RPM — Course Development Project Report

Task P3 : PDL2 Programming of Pick-and-Place Tasks
Task Completion Date : 23/05/2012

Raffaello Camoriano, Yue Hu, Michele Labolani

1. Objectives

The project consisted in the preparation of a laboratory exercise, LP3, in the Robotics Programming Methods course in the EMARO/Robotics Engineering Program. The goal of LP3 was to help the class learn to write and execute PDL2 programs for pick-and-place handling tasks. Additionally, the use of priorities for program scheduling was explored.

In particular, the lab was intended to have these learning outcomes:

1. How program priorities work and how they interact with motion statements
2. Defining auxiliary frames using saved positions
3. Using the HAND statement to control the gripper
4. Safety measures to be respected during the pick-and-place operations
5. Use of MOVE NEAR and MOVE AWAY
6. Collision detection statements

2. Project work: Lab Preparation

In this section describe factually your preparation. Which manuals or other resources did you use? How many hours (approximately) did you work and experiment with the robot? How many hours it took to read and prepare? How long it took from the beginning of work till the completion of the preparation for the lab? How long was the prepared laboratory exercise itself?

Manuals and other resources:

- Comau PDL2 manual
- Comau C4G manual
- Comau motion manual
- Comau “Control Unit use” manual
- Comau EZ PDL2 manual
- WinC4G “causa/rimedio” debugging repository
- PDL2 Programming Language for Comau Robots - Presentation by Elena Grassi

Duration of the Lab activity

Hours of work and experiments on the robot:

Individual (P3 members): ~ 20h
Total (P3 members): ~ 60h

Hours of reading and preparation:

Individual (P3 members): ~ 10h
Total (P3 members): ~ 30h

Total days for preparing the lab (P3 members): 14

Duration of the prepared lab:

Independent work by colleagues: ~ 5h (manuals survey and coding)
Interactive work: ~ 5h (TD presentation, individual explanations, code revision and online tests)

3. Project results: Description of LP3

This is the main part of this report. Use several pages to describe the PDL2 and material that the lab was intended to teach. Here, you describe what you learned and what you decided had to be taught to all. You can include pictures and text from the manuals. You will divide this section into subsections (and in further sub-subsections) as appropriate.

1. LP3 Organization

The lab had two parts, … *Describe briefly and factually how the lab was organized, i.e., who did what when. The substantive content of what was done describe in more details in the following two subsections.*

The lab has been divided in two main parts, one focused on the pick and place task and a second one on programs scheduling in PDL2 with priorities.
To prepare the pick and place program we had to wait until the gripper was mounted and tested correctly by R3, so we started first to work on the priorities in PDL2.
Then we proceeded with a short program to test how the gripper works in PDL2. After this, the position of every piece to assemble and the car floor has been recorded with respect to specific user frames previously defined.
The final phase was to merge the priorities with the pick and place, in order to have a lower priority program that performed some motion while waiting for another car to be assembled with a higher priority program, but this phase failed due to misunderstandings about how PDL2 priorities work.

2. LP3 Presentation

2.1 General structure of the program, priorities and task scheduling

The draft source code of the two programs performing the assembly and waiting tasks has been commented statement-by-statement, marking which new instructions were used (priorities, collision detection, NEAR, AWAY).

In particular, the difference between the semaphore-based synchronization used in LP2 and a possible priority-based synchronization has been shown. The use of priorities has been studied, revising the key concepts of the Round-Robin scheduling algorithm (shown in Figure 1) and the information found in the Comau PDL2 manual.

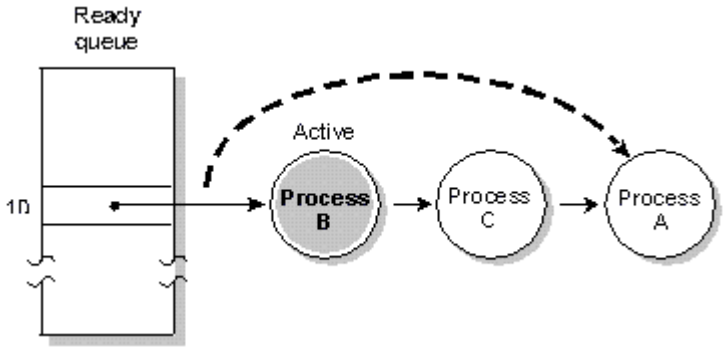


Figure 1: The Round-Robin scheduling algorithm

From Comau PDL2 Manual:

> Program scheduling is done on a priority basis. The programmer can set a program priority using the PRIORITY attribute on the PROGRAM statement.
>
> PROGRAM test PRIORITY=1
>
> Valid priority values range from 1-3, with 3 being the highest priority and 2 being the default. If two or more programs have equal priority, scheduling is done on a round robin basis, based on executing a particular number of statements.
>
> Programs that are suspended, for example those waiting on a READ or SEMAPHORE or paused programs, are not included in the scheduling. Interrupt service routines use the same arm, priority, stack, program-specific predefined variable values, and trapped errors as the program which they are interrupting.

Note:

Before the TD presentation to the class, the priority-based synchronization of two programs performing motions was tested only for single motion statements.

Two programs were created: *assemble* and *emergency*. The *assemble* program has low priority (set to 1) and performs two motions which consist in drawing a line on a

plane, while *emergency* has a higher priority (set to 2) and performs only one motion which is to move to a different position (in joint mode).

In order to make the higher priority program not to be scheduled immediately we used a semaphore. *Emergency* waits for the semaphore when it is launched, while *assemble* signals to make *emergency* scheduled when a certain condition occurs.
We set this condition as when the start button is pressed:

```
CONDITION[1] NODISABLE :
        WHEN START DO
        SIGNAL sem1
ENDCONDITION
```

During further online testing phases, involving more complex motions with more than a single MOVE statement in the higher priority program (e.g. the pick and place program), it has been observed that when the high-priority program performs a motion statement it is then suspended until the end of the movement (motion statements are blocking), allowing the following motion statement of the lower priority program to be scheduled right after it. Many different approaches have been tried for solving this problem, like using ATTACH and DETACH, LOCK and UNLOCK statements, but these did not come useful.

The sequence of prior motions is broken up by the lower-priority ones, making it impossible to synchronize the motions of the robot by just using priorities and without disabling interrupts (which is always a dangerous choice for real-time safety-critical systems).

2.2 The NEAR and AWAY destination clauses in MOVE statements

The NEAR destination clause allows the programmer to specify a destination along the tool approach vector that is within a specified distance from a position. The distance, specified as a real expression, is measured in millimeters along the negative tool approach vector relative to the final position (which is, as a default, along the Z axis of the tool frame).

The destination can be any expression resulting in one of the following types:

- POSITION
- JOINTPOS
- XTNDPOS

For example:

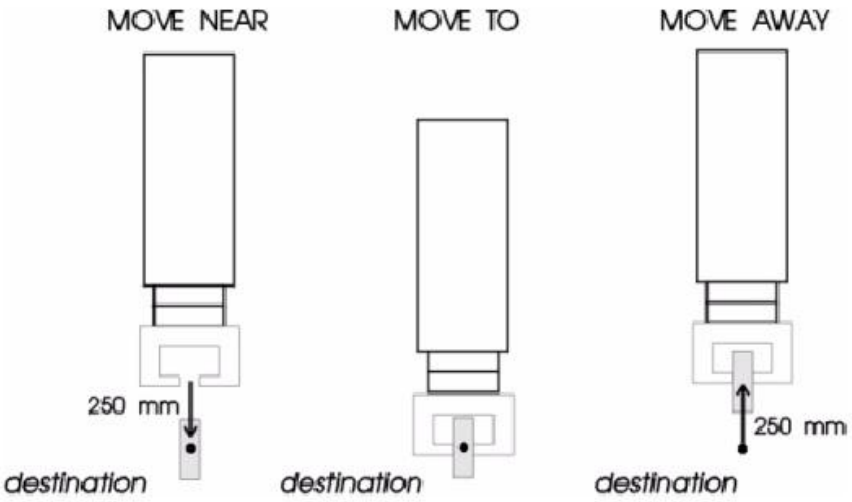
```
MOVE NEAR destination BY 250.0
```

In the same way, the AWAY destination clause allows the programmer to specify a destination along the tool approach vector that is a specified distance away from the current position.

For example:

MOVE AWAY 250.0

The picture below shows how these clauses work.



2.3 Tool definition

In order to be able to properly use a tool mounted on the robot flange, it is necessary to physically define it.
This could be done by directly setting its geometry in the $TOOL variable, which represents how the TCP should be translated and oriented w.r.t. its previous position.
Anyway, in order to achieve a better result, it would be necessary to use the "*automatic tool definition"* (TO_SET) built-in feature.

2.4 Collision detection (optional feature)

The collision detection algorithm gives the system the ability of locking the arm motion as soon as some external force acts on any joint. This is done by continuously monitoring the amount of requested current, compared with presupposed values. This feature can simply be enabled by the user by an apposite variable setting. It is also possible to enable the so called "robot yielding", which would lead, in case of collision, the robot to distribute the energy coming from the impact along all its axis, in order to better absorb it.

Note: collision detection is not supposed to guarantee user safety, but only to preserve from damages the robot structure and the environment it is installed in.

The algorithm is mainly based on a few settings the user can define:

- *Collision detection enabling/disabling:* to activate the feature, it is only necessary to set the system variable $CRNT_DATA[num_arm].COLL_ENBL to TRUE (to deactivate, just set it to FALSE).

- *Robot yielding activation:* there is no predefined variable dedicated to it: it is necessary to directly set to 1 (or to 0) an apposite bit by using BIT_SET($ARM_DATA[num_arm].A_ALONG_1D[12], 11)
  (or BIT_CLEAR($ARM_DATA[num_arm].A_ALONG_1D[12], 11)).
- *Collision error severity*: first, it is possible to define how the robot would react to a collision detection: the set of possible actions is made of 3 choices: a DRIVE OFF, a HOLD or a simple event rising (error code: 197).To do this, the value of the system variable $COLL_EFFECT has to be set respectively to 0, 1 or 2.
- *Collision sensitivity:* robot sensitivity to collision could be defined using the variable $COLL_TYPE. This is strictly dependent on the application the robot will be used for, specially if the tool and the load have not been properly defined (see "*automatic tool definition"* and "*automatic load determination*" built-in procedures on the Comau manuals). The set of values this variable can assume are mainly COLL_LOW, COLL_MEDIUM, COLL_HIGH and COLL_MANUAL (it is also possible to define up to 10 user defined thresholds.)

3. ***LP2 assignment***

3.1 **Assignment and solution**

Once the class has been made aware of what the theoretical content of the lab would have been, and of how it could have been implemented in PDL2, the weekly homework has been assigned.
Our initial proposal would have suggested them to write 2 concurrent programs, the former dedicated to a pick and place operation, the latter to a drawing action. The first scenario premised the two programs to have different priorities: the drawing program would have been assigned to the lower one, while the pick and place to the higher. Such a structure would have allowed, as far has we had thought, to switch from the lower to the higher priority one simply by making it be scheduled.

3.1.1 *The pick and place program*

The structure of this program would have been straightforward: suppose on the lab floor three metal pieces are laying. These pieces have to be mounted on a car floor, situated on the lab floor too. With the aid of the above mentioned magnetic gripper, the robot would have to pick each of the pieces and mount it in its target position on the car floor (to control gripper operations, PDL2 gives at the user disposal to simple statements, CLOSE HAND *num_hand* and OPEN HAND *num_hand*, which respectively make it takes or releases an object).

A *.var* file containing all the needed positions would have been sent by e-mail to the whole class, since some problems with frames and position recording still had to be overcome. As we have previously said, an higher priority (e.g 2, the default one) would have been given to this program; so, in order not to make it be scheduled, a semaphore would have been required, on which it should have waited until a certain condition would have been satisfied.

The two sets of positions (one for the starting ones, one for the target) were supposed to be stored according to two different user frames, so a frame switching would have been required for the program to correctly work. Both user frames have been defined by the statement $UFRAME = POS(x,y,z,a,e,r,'') instead of POS_FRAME(o,x,xy); in this way the user frame is positioned with the origin in the desired position and the orientation equal to the one of the base frame. It would have been necessary also to set the tool frame, according to the tool R3 mounted on the top of the robot flange. This operation allows to move the TCP from its actual position to a new one.

Furthermore, in approaching and leaving each position, the MOVE NEAR and MOVE AWAY statements would have been required.
Finally, two "via point" would have been included in the .var file, one for each set of positions, just for sake of safeness. In any case, the robot motion velocity would have been necessarily low, in order to avoid any accident.
Once completed its task, the program was supposed to automatically terminate.

Note: The collision detection feature was supposed to be inserted in the program, in order to guarantee a higher safety level to the robot's physical structure. Unfortunately, the optional feature appeared not to be installed in the university C4G controller, so we had to deactivate it (as we discovered, if the collision detection feature has been activated for a robot which does not implement it, it turns it drive off in order not to risk any incident).

3.1.2 *The drawing program*

On the other hand, a lower priority program would have taken the robot from the beginning. It would have been supposed to simply draw a generic path w.r.t. a defined user frame, and to be able to release, under a specific condition, a semaphore, unlocking the other program.

Anyway, as for what came out from what we learned by testing priorities, the plan we made (had been (was???)) changed into new ones: the class would have been requested only to write down a pick and place program, without any priority issue. Then, as an optional assignment, two further programs could have been implemented, in order to show how PDL2 priority scheduling works.

3.1.3 ***Program code example***

Here, a commented possible solution of the programs is proposed:

Pick and place program example

```
PROGRAM newassemble

VAR
  PIECES : ARRAY[3] OF POSITION
  TARGET : ARRAY[3] OF POSITION
  OBJ1, OBJ2, OBJ3, FRAME_OBJ, VIA_OBJ : POSITION
  TAR1, TAR2, TAR3, VIA_TAR, FRAME_TAR : POSITION
  COUNTER : INTEGER
```

```
BEGIN

  -- Collision detection setting. Nice optional feature useful in case of collision possibility.
  -- Our Comau system does not contain the required drivers unfortunately

   --$CRNT_DATA[1].COLL_ENBL:=TRUE      --collision detection enabling
   --$COLL_EFFECT:=0       --set drive off when a collision has been detected
   --BIT_SET($ARM_DATA[1].A_ALONG_1D[12],11)    --enable arm yielding after collision
   --$COLL_TYPE:=COLL_MEDIUM    --apply a medium sensitivity to collision detection

 -- Set reference User Frame.
 -- $UFRAME is the position of the user frame relative to the world.
 -- We first set it corresponding to the world frame.
 -- Then we used coordinates obtained online with the user frame corresponding to the world frame.


 $UFRAME := POS(0, 0, 0, 0, 0, 0, '')

 -- We have to set the tool parameter in order to move the TCP in the center of it.
 -- In the case of our magnetic gripper, becaus of its being perfectly symmetric wrt
 -- the Z axis of the flange, we have only had to set the Z offset

 $TOOL := POS(0, 0, 0, 0, 0, 0, '')
 $TOOL := POS(0, 0, 102.2, 0, 0, 0, '')

 -- Set GLOBAL motion variables, in order to have the desired default values.
 $SPD_OPT := SPD_LIN   -- Tells to use $LIN_SPD to determine motion speed.
 $MOVE_TYPE := JOINT   -- JOINT motion type.

 -- Store all pieces and target positions into 2 different apposite arrays,
 -- for an easier indexing in loop environment
 PIECES[1] := OBJ1
 PIECES[2] := OBJ2
 PIECES[3] := OBJ3

 TARGET[1]:=TAR1
 TARGET[2]:=TAR2
 TARGET[3]:=TAR3

 -- In the following while loop, the pick&place operation is performed:
 -- The robot moves once at a time the 3 pieces from their actual position
 -- to the respective target one.
 -- In doing it, it has to follow a precise "path": between each initial and
 -- final position it has infact to pass through to via-points, each one directly
 -- above one of the two sets of positions. This is only a safety reason, in order
 -- not to make the arm accidentally hurt anything.
 -- in order to have a precise picking up and positioning of the pieces also move near
 -- and move away has been used. These types of motion make the TCP approach the position
 -- (or leave it) by a defined offset in a direction specified by the "approach vector"
 -- of the very position. The necessity of modifying didn't arise, since it is as a default
 -- along the Z axis of the tool.
 -- The program is supposed to end definitely after its first execution.

          COUNTER:=0
          WHILE COUNTER<3 DO
            COUNTER:=COUNTER + 1
            $UFRAME:=FRAME_OBJ
            MOVE TO VIA_OBJ
      $PROG_SPD_OVR:=15
            MOVE NEAR PIECES[COUNTER] BY 50
            MOVE TO PIECES[COUNTER]
            CLOSE HAND 1
            MOVE AWAY 50
            MOVE TO VIA_OBJ
            $UFRAME:=FRAME_TAR
            MOVE TO VIA_TAR
            MOVE NEAR TARGET[COUNTER] BY 50
```

```
    MOVE TO TARGET[COUNTER]
    OPEN HAND 1
    MOVE AWAY 50
    $PROG_SPD_OVR:=50
    MOVE TO VIA_TAR
   ENDWHILE

END newassemble
```

Prioritized programs : low priority one

```
PROGRAM assemble DETACH, PRIORITY = 1

VAR
 corner, x, xy : POSITION
 sem1 : SEMAPHORE EXPORTED FROM assemble NOSAVE GLOBAL
BEGIN
 -- Set reference User Frame.
 -- $UFRAME is the position of the user frame relative to the world.
 -- We first set it corresponding to the world frame.
 -- Then we used coordinates obtained online with the user frame corresponding to the world frame.
 -- The values of corner, x and xy are saved in the .var file.
 $UFRAME := POS(0, 0, 0, 0, 0, 0, '')
 $UFRAME := POS_FRAME(corner, x, xy)

 -- The condition is verified after 2000ms from the beginning of the program
 CONDITION[1] NODISABLE :
  WHEN START DO
   SIGNAL sem1
 ENDCONDITION

 ENABLE CONDITION[1]
 -- Set GLOBAL motion variables.
 $SPD_OPT := SPD_LIN   -- Tells to use $LIN_SPD to determine motion speed.
 $MOVE_TYPE := JOINT   -- JOINT motion type.

 WHILE TRUE DO
  MOVE JOINT TO POS(0, 0, 0, 0, 0, 0, '')
  MOVE JOINT TO POS(-100, 0, 0, 0, 0, 0, '')
  MOVE JOINT TO POS(-100, -100, 0, 0, 0, 0, '')
  MOVE JOINT TO POS(0, -100, 0, 0, 0, 0, '')
 ENDWHILE


END assemble
```

Prioritized programs : high priority one

```
PROGRAM emergency DETACH, PRIORITY = 2
-- this program simply has to take the arm and perform a single motion

VAR
 corner, x, xy : POSITION
 sem1 : SEMAPHORE EXPORTED FROM assemble NOSAVE GLOBAL

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

 WHILE TRUE DO
  WAIT sem1
  MOVE JOINT TO POS(0, 0, -300, 0, 0, 0, '')
 ENDWHILE
END emergency
```

3.2 **Correction and discussion**

Once all the class has tried to complete the proposed assignment, a room was reserved in order to make it possible for us to check their programs before the online final test.
They appeared to have mostly understood the main issues, so the only necessary thing was to explain them how they would have to modify their programs from the initial version (the prioritized one) to the final one. That was necessary because we discovered the problem the very evening before the due date, so not all of them were conscious of the changes.

The programs were correct, at least for the most, even if someone successfully tried to modify the structure from the one we had proposed during the previous lecture.

Last, but not at least, we informed our colleagues about the best practice for motion synchronization between programs, advising them to use semaphores (already studied and experimented in LP2) for obtaining a deterministic and predictable sequence of motions. This suggestion has also been confirmed directly by Comau engineers during our visit in Grugliasco, who specified that priorities are to be used for scheduling programs not involved in motion (e.g. Graphic User Interface updates or logging).

3.3 **Program testing on the robot**

For the final online test, we decided to run each program only as a simulation first, and to test them with the pieces on the floor only once the first test would have resulted positive.
Most of the programs worked at the very first try, and only a few among the whole set needed to be overhauled again from one of us.
We can state we were completely content of the class results.

3.4 **Alternative and additional tasks**

As an additional task, an “artificial scheduling” could be implemented with the aid of semaphores, as P2 showed in its laboratory. This could just be a possible solution to replace what was our intent with prioritized programs.
Then, an higher use of events could be made, and an apposite program for their management could be implemented.

4. ***Conclusions***

This lab has been particularly successful in its pick and place part, because in this task it is sufficient to pay a particular attention in the position recording, the frames positioning and the trajectory motion definition.
At the beginning we found it very difficult to define a user frame on the teaching pendant and use it in a program, because the TP doesn’t permit users to access the frame variables recorded in it except the origin. Also the vice versa has been difficult because the TP IDE doesn’t open programs created with the Win4G IDE on the PC, so it is not possible to define a user frame in a program and try to record positions using this program on the TP.

The solution we found is to define the user frames on the TP oriented as the base frame and use their origin to define them in the program. In PDL2, if $UFRAME = POS(x,y,z,a,e,r) is used, the user frame is defined with the origin in this position and oriented as the base frame.
In this way we were able to record every position with respect to the proper user frames.

The priority part didn't work as we expected, thus it resulted unusable for our purpose. It is recommended to not use it to schedule programs where more than one performs motions, since the priority is used to schedule other kind of programs. This is why we weren't able to create a program that performs some motions while waiting another car to arrive.

It is also recommended to not use collision detection in future programs, because this feature is not implemented in the software version of the COMAU in the lab. In fact, if it is used, it blocks the arm until the feature is disabled, and the deactivation of the collision detection has to be done with another PDL2 program with the proper statements.