RPM — Laboratory Exercise Report

Lab Exercise LP1: Basic PDL2 Motion Programming
Lab Dates: 26/04/2012

Raffaello Camoriano

1. Objectives of LP1

The goal of LP1 was to help the class learn to program basic robot motions using the PDL2 language and the WinC4G environment.

In particular, the following topics were discussed and practiced:

(1) An Introduction to the C4G IDE.
(2) Programs and routines.
(3) Basic statements, user variables and global variables.
(4) Defining a user frame.
(5) Motions, paths, and nodes

2. Questions

In all questions, the robot programming operations are to be performed using PDL2.

2.1. How to set $UFRAME using 3 points expressed with respect to the base frame? Why is this important?

Answer:

The $UFRAME can be set in the desired position by means of the following sequence of instructions:

```
$UFRAME := POS(0, 0, 0, 0, 0, 0, '')
$UFRAME := POS_FRAME(origin, x, xy)
```

The reason is that the positions of the 3 points which define the new user frame are expressed with respect to the base frame. Thus, it is necessary to first set the user frame as coinciding with the base frame in order to place the new user frame basing the computation on the right points (origin, x, xy), and not on some other points with respect to an arbitrary user frame which might be active at the moment of the execution of the POS_FRAME function.

NOTE: The POS_FRAME function returns a frame basing the computation on the following data:

- Origin: point of origin of the frame
- X: a point along the X axis of the frame
- XY: a point on the XY plane of the frame

These three points are sufficient for defining the new frame.

2.2. What is $ARM_DATA? What is the difference between using the variable $ARM_DATA[1] and the WITH clause?

Answer:

$ARM_DATA is a dynamic array whose elements group the data associated to each arm installed in the robotic system. These elements can change through time, because of possible reconfigurations or modifications of the system (for example, an upgrade for a production cell). Every element has got different fields containing a certain piece of information related to the single arm, which can be accessed and modified by the programmer with an intuitive dot syntax (as for objects in OOP).

Example 1: overriding speed of arm 2 at 50% for motions issued from the calling program

```
$ARM_DATA[2].PROG_SPD_OVR := 50
```

Note: the fields of the $ARM_DATA element related to the arm which is specified in PROG_ARM can be accessed without the prefix $ARM_DATA[x] (see example 2).

Example 2:

```
PROGRAM armspeed PROG_ARM=1
BEGIN
$ARM_DATA[2]. PROG_SPD_OVR := 30 – Override speed of arm 2
$ PROG_SPD_OVR := 50 -- Override speed of arm 1 (no prefix)
END armspeed
```

The difference between using $ARM_DATA and the WITH close lies in the fact that changing an element of a component of $ARM_DATA results in a permanent effect, which lasts for the entire execution of the program (of course, unless the value is changed again) affecting all the motion instructions.

On the other hand, the WITH clause enables a temporary modification of the values of the predefined motion variables reported in the following list, only for the duration of a single instruction.

Example 3: performing a temporary speed override analogous to the one of example 2, only in correspondence with a single MOVE instruction.

```
MOVE TO point WITH $PROG_SPD_OVR = 30
```

Predefined variables which can be temporarily modified using the WITH clause:

$ARM_ACC_OVR	$ARM_DEC_OVR	$ARM_LINKED
$ARM_SENSITIVITY	$ARM_SPD_OVR	$AUX_OFST
$BASE	$CNFG_CARE	$COLL_SOFT_PER
$COLL_TYPE	$FLY_DIST	$FLY_PER
$FLY_TRAJ	$FLY_TYPE	$JNT_MTURN
$JNT_OVR	$LIN_SPD	$MOVE_TYPE
$ORNT_TYPE	$PAR	$PROG_ACC_OVR
$PROG_DEC_OVR	$PROG_SPD_OVR	$ROT_SPD
$SENSOR_ENBL	$SENSOR_TIME	$SENSOR_TYPE
$SFRAME	$SING_CARE	$SPD_OPT
$STRESS_PER	$TERM_TYPE	TOL_COARSE
$TOL_FINE	$TOOL	$TOOL_CNTR
$TOOL_FRICTION	$TOOL_INERTIA	$TOOL_MASS
$TOOL_RMT	$TURN_CARE	$UFRAME
$WEAVE_NUM	$WEAVE_TYPE	$WV_AMP_PER

2.3. What is the difference between: MOVE JOINT TO POS(0,0,0,0,0,0,'') and MOVE JOINT TO JNT(0,0,0,0,0,0)? Why is it important to use MOVE JOINT TO as a first instruction in a path?

Answer:

MOVE JOINT TO POS(0,0,0,0,0,0,'') performs the motion of the arm to a target position with respect to the active reference frame. This target POSITION variable is returned by function POS( ) on the basis of the function parameters, which are:

- Cartesian coordinates X, Y, Z
- Euler angles Z, Y, Z
- Configuration string (optional): it specifies the configuration related to the position, solving possible ambiguities.

MOVE JOINT TO JNT(0,0,0,0,0,0) moves the arm to the joint position (JOINTPOS) specified by the arguments of function JNT( ). In fact, each of these arguments corresponds to a joint parameter (an angle in degrees). Thus, this motion does not depend on the active user frame, because the joints move independently of each other, only on the basis of their target values.

The first MOVE instruction in a path should always include the trajectory clause JOINT, because the arm could initially be in any configuration, and reaching the first point in the path using any other trajectory clause may result in the arm passing through singular configurations or configurations at full stroke. In this mode, the joint angles are linearly interpolated between their initial and final values. Therefore, the movement is always feasible. The movements of the single joints begin and end at the same instants, while the movement of the TCP is not linear or circular and hence it is very difficult to predict a-priori.

2.4. What are the following variables used for? $ARM_SPD_OVR, $MOVE_TYPE, $FLY_TYPE? How can you get the TCP to move with a constant velocity? Is it always possible?

Answer:

1) **$ARM_SPD_OVR (Arm speed override):** it is a field of $ARM_DATA[i], defining the speed override percentage related to the motions specified for an arm. This percentage overrides the $GEN_OVR speed percentage, which is valid in the whole system. Overriding the speed does not affect acceleration and deceleration, but it may affect the trajectory.
2) **$MOVE_TYPE:** it defines the type of interpolation used for computing the trajectory to be followed during a motion statement.
   For standard motions, its values can be the following predefined constants:

   - LINEAR
   - CIRCULAR (note: needs a VIA position)
   - JOINT

   In paths, $MOVE_TYPE can also be defined as SEG_VIA for the node included in a circular motion.

3) **$FLY_TYPE:** it specifies the type of Cartesian fly motions:

- **FLY_CART** (Controller Aided Resolved Trajectory): the speed of the TCP is kept constant, while the arm stress constraints are respected. If these constraints are not respected, then the trajectory is changed and/or the speed is reduced. The stress threshold is defined by $STRESS_PER (stress percentage in Cartesian fly), whose default value is 50%. FLY_CART only works for Cartesian motions, and $SPD_OPT must be set to SPD_LIN.

- **FLY_NORM**: the speed of the TCP is not kept constant. This is the default value for $FLY_TYPE, and allows us to specify how much the deceleration and the acceleration before and after a position should overlap, by specifying a percentage in the variable $FLY_PER. Hypothetically, though, setting $FLY_PER = 0 would degenerate the MOVEFLY to a MOVE statement.

We can force the TCP to move with constant velocity by setting:

```
$FLY_TYPE := FLY_CART
```

This is also possible for a node belonging to a path, for instance:

```
pth_laptop.NODE[5].$SEG_FLY_TYPE := FLY_CART
```

Once we have set FLY_CART, it is possible to vary the stress percentage of the arm which must be respected (e.g. 30%):

```
$STRESS_PER := 30
```

NOTE: $SEG_STRESS_PER has the same meaning of $STRESS_PER, but it is valid for paths.

As already introduced, it is not always possible to impose a constant speed for the TCP, because if the stress constraint is not respected the speed may be reduced by the system.

2.5. How can you make the TCP move on a circle while the end-effector moves from the current position to a target position, *P*? What additional data (apart from *P*) needs to be provided in the command and in what form? How do we define a circular segment inside a path?

Answer:

It is possible to set the $MOVE_TYPE to CIRCULAR, in order to make the TCP move on a circle from the initial point to the target. It is mandatory to specify a point through which the arc must pass, in order to determine it univocally. This can be done using the VIA clause of the MOVE statement. These three points (initial, via and target) are sufficient for univocally computing the arc trajectory.

```
MOVE CIRCULAR TO target VIA arc_point
```

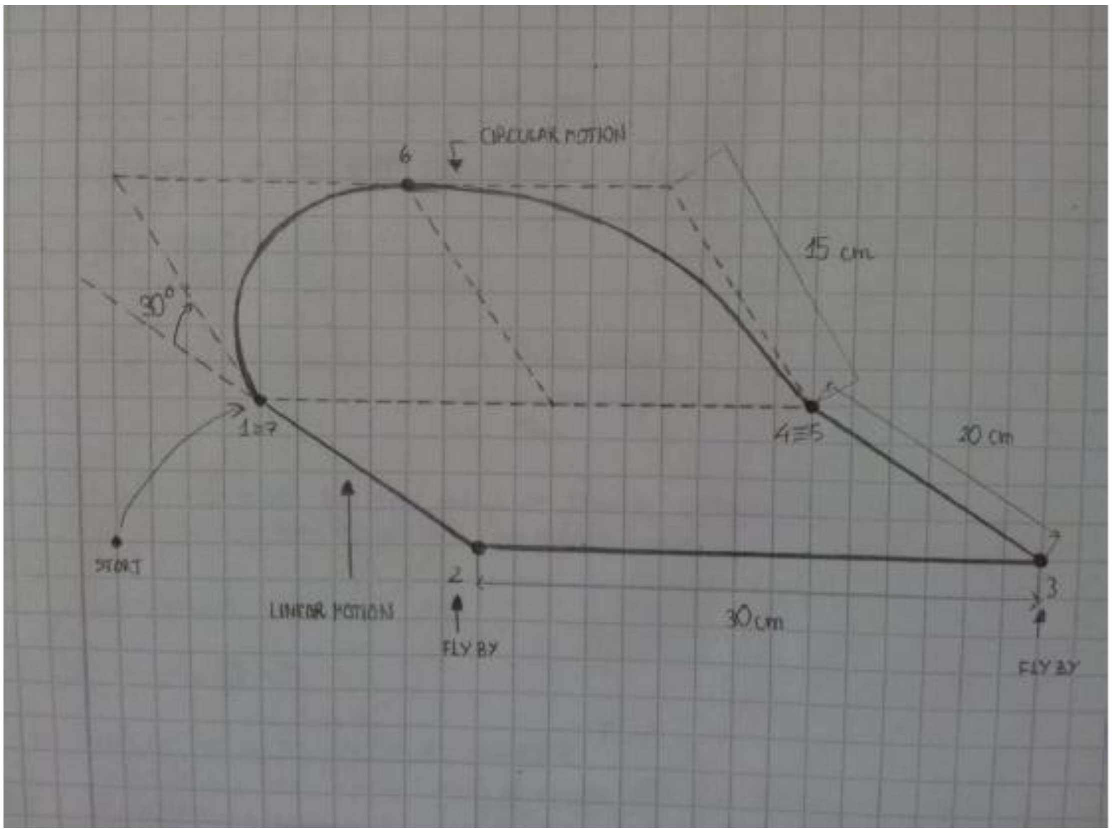
A circular motion can also be specified inside a path, by defining the target node's $MOVE_TYPE as CIRCULAR and the previous node belonging to the arc as SEG_VIA.

Example: circular motion defined inside a path:

```
…
pth_laptop.NODE[6].$MOVE_TYPE := SEG_VIA
pth_laptop.NODE[7].$MOVE_TYPE := CIRCULAR
…
```

3. Assigned program and verification

3.1. *Assignment*. Write a program in which the following end-effector motion is defined as a PATH and executed. First, the end-effector moves above the three sides 12, 23, and 34, of a horizontal (in the user frame) rectangular contour. The TCP remains at a constant distance, *d*, from the plan, while the TOOL frame *z* axis is directed normally towards the plane. Then, the end-effector moves above the 180-degree arc 567 which lies in an inclined plane, as indicated in the figure below. Again, the TCP remains at a constant distance, *d*, from the plan, while the TOOL frame *z* axis is directed normally towards the plane. The TCP must move with a constant velocity. The dimensions are given in the figure.



Commented program code:

```
-- PATH LAPTOP

PROGRAM path_laptop PROG_ARM = 1

   --NODE DEFINITION
TYPE
   node1 = NODEDEF
      -- Type of positional data of the node
      $MAIN_POS      -- Cartesian position

      -- Other variables that define the motion behaviour for each segment.
      $MOVE_TYPE      -- Movement type
      $LIN_SPD      -- Linear speed
      $SEG_REF_IDX      -- Frame of reference
      $SEG_FLY
      $SEG_FLY_TYPE    -- Type of Cartesian fly motion.
   ENDNODEDEF

VAR
   corner, x, xy : POSITION
   pth_laptop : PATH OF node1
   -- This routine deallocates the memory of a path.

ROUTINE reset_path(pth : PATH OF node1)
BEGIN
   -- It is checked wether the path was actually allocated first, to avoid runtime errors.
   IF PATH_LEN(pth) > 0 THEN
      NODE_DEL(pth, 1, PATH_LEN(pth))
   ENDIF
END reset_path

   -- Initializes all nodes of a path at position 0, movement type specified, linear speed
specified
   -- The move type of the first node will always be set to JOINT, to be sure the robot
will reach it.

ROUTINE init_path(pth : PATH OF node1; move_type : INTEGER; lin_spd : REAL;
frame_num : INTEGER)
VAR
   i : INTEGER
BEGIN
   IF PATH_LEN(pth) > 0 THEN
      FOR i := 1 TO PATH_LEN(pth) DO
         pth.NODE[i].$MAIN_POS := POS(0, 0, 0, 0, 0, 0, '')
         pth.NODE[i].$SEG_REF_IDX := frame_num
         IF i = 1 THEN
            pth.NODE[i].$MOVE_TYPE := JOINT
         ELSE
            pth.NODE[i].$MOVE_TYPE := move_type
         ENDIF
         pth.NODE[i].$LIN_SPD := lin_spd
      ENDFOR
   ENDIF
END init_path



   -- Main program
BEGIN
   -- Set reference User Frame.

   $UFRAME := POS(0, 0, 0, 0, 0, 0, '')
   $UFRAME := POS_FRAME(corner, x, xy)
```

```
  -- Set GLOBAL motion variables.
  $SPD_OPT := SPD_LIN      -- Tells to use $LIN_SPD to determine motion speed.
  $MOVE_TYPE := JOINT      -- JOINT motion type.
  $FLY_TYPE := FLY_NORM -- Constant velocity of the EE during MOVEFLY is
the default

  -- Deallocate path nodes

  reset_path(pth_laptop)

  -- Initialize auxiliary frame

  pth_laptop.FRM_TBL[1] := POS(0, 300, -30, 0, 30, 0, '')

  -- Allocate path nodes.

  NODE_APP(pth_laptop, 8)

  -- Initialize path

  init_path(pth_laptop, LINEAR, 0.2, 0)

  -- Set nodes 2 and 3 belonging to the rectangular path to be passed through with
constant velocity, using FLY_CART

  pth_laptop.NODE[2].$SEG_FLY_TYPE := FLY_CART
  pth_laptop.NODE[3].$SEG_FLY_TYPE := FLY_CART

  -- Set nodes of the circular path to work with respect to the auxiliary frame

  pth_laptop.NODE[5].$SEG_REF_IDX := 1
  pth_laptop.NODE[6].$SEG_REF_IDX := 1
  pth_laptop.NODE[7].$SEG_REF_IDX := 1


  -- Set the target node of the circular motion
  pth_laptop.NODE[7].$MOVE_TYPE := CIRCULAR

  -- Node 6 must be defined as "via" node
  pth_laptop.NODE[6].$MOVE_TYPE := SEG_VIA
```

```
    -- Avoid MOVEFLY between the rectangular and the circular sub-paths.
    pth_laptop.NODE[4].$SEG_FLY := FALSE
    pth_laptop.NODE[5].$SEG_FLY := FALSE
    pth_laptop.NODE[8].$SEG_FLY := FALSE

    ---- PATH NODES DEFINITON

    ---- Reference frame of the following nodes: user frame

    -- Sides of the keyboard
    pth_laptop.NODE[1].$MAIN_POS := POS(0, 0, -30, 0, 0, 0, '')
    pth_laptop.NODE[2].$MAIN_POS := POS(-200, 0, -30, 0, 0, 0, '')
    pth_laptop.NODE[3].$MAIN_POS := POS(-200, 300, -30, 0, 0, 0, '')
    pth_laptop.NODE[4].$MAIN_POS := POS(0, 300, -30, 0, 0, 0, '')

    ---- Reference frame of the following nodes: aux frame

    -- Circular path
    pth_laptop.NODE[5].$MAIN_POS := POS(0, 0, -30, 0, 0, 0, '')
    pth_laptop.NODE[6].$MAIN_POS := POS(150, -150, -30, 0, 0, 0, '')
    pth_laptop.NODE[7].$MAIN_POS := POS(0, -300, -30, 0, 0, 0, '')

    ---- Reference frame of the following nodes: user frame

    -- Final position
    pth_laptop.NODE[8].$MAIN_POS := POS(0, 0, -30, 0, 0, 0, '')

    ---- END OF PATH NODES DEFINIITON

    -- Move along path

    MOVE ALONG pth_laptop

    -- Deallocate path nodes

    reset_path(pth_laptop)

END path_laptop
```

3.2. *Corrections and comments (optional)*

Here you can describe what you learned during the correction and verification. E.g., did you correct any interesting errors or clear out any misunderstandings about how the language works? Can these be avoided in the future by introducing the material differently? Described errors will not affect negatively the evaluation of this report. These comments can only improve your mark for the report.

During the debug phase, I have discovered that all the extra information which is needed for defining a particular path (for example, the auxiliary reference frame index $SEG_REF_IDX, related to the frames contained in FRM_TBL) must be declared in the node definition.

However, the program worked correctly at the first execution, so there are no further issues to report.

4. Conclusions and recommendations (optional)

Briefly comment on the lab. Did you find it useful? What changes and improvements would you recommend for the future?

This lab session has been very useful for me, because it included the first execution of a simple program of mine on a real robot. I believe that all the students should pass as much time as possible working directly with the robot, this approach works and it is very stimulating.

Main references: The Comau manuals “PDL2 Programming Language Manual” and “C4G Motion Programming”