6. Robot Languages and Programming

6.1. ROBOT LANGUAGES

Robot languages have been developed for ease of control of motions of robots having different structures and geometrical capabilities. Some of the robot languages have been developed by modifying the existing general-purpose computer languages and some of them are written in a completely new style. Programming languages have been developed by the pioneering efforts of various researchers at Stanford Artificial Intelligence Laboratory, research laboratories of IBM Corporation, under U.S. Air Force sponsorship, General Electric Co., Unimation and many other robot manufacturers.

Bonner classified robot programming languages into five distinct levels to indicate the basic features of each language. The levels are as follows:

(a) Microprocessor/microcomputer level, (b) P-t-P level, (c) Motion level, (d) Structural programming level, (e) Task-oriented level.

In the microprocessor level, traditional computer languages like C or assembly language is used. Such language is used due to hardwired or physical structure of the robot joints.

In the Point-to-Point (P-t-P) level, the robot joints are moved through a series of points in the work envelope by either a teach pendant or by manual movement through appropriate points in the work space as is done in case of teaching of a spray painting robot. The points are stored, and the stored program if required can be edited. External signals can also be interfaced. Mostly T3 type of robot follows such programming technique. A simple robot using p-t-p control may be programmed as

1. GO TO POINT C, STOP.
2. GO TO POINT D, STOP.
3. OPEN GRIPPER.

In the motion level of robot programming, P-t-P motions can be implemented. Besides, branching, subroutines, and sensing capabilities representing a motion to a frame are some of the features available. RPL, SIGLA, and VAL are some of the languages included in this group.

In the structural programming level, extensive use of coordinate transformations and frames are possible. Complex data structures, sensors processing and parallel processing capabilities are the features of the robot languages. Constructs as 'if… then…. else' and 'while …. do' provide powerful control aids and data structures. The definable subroutines can be accommodated. Sensor commands in the task level languages are similar to those at the motion level. AL, MCL, PAL, HELP, and AML are some of the structured robot programming languages.

In the taskoriented level of robot programming, high-level commands such as PLACE object 1 on object 2 is given. This provides world modeling system to keep the track of objects. PLACE commands involve identifying object 1 and object 2, determining approach and pick up for object 1, placing object 1 on object 2, and recording the relationship between them. AUTOPASS command belonging to this group follows the following types of commands such as (1) state change (such as PLACE), (2) tool statements (such as OPERATE), (3) fastener statements (such as RIVET), and miscellaneous statements (such as VERIFY).

Some of the programming languages are WAVE, AL, VAL, AML, MCL, RAIL, HELP, RPL, PAL, and ADA. These are briefed with their special features.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

6.1.1. 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.

6.1.2. 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.

6.1.3. 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.

6.1.4. MCL

US Air force ICAM project led to the development of another manufacturing control language known as MCL by McDonnel-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.

6.1.5. 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.

6.1.6. HELP

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

6.1.7. 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.

6.1.8. 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.

Besides these, there are some other languages like PAL, ADA etc. PAL has been written by Richard Paul by modifying WAVE and incorporating features of PASCAL. But the representations of syntaxes used in the program are difficult to handle. ADA developed by the Department of Defence (DOD) in USA is a real-time system that can be run on several microcomputers like Zilog, VAX, Motorola 68000, etc. ADA is convenient for controlling the robots used in a manufacturing cell.

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.

6.1.9. 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.

6.2. 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

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

6.3. 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

The above modes constitute the operating system.

6.3.1. Monitor Mode

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

6.3.2. 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.

6.3.3. 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.

The operating system for implementing robot language program uses either an interpreter or a compiler. An interpreter takes the source program one line at a time and generates equivalent code that is understood by the controller. Any mistake in the source program, will be indicated to the user. The user can correct the source program and the line is reinterpreted. Every line is executed by the interpreter as and when it is written. So an interpreter works slowly while executing the program as it may reinterpret the same instruction repeatedly appearing at different lines of source program. But it is flexible. VAL is a language of PUMA dass of robots and the source codes or instructions are processed by an interpreter. Compiler is a software in the operating system that converts source code into the object code (machine code) after compilation of the whole program. The robot controller can then read and process the machine codes. The compiler passes through the entire program several times and checks all the variables, addresses, constants, etc. and produces equivalent machine codes. As it passes several times, the process is a lengthy one. A computer does not repeat to process similar programs appearing several times, but it only recopies those parts. Execution time for a compiled program is fast while editing of an interpreted program is very fast.

6.4. VAL SYSTEM AND LANGUAGE

6.4.1. 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

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

6.4.2. 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.

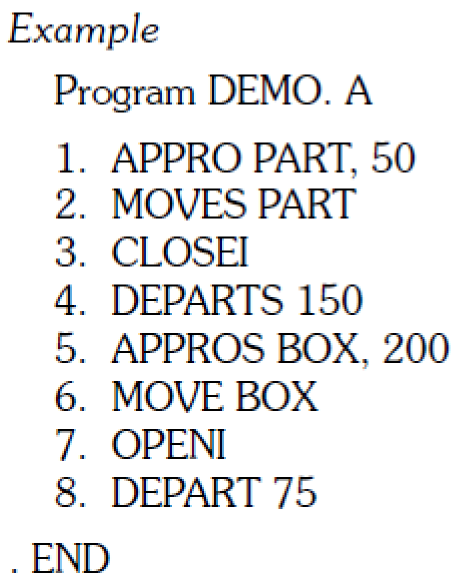
Compound transformations are also available in VAL which define a location relative to other locations and are written as strings of transformation names separated by colons.

6.4.3. 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 robot tip along a straight-line path. For the first case, called *joint interpolated motion,* the total motion time is set to that of the joint requiring the longest time to complete its motion. This provides the fastest response of the robot. On the other hand, *straight-line motion* is produced by applying an interpolating function to the world-coordinate location of the robot tool and rapidly transforming the interpolated tool location to joint commands. In this case, the motion speed of the robot tool-tip can be accurately controlled, but it has the disadvantage of being slower than corresponding joint-interpolated motions.

Figure 6.1 shows an example of a VAL program, where the meaning of each line is as follows:

Figure 6.1 The VAL program



```
Example
   Program DEMO. A
   1.  APPRO PART, 50
   2.  MOVES PART
   3.  CLOSEI
   4.  DEPARTS 150
   5.  APPROS BOX, 200
   6.  MOVE BOX
   7.  OPENI
   8.  DEPART 75
. END
```

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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. Qose 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.

However, an optional level may be given, such as

```
10      APPRO   PART, 50
```

An arbitrary level 10 may be included so that the programmer can instruct the computer to go back to this same instruction. In the example problem, the one-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.

6.4.4. 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.

6.4.4.1. 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.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

6.4.4.2. 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,

```
EDIT SRD
•
•
•
E
```

E means exit of the editing mode and return to the monitor mode.

6.4.4.3. Listing Program and Location Data

The command DIR displays the names of all user programs in the system memory. The commands

```
LISTL    and    LISTP
```

display the value of location variables and the steps of user programs respectively.

6.4.4.4. 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.
```

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

Besides, VAL and VAL II can accept commands

COPY	–	for copying the program
RENAME	–	for renaming the files
DELETE	–	for deleting the files.

6.4.4.5. Program Control

The command that specifies the speed for all subsequent robot motions under program control is

SPEED 30

The commands that execute a specified user program for once, any number of times or indefinitely are

```
EXECUTE
EXECUTE, 5 (execute five times)
EXECUTE, –1 (indefinitely)
```

The command that terminates program execution after completion of the current step is

```
ABORT
```

In VAL II, a single joint (say joint number 2) may be changed by driving it, say 60° at a speed of 30 per cent of the monitor speed. The command, for example, is

```
DRIVE   2, 60, 30
```

The DO command allows a robot to execute a program instruction, say, ALIGN which is used for motion control to align the end-effector. In order to grip some objects, the end-effector is required to align such that its Z-axis is parallel to the nearest axis of the World Coordinate System. The command is

```
DO ALIGN
```

Similarly the other DO command may be

```
DO MOVE PART   (Part is variable location)
```

6.4.4.6. Program Instructions

The program example as shown in Fig. 6.1 indicates some program instructions. This section describes some important instructions that can be included in user programs. The instructions are divided into following categories (i) robot configuration control, (ii) motion control, (iii) hand control, (iv) location assignment and modification and (v) program control, interlock commands and I/O controls.

6.4.4.7. 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.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

6.4.4.8. 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.

6.4.4.9. 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 dose 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.

6.4.4.10. Location Assignment and Modification

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

```
SET    and    HERE
```

6.4.4.11. 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
```

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

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.

GOTO 20 performs an unconditional branch to the program step identified by a given level, 20.

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.

For example,

```
IF   ROW LT 3 THEN
        .
        .
        .
(A number of instruction steps)
        .
        .
        .
       ELSE
        .
        .
        .
(A number of instruction steps)
        .
        .
        .
        END
```

If the logical expression (say Row is less than 3 in a matrix) is true, then the instruction steps between THEN and ELSE are executed. If the logical expression is false then instruction steps between ELSE and END are executed. The next program steps after END are continued.

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.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

$$\text{WAIT SIG } (-1,\ 2)$$

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

The additional command

$$\text{REACT–VAR 2, SUB TRAY}$$

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.

$$\text{IOPUT} \quad \text{and} \quad \text{IOGET}$$

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

Analog signals can also be communicated through analog input/output modules by analog to digital converter (ADC) or digital to analog converter (DAC).

For example, the commands

$$\begin{aligned}&\text{DAC } 1 = \text{SENSR } 1 \text{ (real variable)}\\&\text{DAC } 1 = \text{CONST (constant)}\\&\text{DAC } 1 = 3 + (5^{*}\text{V}) \text{ (arithmetic expression)}\end{aligned}$$

indicate that the analog output voltage is proportional to the value indicated on the right hand side. The output signal varies between some voltages, say, from –10 Volt to +10 volt. If the signal is weak, it may be amplified to lie between –10 V and +10 V. DAC sets the voltage proportional to the binary value in the range from, say, –2048 to + 2047 depending upon the analog output voltage setting.

Similarly, a command

$$\text{VAR } 1 = \text{ADC } (1)$$

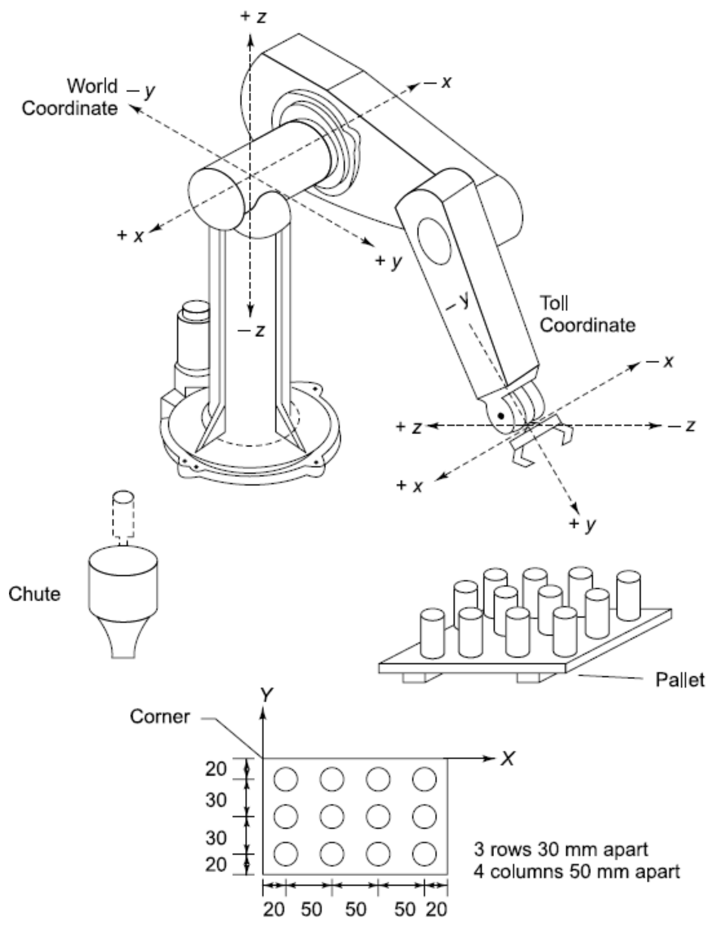
returns the current input at analog channel number 1 as an integer value in the range from –2048 to 2047.

Example: Depalletizing

In a pallet objects protruding 40 mm from the face of the pallet are located in a number of rows and columns. The pallet has 3 rows that are 30 mm apart and 4 columns that are 50 mm apart. The plane of the pallet is assumed to be parallel to the X–Y plane. The rows are parallel to X-axis and the columns are parallel to Y-axis. The objects are to be picked up one after another from the pallet and placed in a location chute. Figure 6.2 indicates the pallet.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

Figure 6.2 *Depalletizing*



```
. 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
```

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

```
                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
```

6.4.5. 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,

```
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

```
WVSET 1 = 10, 7, 2, 0, 1, 3, 0
```

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

is illustrated in Fig. 6.3.

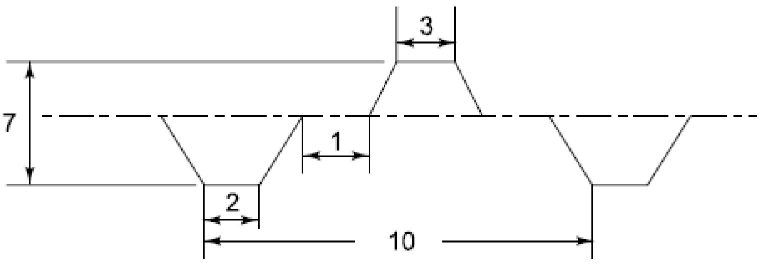


Figure 6.3 The trapezoidal weave pattern

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 pathsX_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 (Fig. 6.4)

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

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

18. WSTART 3, 2

19. MOVES XI1

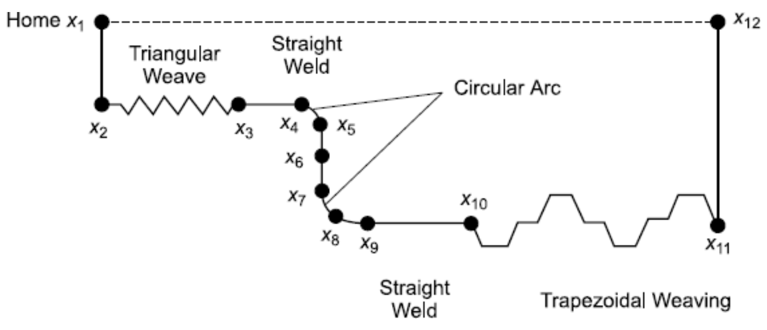
20. CRATERFILL 0.8, 3

21. WEND 0.5

22. MOVE X12

. END

Figure 6.4 The weld trajectory

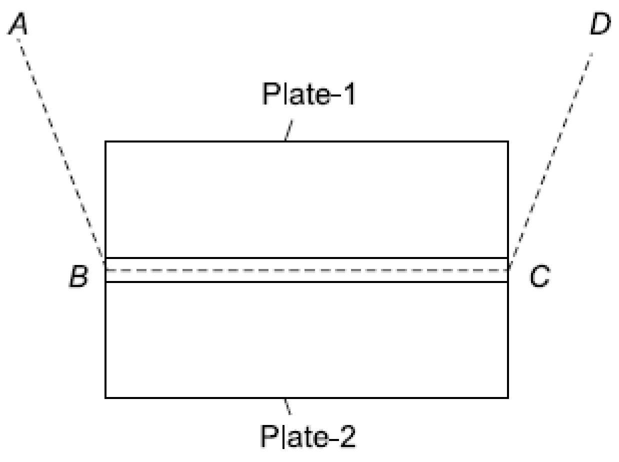


However, there are many VAL and VAL II statements like WHILE..DO..END, DO..UNTIL..and CASE..OF..VALUE..END, that are available in User's Guide or manual from Unimation Inc. and Kawasaki Unimation Japan. Some I/O signals and necessary instructions are illustrated in Chapter 7. Vision sensors and the instructions in V$^+$ language have been illustrated in Chapter 5.

6.5. EXERCISES

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.

Figure 6.5 Figure of exercise Problem 6.1



6.2 Two plates are to be joined as shown in Fig. 6.6. The welding torch is at an angle of θ = 45°. Write a program using VAL language for uniform seam width.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

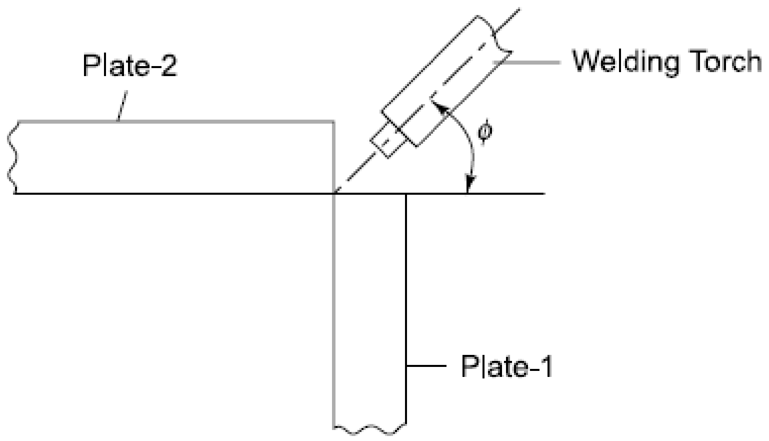
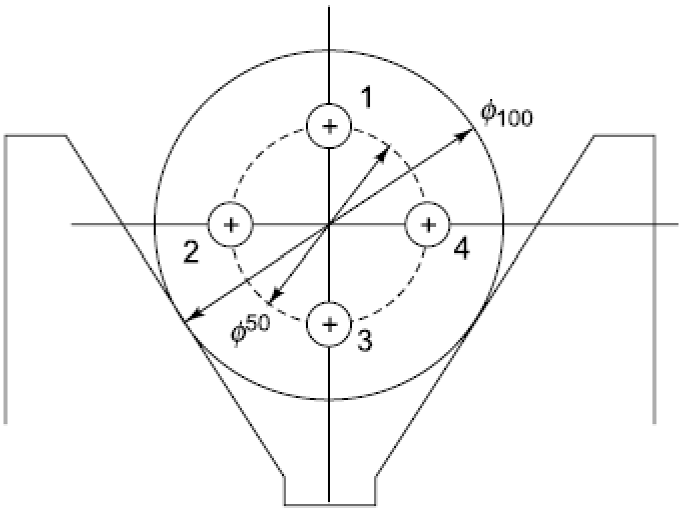


Figure 6.6 Figure of exercise Problem 6.2

6.3 Four drill holes are to be made on a circular workpiece of diameter 100 mm as shown in Fig. 6.7. Four holes are located in a circle of radius 25 mm. Write a program to bring the end-effector holding a drill of 6 mm diameter to each location of the hole in sequential order of 1, 2, 3, 4.

Figure 6.7 Figure of exercise Problem 6.3

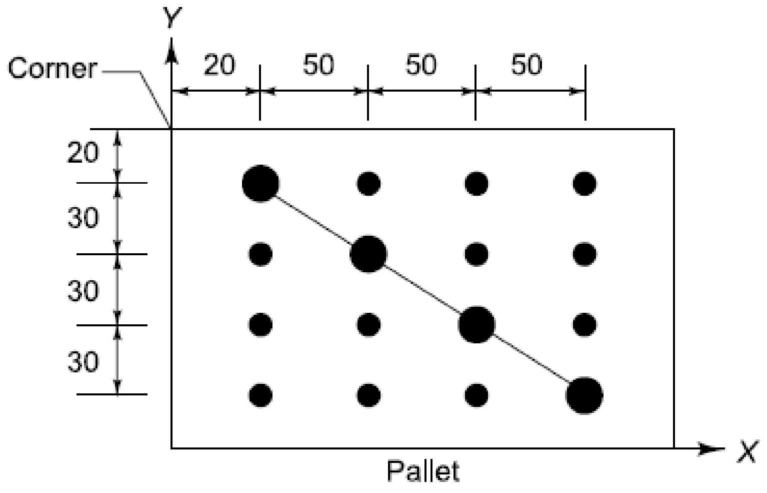


6.4 1000 parts are being inspected. The robot is loading and unloading the parts to the inspection work station fitted with sensors. The robot after loading must signal the inspection work station that the part has been loaded. The inspection work station after inspection must signal back to the robot that the part is either acceptable or rejected. The robot is sorting the parts and placing the acceptable parts in one chute and rejected parts in another chute. Write a program to accomplish the above activities. Count and print the number of accepted and rejected parts.

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.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

Figure 6.8 Figure of exercise Problem 6.5



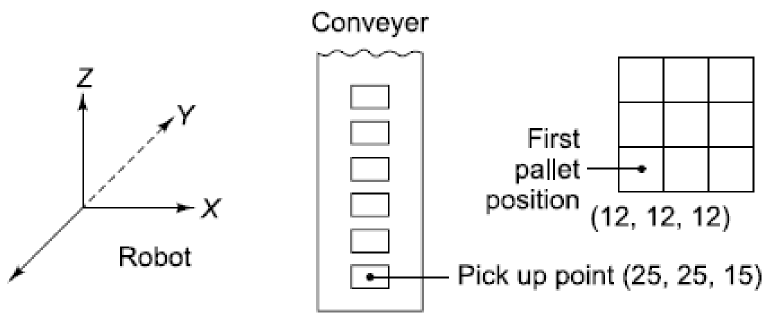
6.6 What are the different textual robot languages? Discuss their relative merits and demerits. Distinguish between an interpreter and a compiler. Name which of the languages use either interpreter or compiler.

6.7 Distinguish between first generation and second generation robot languages.

6.8 Describe briefly the robot language elements and functions.

6.9 Write a program for depalletizing operation from a pallet as shown in Fig. 6.2. Assume the corner to be located at left-bottom (instead of assuming corner at left-top).

Figure 6.9 Palletizing



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

6.6. BIBLIOGRAPHY

Adept Technology, Inc., *V+ Reference Guide*, Dec., USA, 1987.

Albus, J.S., *et al*., "Theory and Practice of Hierarchical Control", *Proceedings of the 23rd IEEE Computer Society Int. Conference,* Sept., 1981.

Automatix Inc, *RAIL Software Reference Manual Doc,* No. MN-RB-07, Oct., 1983.

Bonner, S.A., 'Comparative Study of Robot Programming Languages', MS Thesis Rensselaer Polytichnic Institute, Troy, NY, 1983.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.

Barr, A., P. Cohen and E. Feigenbaum, *The Handbook of Artificial Intelligence,* William Kaufmann, Los Altos, CA, 1981.

IBM Corporation, *A Manufacturing Language Concepts and User's Guide,* Dec., 1982 and *A Manufacturing Language Reference,* Base Publication, Boca Raton, FL, Oct. 1983.

Gossman, D.C., 'Programming of a Computer Controlled Industrial Manipalating by Guiding through the Motions', IBM Research Report, RC 6393, Yorktown Heights, NY, 1977.

Kawasaki Heavy Ind. Ltd., *User's Guide to VAL: A Robot Programming and Control System,* June, 1980.

Kawasaki Heavy Ind. Ltd., *VAL Manual: Supplement,* Feb., 1986.

Mujitaba, M.S., *et al.,* "Stanford's AL Robot Programming Language", *Computers in Mechanical Engineering,* Aug., 1982.

Taylor, R.H., et.al., "AML : A Manufacturing Language",*The International Journal of Robotics Research,* 1982.

Toepperwein, L.L., *et al.,* "ICAM Robotics Application Guide", *Technical Report AFWAL-TR-80-4042,* Vol. II, Materials Lab., Air Force Wright Aeronautical Lab., OH, April, 1980.

Unimation, Inc., A Westinghouse Co., *Programming Manual—User's Guide to VAL II,* Danbury, Connecticut, August, 1984.

Wood, B.O. and M.A. Fugelson, "MCL, the Manufacturing Control Language", *Conference Proceedings, 13th Int. Symposium on Industrial Robots,* Robots-7, Chicago, IL, April, 1983.

© McGraw-Hill Education. All rights reserved. Any use is subject to the Terms of Use, Privacy Notice and copyright information.