Chapter 6

INTRODUCTION TO ROBOT PROGRAMMING

Dr. Mayez Al-Mouhamed
Computer Engineeing Department
King Fahd University of petroleum and Minerals

This chapter introduces the main features of VAL II programming language for the PUMA-560 robot arm. First, we introduce the way VAL II allows to define the robot locations, variables, and compound transformations. The later is very important in defining structured objects. The idea consists of referencing sub-objects using frames attached to the base frame of the main object. Therefore, moving the composite object leads to update the base frame while the sub-objects frames are defined in a relative way. Second, the chapter investigate the motion instructions and their meaning. Some motion primitives generate joint interpolated motions while others generate straight-line path in the cartesian space. The later motion are more suitable and predictable but slower.

6.1 The Unimate Puma 560

The Unimate Puma 560 is a compact, computer-controlled robot for medium-to-lightweight assembly, welding, materials handling, packaging, and inspection applications.

The Series 500 is the most widely used model in the UNIMATE PUMA line of electrically driven robots. With a 36 inch reach, and 5 pound payload capacity, the PUMA Series 500 robot is designed for assembly and applications requiring high degrees of flexibility and reliability.

With thousands of units in the field, its capabilities are particularly suited to the electronics and other industries where light-to-medium weight parts handling or processed functions are carried out.

VALtm, a revolutionary advance in robot control systems, is used to control and program PUMA robots. The system uses an LSI-11 as a central processing unit and communicates with individual joint processors for servo control of robot arm motions. The results are ease in set-up, high-tolerance repeatability, and greater application versatility.

VALtm combines a sophisticated, easy-to-use robot programming capability with advanced servo control methods. Intuitive English-language instruction provides fast, efficient program generation and editing capabilities. All servo-path computations are performed in real time, which makes it possible to interface with sensory-based systems.

With it's high speed, repeatability, and flexibility, the PUMA 500 robot is suited to a wide range of small parts-handling applications, and VALtm control makes it easy to design application programs to carry out the most difficult robotic tasks.

Current assembly applications include automotive instrument panels, small electric motors, printed circuit boards, subassemblies for radios, television sets, appliances and more. Other applications include packaging functions in the pharmaceutical, personal care, and food industries. Palletizing of small parts, inspection, and electronic-parts handling in the computer, aero-space and defense industries round out the present installed base.

Puma stands for Programmable Universal Machine for Assembly. Pumas are probably the most common robot in university laboratories and designed by Vic Schienman in the mid-70's. Unimation PUMA-560 is a 6-R (revolution) type robot. It has 6 degree-of-freedom and uses DC servo motors as its actuators. The range of these angles from θ_1 to θ_6 is the following (320°, 250°, 270°, 280°, 200°, 532°). The corresponding link lengths from L_1 to L_6 are (432,432,433,56,56,37.5) mm.

The original controller LSI-11 uses the VAl robot language (Victor's Assembly Language) to communicate with the AIB (Arm Interface Board) through a bidirectional parallel bus. The AIB communicates with six digital servo boards over a DEC backplane. These boards are responsible for the execution of commands, parameter setting, and implementing the position control loop. They are connected through the backplane to the analog servo boards which implement nested velocity and current control loops.

However the VAL programming language does not support efficient real-time updating of the manipulator trajectory based on sensory data (such as from force / torque or vision sensors). Unimation has confirmed that there is an indocumented bug in the tool mode under real-time path control. During real-time control, such as alter mode, the controller does not update the rotation matrix for the tool coordinate system as the robot moves.

The features of the Unimate Puma 560 arm are the following. The arm has up to 6 degrees of motion with electric DC servo controller. The main computer is a system computer (LSI-11). It uses the teaching method by using a teach control and/or computer terminal. The programming language is VAL PLUS or VAL II. The program capacity is 8K CMOS user memory in VAL PLUS and 24K CMOS user memory in VAL II with options for additional user memory. The external program storage is a floppy disk. The gripper control is a 4-way pneumatic solenoid. The power requirement is 110-130 VAC, 50-60 Hz, 500 watts. The optional accessories are the TTY Terminal, I/O module (8 input/8 output signals, isolated ac/dc levels) up to 32 I/O capacity, pneumatic gripper without fingers, and special software packages. The repeatability of motion is +/- 0.004 in. (0.1 mm) with straight line velocity of 49 in/sec. max (1.245m/sec.). The maximum payload for static Load 2.2 lbs. (1.0 kg).

6.2 VAL robot programming

VAL II is a language with which program specifications are related to locations that specify either position in the cartesian space or frames that are attached to the objects. Information regarding the operating system environment and the multiprocessor architecture (LSI-11) under which VAL II is implemented should be developed in the laboratory. There are two frames used: (1) the world frame, and the (2) tool frame.

The World frame is fixed at the end of link-1 of the robot. For an observator placed on link 1 and looking forward to the robot hand, the Z axis goes up, the Y axis goes forward, and the X axis goes to the right.

The Tool frame is located at the robot hand with its Z axis along the hand forward direction, its X axis is going to the right of the hand, and its Y axis is going down from the hand.

6.2.1 Robot Locations

A *'Point'* or a *'Position'* is a cartesian reference ($X\ Y\ Z$) in the work space. A *'Location'* is a point plus an orientation. For example, when executing a motion instruction the robot moves so that the tool point goes to the specified destination point and the tool frame is oriented to the specified destination orientation.

1. Location Values

   There are data defining the robot locations. To define them, two methods are possible:

   (a) *Precision Points* : Location value is a precision point when the robot location is represented by the exact individual robot joint angles. The advantage is that we obtain maximum precision and there is no ambiguity regarding the robot configuration. The drawbacks are that these are robot dependent, i.e. cannot be used for other robot structures and they cannot be modified.

   (b) *Transformations*: Is a robot independent representation of position and orientation for the tool. The location consists of $(X, Y, Z, O, A, T) in which (X, Y, Z)$ are the coordinate of the Tool frame origin and angles (O, A, T) are three Euler angles defining the orientation of the Tool frame. The Tool frame is located at the robot hand with its Z axis along the hand, its X axis going to the right of the hand, and its Y axis going down from the hand. Based on the above, angle O is a ROTY for the tool frame, angle A is a ROTX, and angle T is a ROTZ.

   A location is then defined as a point (X Y Z) that is relative to the base coordinate of the robot and three angles. For example, to shift a location in the X direction, only the X component in the translation needs to be adjusted. For example, one may define a location of a part ralative to a convoyer belt located relative to the robot. Change in the coordinate of intermediate sub-objects can be made without changing the other top ranked frames. These relative tranformations are called *Compound Tranformations.* The advantage is that it can be used with other robot. It can be modified by shifting a position or an angle. One important feature is that a transformation can be a combination of relative transformation. This can be used to define a location relative to another. The drawbacks is that no information about robot configuration (obstacles) the user should include instruction to specify configurations. They have less precision than precision points.

2. **Location Variables**

   Symbols can be used to reference a location. Two types of variables (scalar and arrays) are allowed:

   (a) Variable names, these are assigned by the user to reference precision points and transformation. Variable names should start with a letter (Upper case or Lower case). Examples of location variables:

   Valid : P, Feeder, pallet. to. part. 3

Invalid: 3p, part-x, hand (reserved)
Precision points must be preceded by (#):
(# pick) is a precision point.

Note : Once location variables are defined, they can
be referenced by any program in the system memory.

(b) Array variables: These are location variables that can be of array type:

Part $\{I\}$ where $I : 0, 65535$ and Part{ } is part{0}
Using part in an Integer expression is valid.

3. Defining Variables

   (a) Under VAL II Operating System

   - using the teach pendant and recording a series of locations
   - using HERE command to assign a name to current robot location.
   - using POINT command to define locations based on other locations or from component value entered by the user.

   (b) During Program Execution

   - Using HERE command
   - Using SET command to give a value to a location variable.

6.2.2 Compound transformation

The objective of using compound transformation is to represent the robot position and orientation relative to object positions and orientations. For example, the instruction

```
MOVE plate:object:grasp
```

where (grasp) is a location relative to (object) which is a location relative to (plate). Location (plate) is relative to robot base frame of reference.

Another example is the instruction

```
APPROS base:plate:object:grasp:Trans(0,0,0,-90,90,0)
```

where APPROS is a location function, Trans(0,0,0,-90,90,0) is a compound transformation, and base is the robot base frame. We note that the order of these transformations is crucial and positioning errors may occur because of cumulative computational errors.

The *Relative Transformations* are defined as follows. Location 'plate' is defined using teach Pendant that allows moving the robot to location plate and using:

```
HERE plate
```

which defines the position and orientation of the transformation plate, next the tool (robot) is located at object:

```
HERE plate: object
```

which sets the relative transformation object relative to plate, finally:

```
HERE plate: object: grasp
```

Using separate command, assuming plate and object were already defined:

```
MOVE plate: object: grasp !
```

Where "!" indicate external references plate and object.

We now consider the *Computational considerations* of these instructions. Consider the following two,set of instructions:

```
APPRO plate:object:grasp,100.0
MOVE plate:object:grasp
```

Here the robot goes to a location approaching location (plate:object:grasp) by a backward translation of 100 mm with respect to the Z axis.

This set requires computing the transformation twice, therefore, the following sequence requires less computing operations:

```
SET X = plate:object:grasp
APPRO X, 100.0
MOVE X
```

where (plate:object:grasp) will be evaluated only once

6.2.3 VAL II instructions

1. **Format of an instruction**:
   <Step Number> {<label> < space >}{< Instruct >}{<space >}{< comment >} where (step Number) is provided by Val II, and (Instruct) is to be < Instr. name > {<space >< arguments >}.

2. **Assignments**

   <variable> = <value>

   where (value) is real or integer constant or arithmetic expression. $Ex : n = n + 1$, X =radius*SIN(a)

3. **Decomposition**

   DECOMPOSE <array name> { } = <Location>

   where the location consists of $(X, Y, Z, O, A, T) in which (X, Y, Z)$ are the coordinate of the Tool frame origin and angles (O, A, T) are three Euler angles defining the orientation of the Tool frame. The Tool frame is located at the robot hand with its Z axis along the hand, its X axis going to the right of the hand, and its Y axis going down from the hand. Based on the above, angle O is a ROTY for the tool frame, angle A is a ROTX, and angle T is a ROTZ. To manipulate the above data we may use the following example:

DECOMPOSE $X\{\ \}$ = point

where $X\{1, 2, .., 6\} = \{XYZOAT\}$ of location point. Assume (#ref) then (DECOMPOSE X[]= #ref) stores the components into x[] so that to index them. The components can then be used as (MOVE #ppoint(X[0],...,X[5])).

4. **Transformation**

   The syntax is (TRANS(exp1,...,exp6)) like in the following instruction:

   MOVE frame:TRANS(r*cos(a),r*sin(a),0,90,-90,0) or
   SET X = frame:TRANS(r*cos(a),r*sin(a),0,90,-90,0)

   where (frame) is a transformation defining the position of the center of a cercle, (r) is the radius, and (a) id the angle. Another form is the (STRANS <array name> { }) where (array name) is to receive the robot hand data. For example: STRANS $X\{\ \}$ produces the array X such that the first 9 components consists of the orientation matrix and the last three components consists of the cartesian position.

5. **Location variable assignments**

   (a) **HERE < location var>**
   Allows copying the robot coordinate into a variable.
   Ex1: (HERE part) where (part) becomes the current Robot location
   Ex2: (HERE #part) where (#part) becomes the current Robot precision point

   (b) **SET <location var> = <location value>**

   Generally <location var > is a compound $obj(1) : obj(2) : ... : obj(k)$ which should be previously defined. $obj(i)$ is set relative to $obj(i+1)$ using <loaction variable>.

   Ex. SET #ref = #ppoint(exp1, ..., exp6)
   EX. SET picks = HERE: Shift

   sets picks equal to transf. shift relative to current
   robot location (HERE)
   SET # place = # post
   precision point "# place" becomes = that of "# post"

6. **Motion Instructions**

   (a) **Move <location>**
   Moves using joint interpolated motion (J.I.M), in the joint space, to a destination specified as a location or a precision point.

Ex1:	MOVE # picks
	moves J.I.M. to precision point # picks
	MOVE pallet
	moves J.I.M. to location pallet
Ex2:	MOVE #ppoint(exp1,exp2, ...,exp6)

```
Ex3:        Here #ref
            DECOMPOSE X[]= #ref
            MOVE #ppoint(X[0],...,X[5])
```

(b) **MOVET <location>, <hand opening>**

  moves J.I.M. to (location) and hand opening
  EX. MOVET part1, Hand-5
  moves J.I.M. to (part 1), and Hand Hand-5

(c) **MOVES <location>**

  moves on a straight-line path (SLP) to "Location"
  Ex. MOVES place

(d) **MOVEST <location> , <hand opening>**

  moves SLP and hand opening (mm)
  EX. MOVEST part #, 25
  moves SLP and hand 26 mm

(e) **APPRO <location>, <Distance>**

  moves J.I.M. to Location - (Distance)
  Ex. APPRO place, offset
  moves J.I.M. to place, only z component to place
  is affected by (- offset).

APPROS <location>, <Distance>
moves S.L.P. to Location - (Distance)

(f) **DEPART < Distance >**

  moves J.I.M. from current robot location backs
  by (Distance)
  Distance > 0 .... > backs Z axis
  Distance < 0 .... > forward Z axis

(g) **DEPARTS <Distance>**
moves SLP from current robot location backs (Distance)Z.

6.2.4 VAL II structured construct

1. **The case structure** Allows selecting actions depending on external conditions:

```
CASE <exp1> OF

        VALUE <exp> {,...<exp>}: <group of steps>
        VALUE <exp> {,...<exp>}: <group of steps>
        {ANY} <group of steps> END
```

Description:

(a) <exp 1> is evaluated, let Val be its value,

(b) All the <exp> stat. following VALUE are evaluated, if any of these returns a value equal to Val, then the <group of steps> is executed and control is transferred to statement following the END.

(c) If there is no <exp> whose value matches with Val and if an ANY statement is included, then the <group of steps> following ANY is executed and control is transferred to the statement following the END.

Example: CASE INT(X) OF

```
        VALUE 0, 2, 4, 6, 8, 10
        TYPE "The number is even"
        VALUE 1,3,5,7,9
        TYPE "The number is odd"
        ANY
        TYPE "The number is < 0 or > 10"
END
```

2. **DO-UNTIL** DO
{< group of steps >}
UNTIL <logical exp>

   (a) The <group of steps> is executed as long as <logical exp> is false.

   (b) When <log. exp> is true, then control is transferred to the statement following the END.

3. **FOR** <log Var.> = <initial> TO <final> {STEP increm.>}

```
FOR part = 1 TO row.length

    CALL move.part
    SET hole = SHIFT (hole by 100,0,0)
END
```

   This assumes that (move.part) is a subroutine to pitch up a part and it down at the location "hole".

4. **IF** <log. exp> THEN
<group 1 of steps>
{ELSE
<group 2 of steps>}
END

5. **WHILE**
<logical exp> DO
{<group of steps> } END

Exercises

1. Give example of compound transformations and explain their benefits in manipulating an object such as a cup
2. Find the mathematical transformations that are required to perform the following type of motions:
   (a) Joint interpolated motion
   (b) Straight-line path in the cartesian space.
3. A robot is assigned to pick blocks off of a conveyer belt and deposit them in a pallet with three-by-three array of positions for the blocks. Write a VAL II program to perform this operations.
4. A robot gripper is equipped with three proximity sensors: left (SL), right (SR), and hand(SH). All three sensors can be read by VAL II and give binary values.
5. If an object is in front of a sensor, then part of the emitted light is reflected on consequently the value returned is logic 1. Otherwise it returns O. The opening or closing of the robot hand can be programmed in an incremental way. Write a program in VAL II language to perform the following:
   (a) Search on the workspace table (L_1, L_2) an object. For this left and right motions will be required until either SR or SL return a 1.
   (b) Center the robot hand on the object: moves right or left (if $SR = 1$ or $SL = 1$, respectively) until both sensors return 0. Then close the gripper by Δd until either $SR = 1$ or $SL = 1$. If both sensors generate 1 then exit.
   (c) Now the robot hand is centered in front of the object. Moves to the object and check all three sensors. If only Sr or SL return 1, then center again. If only SH returns 1, then grasp the object and exit (success.)