ - $y$  plane. The rows are parallel to  $x$ -axis and the columns are parallel to  $y$ -axis. Write a program to pick up the objects along the diagonal only, starting from the left top to right bottom.



**FIG. 6.8** Figure of exercise Problem 6.5



**FIG. 6.9** *Palletizing*

- 6.10 Write a program to pick the blocks off a conveyor belt and place them in a pallet in  $3 \times 3$  array positions as shown in Fig. 6.9. The blocks are precisely positioned on the conveyor, and conveyor stops at the pick up points. Conveyor is reactivated manually.