University of Stavanger

Faculty of Science and Technology

Stavanger, August 25, 2021

ELE610 Robot Technology, autumn 2021

ABB robot assignment 3

In the third ABB robot assignment you should make a simple program for UiS ABB robot Norbert. The program should pick and place some pucks on a desk beside the robot. Use the RAPID documentation extensively during the work on this assignment.

Approval of the assignment can be achieved by demonstrating the robot program to the teacher, and then submit a report including RAPID code on *canvas*. Only the RAPID code needs to be submitted, note that this should be a txt-file (not the mod-file that Windows will interpret as a video file). Possibly, you may submit a pdf-file and perhaps also include comments or questions. Make sure that the relevant RAPID code is clearly marked, for example by using Courier font. Nothing but RAPID code needs to be included in the report.

3 Pick and place program for Norbert

You don't need to start from scratch in this assignment, a pack-and-go file that contains a station similar to the actual laboratory in E458 can be used as a starting point, download Pack-and-go-file, `UiS_E458_nov18.rspag`, and make sure that file extension is `rspag`.

3.1 Open pack-and-go-file

Start RobotStudio and open Pack-and-go-file, `UiS_E458_nov18.rspag`; {File}-tab, [Open]-sidebar and follow the wizard. Make sure you store the solution in where you want it, ex. `R:\Lab3`. Use the options and values proposed by the wizard. A question to localize 'PoseMover' may pop up, this file should be in the directory where Robot Studio program is installed; 'ABB Library' 'Components'. A model of robots in E458 should eventually be ready in {Home}-tab.

Karl Skretting, Department of Electrical Engineering and Computer Science (IDE),
University of Stavanger (UiS), N-4036 Stavanger. E-mail: karl.skretting@uis.no.

As in assignment 2 you may clean the station by removing the parts you don't need, here you need Norbert and the desk beside Norbert and the puck on the desk. Run a simulation, 100% speed, and check that the pick and place simulation still works. You may store the station, a new name can be useful ex. 'Lab_3_1.rsstn'.

3.2 Add more positions

Examine the station and associated RAPID code. The station layout has a component 'Puck1', this can be picked and placed by the *smart component* tool 'SC_Gripper' attached to Norbert. The simulated gripper, the smart component in the station, and the actual gripper on Norbert in E458 can both be opened and closed by digital I/O signals that gives a connection from RAPID to the simulated or actual gripper. The names for these outputs are shown in {Controller}-tab, [Configuration Editor] and [I/O System]. You should note that these signals have different names for the simulated and the actual gripper, in RAPID module two versions of the function `closeGrippe(..)` exists, the one not used must be commented out. You note that the difference between these two function is the names of the I/O signals.

We want more positions on the desk where the pucks may be placed. To do this in a flexible way we define the points in an array, see below. We want at least 7 points, note that not all positions on the desk are reachable for the robot. We also want to remove 'target_K1' and 'target_K2' as these are not to be used. Note that the points need to be removed from station as well, synchronization do not remove anything but simply add and update. Do the necessary changes in the code and synchronize to station. Change the movePuck function to place the puck in all positions, and finally put it back in initial position. You may rename the RAPID program, ex. 'prog_3_2' and store it. You may also store the station, ex. 'Lab_3_2'.

```
VAR robtarget targets{7} := [
   [[0,-200,0],  [0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[200,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[0,0,0],     [0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[200,0,0],   [0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[-200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[0,200,0],   [0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9
   ]],
   [[200,200,0], [0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]
    ];
VAR num currentPos := 1;
```

3.3 More pucks

In this subsection five pucks are wanted, and we need to create more pucks. From {Modeling}-tab select [Solid] and [Cylinder], height 30 mm and diameter 50 mm. Give the puck a name and a color and place it on the top of the other puck(s) in position 'targets{1}'.

Change the program so that it moves the five pucks from position 'targets{1}' to 'targets{4}' and back. You may rename the RAPID program, ex. 'prog_3_3' and store it. You may also store the station, ex. 'Lab_3_3'. Synchronize and check simulation and store the program. The simulation should be shown to teacher.

There are many ways to make the RAPID program. One example may use the following global variables

```
CONST robtarget target_K0 := [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9
    ,9E9,9E9,9E9,9E9,9E9]];
CONST speeddata vSlow := v100;
CONST speeddata vFast := v1000;
CONST num puckHeight := 30;
CONST num safeHeight := 240;
CONST num nPos := 7;  ! number of different positions on the
   table
VAR num lastPosNo := 0;
VAR num nOnPos{nPos} := [ 5, 0, 0, 0, 0, 0, 0];
VAR robtarget targets{nPos} := [
  [[0,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[200,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[200,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[-200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[0,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]],
  [[200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]
   ];
```

3.4 The function that moves one puck

You should put some effort into making the function that moves one puck from one position to another position as smooth and effective as possible. The first lines of the function should be

```
PROC movePuck(num fromPosNo, num toPosNo)
    ! moves a puck from position given by fromPosNo to position
    given by toPosNo
    ! num array nOnPos keeps the number of pucks in each
   position
```

Important issues you should consider is

a. The movement should be fast most of the time, except the last millimeters when a puck is placed.

b. The function should update the global variable ‘nOnPos’, and ‘lastPosNo’.

c. After the puck is placed in the correct position the gripper is open, it should then be moved to a position a little bit above the placed puck, ex. two times the puck height, and wait for the next movement there.

d. If the puck to pick is the same as the last that was placed, the robot should simply move down and pick it again.

e. Else, the path from last puck to the one to pick should move through a point with a safe height, to avoid collision with any potential stack of pucks between the two positions.

f. When placing a puck it may be rotated 90 degrees, this will make the movements robust to tiny translations (in x-direction) in each move. These tiny disturbances may add up to a larger distance if hundreds of movements are done.

g. The function may check that the input arguments are within legal ranges, and that there actually is one, or more, puck available at ‘fromPosNo’, and that the ‘toPosNo’ position is different from the ‘fromPosNo’.

h. The `TPWrite` instruction may be helpful if you want to print some values onto the FlexPendant during simulation.

When you are satisfied with the function use it to move the stack in position 1 to another position and back. You don’t need the `getPuck(..)` or the `putPuck(..)` functions any more. You may store the RAPID program as before, ex. ‘prog_3_3’ and the station as before, ex. ‘Lab_3_3’, since this should be a better (?) solution to the same task as previous section.

3.5 A function that moves one stack

Make a function that moves one stack, all pucks in a given position, to another position. Use the function ‘movePuck’ to do the work. The first lines of the function should be

```
  PROC moveStack(num fromPosNo, num toPosNo)
      ! moves all pucks from position given by fromPosNo to
position given by toPosNo
```

3.6 A function that flips one stack

Make a function that flips one stack, the order of the pucks in a given position should be reversed. Not that this can be done by first moving all pucks to an empty position, then move the stack once more to another empty position and finally move the stack back to the initial position. But it is more effective to spread the stack around, and then collect the pucks again. Use the function 'movePuck' to do the work.

The first lines of the function should be

```
PROC flipStack(num fromPosNo)
    ! flips the order of the pucks in position given by
   fromPosNo
```

3.7 Interface on FlexPendant

You should now make a main program that starts by open the gripper and move it above position one, 'targets{1}', ready to close on the upper puck in a stack of five. Pausing the program execution after this instruction, you may place the pucks exactly where the program expects them to be. Then you are ready to continue the program. Now a menu on the FlexPendant should display the interface that makes it possible to select what the robot should do, and perhaps also display the current state. The following instructions are useful: `TPReadFK`, `TPReadNum` and `TPWrite`. You may rename the RAPID program, ex. 'prog_3_7' and store it. You may also store the station, a new name can be useful ex. 'Lab_3_7.rsstn'. Also store this program catalog, with the files, on a memory stick or on your laptop as you need to move this to laboratory E458 for the final task in this assignment.

Possible choices on the menu should be:

- Move puck, or a given number of pucks, from one position to another.
- Move stack from one position to another.
- Flip stack.
- Collect all pucks to stack 1.
- Quit.

3.8 Move pucks using Norbert

Finally the program from previous subsection should be tested on the actual robot, Norbert. Norbert has address '152.94.0.38'. Load the program from previous subsection, ex. 'prog_3_7'. The actual gripper and the virtual gripper use different IO-signals, so you need to swap the commenting of the relevant lines in the gripper module 'gripper.mod'. Also, for the actual gripper on 'Norbert6.08' the IO signals are named 'AirValve1' and 'AirValve2', so these names must be used in the gripper module 'gripper.mod'. When you are pleased with your program you may store it as 'prog_3_8' (a catalog on your memory stick or on your laptop). There is no need to store the station, as the "station" in this case is the actual robot and its environment in laboratory.

When Norbert moves the pucks around as told from the FlexPendant you show this to one of the teachers, and you submit the RAPID code on *canvas*.