ARPRO-IRP – Lab n° 1

In this document, identifiers in **bold** are Val/V+ keywords. Identifiers in *italics* are predefined location names stored in the file mentioned at the beginning of each exercise. Loading the file causes the identifiers to be known to the interpreter. Use them in your programs.
For binary inputs/outputs numbers, see lab documentation.

Exercises 1 to 3 have a "Before programming" section. You will solve all the "Before programming" sections first. Before testing your program, you must:

- **Write down your answers to the "Before programming" questions in a word processor file or text file.**
- **Copy your code to the file.**
- **Submit as a unique file by sending the file by email to gaetan.garcia@ec-nantes.fr**

Only after you have done this, you are allowed to check your program on the robot.
One your program works, re-submit the code if it has been modified with respect to the initial version.

Exercise 1:

Files to load for this exercise: « pickconv.lc » and « alltools.v2 » .
Tool to use: succion pad.
Tool transformation name: succion.pad.
Initialisation: set output 5 (opens valve for air pressure).
No speed instructions, except in part 2 of the exercise.

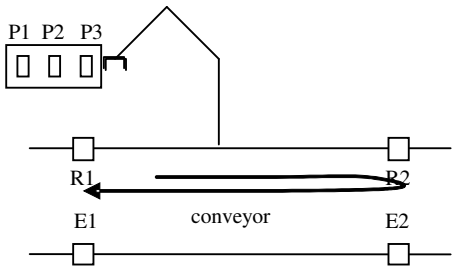


Diagram of the situation

- ❑ The operator places a part on *P1* or *P2*, both connected to binary inputs of the controller.
- ❑ The robot grasps the part at the corresponding location with the suction pad (instructions **closei**/ **openi**).
- ❑ The part is placed at the *place* location on the conveyor, between detectors 1 and 2.
- ❑ The conveyor is started towards detector 2 and stopped when the object is at detector 2.
- ❑ It is restarted in reverse direction and stopped when the part crosses detector 1.
- ❑ The part is grasped (location *grasp.loc.stopped*) and placed at *P3* (figure)

Robot initial and final locations: *wait.loc*.

Introduce a delay between the stop of the conveyor and the moment it is restarted in the opposite direction (Use the **timer** instruction, see below).

Next, create a second version of the program, in which you will try to grasp the part without stopping the conveyor. Calculate the conveyor speed by measuring the time elapsed between the two detectors (see documentation for detector spacing). Let the part pass detector 2 before changing conveyor direction. The robot will wait for the part 150 mm above the previous grasp location *grasp.loc.stopped* and will grasp the part 150 mm farther along the conveyor, using a straight line motion, as indicated in the next figure.

The conveyor is aligned with the robot's Y axis (see drawing in the lab documentation).

Before programming:

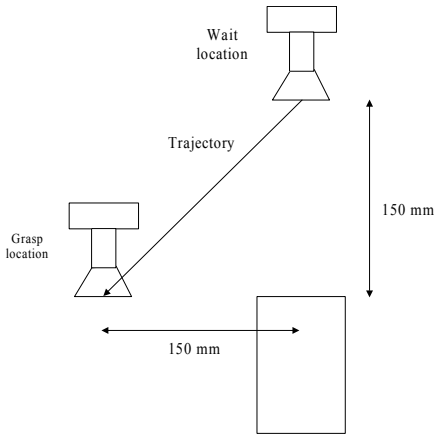
- List the inputs used in the exercise.
- For each of them, explain if you will handle it synchronously or asynchronously. Briefly justify your answer. If asynchronously, justify the choice between react and reacti instructions.

Using timers:

Timers are identified by a number (1 to 15). All times are in seconds.

Setting a timer: **timer** <timer_number> = <value>

Reading a timer: <variable > = **timer**(<timer_number>)



For this second part, you will also answer the "Before programming" questions and submit with the code before testing your program. Unless something has changed in the handling of the inputs, there is no need to discuss the handling of the inputs again.

For the second program of this exercise, insert the following instructions at the beginning of the program:

speed 30 **always** ;
speed 100 **mmps always** ;

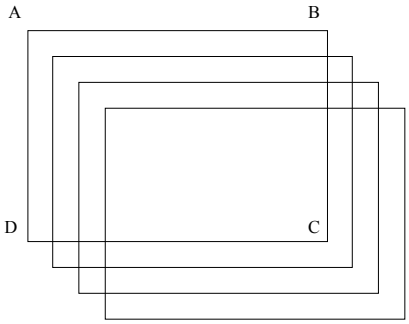
Answer the following questions in your report:

In the second part of the exercise, why should you let the part pass the second detector before changing the direction of the conveyor?
Test the program at low/high speed. What are the reasons for your program to fail?
In the location file « pickconv.lc », you have the location *grasp.loc.stopped* for grasping the part. How would you operate to teach this location to the robot?

Exercise 2: gluing.

Files to load for this exercise: « glue.lc » and « alltools.v2 ».
Tool : pen.
Tool transformation name: pen.
Initialisation: set output 5 (opens valve for air pressure).
No speed instructions.

The robot is supposed to put glue along trajectory ABCD shown in the next figure. If the glue gun is empty (as detected by input 9, button A on the conveyor manual control box), it must be recharged at location "*recharge*". The end of the recharging operation is indicated by input 10 going high (button B). The robot can then resume its task. At the end of each cycle, it moves to the *wait.loc* location. The glue gun (whiteboard marker) is controlled using instructions **closei** (start gluing/drawing) and **openi** (stop gluing).
A, *B*, *C* and *D* are given in "glue.lc" as locations relative to the *board* frame.
We will allow execute the program in "infinite loop" by "**ex** prog.name , -1". In order to not rewrite the exact same trajectory, shift the whole rectangle 10 mm along the X and Y axes of the board frame, as show on the next figure. You must do that in a single instruction using a compound transform.



Four successive gluing trajectories

Before programming:

- List the inputs used in the exercise.
- For each of them, explain if you will handle it synchronously or asynchronously. Briefly justify your answer. If asynchronously, justify the choice between react and reacti instructions.

Before starting to program, answer the following questions and include the answers in your report. Should the glue default event be handled synchronously or asynchronously? Which are the Val/V+ instructions that can be used? Which one will you choose and why? What is the solution to make sure that the object for which a glue default occurred is correctly glued along the whole trajectory?

Exercise 3:

Files to load for this exercise: « alltools.v2 » plus your own location file.
Tool : parallel gripper.
Tool transformation name: gripper.
Initialisation: set output 5 (opens valve for air pressure).
No speed instructions.

Write a program to de-palletize the pallet located close to the robot. The objects will be stacked at a particular location. Define a pallet frame and use compound transforms to express the coordinates of the objects in the pallet frame in a simple way.
Pallet characteristics: see robotics lab setup – geometry. Object thickness: 15 mm.

Before programming (submit by email with code before testing). You can draw on paper and take a picture (at reasonable resolution) to include in the submitted file.

- Draw the 2x3 pallet, oriented as you see it.
- Draw the pallet frame you would define if you had to define the frame yourself.
- Indicate on the figure the locations you would use to define the pallet.
- Finally write the Val/V+ instruction you would use to create the pallet variable in the program.

Exercise 4: Guarded motions

Files to load for this exercise: none.
Tool : any (not used).
Tool transformation name: null.
Use speed instructions in mm/s. Use:
speed 30 **always** ;
speed 100 **mmps always** ;
in the initialisation of the program to always maintain reasonable speeds.

The apparently simple idea to obtain a guarded motion is to decompose the (straight line) motion into small displacements and test the additional stop condition after each elementary displacement. Implement this idea and add code to calculate the average speed of the motion. Compare with desired speed for various increment lengths and desired speeds.
Then use the Process Control Program (Puma) or a task (RX90) to implement the guarded motion. Again, add code to calculate the actual average motion speed. Compare to desired speed.
Which solution is best?
Why is the average speed always lower than the desired speed?

In order to keep things simple, and without loss of generality, the nominal motion of the robot will be a vertical translation from its initial location downward by a given distance.
The additional motion stop condition monitored by the task run "in parallel" will be the setting to 1 of input 1009 but of course it may be any (more complex) condition.

Note: How to execute a supplementary task in V+ from the robot program (*i.e.* task 0):
Execute 1 prog.name , -1

This will execute "prog.name" in infinite loop. Prog.name just needs to test the condition and stop the motion using the **brake** instruction when the condition turns true.

Send your initial version of each program by email before you test the program on the robot. Send final version once the programs work.