Stavanger, July 5, 2024

University of Stavanger

Faculty of Science and Technology

ELE610 Applied Robot Technology, H-2024

ABB robot assignment 3

In the third ABB robot assignment you should make a simple program for ABB robot IRB 140, the one UiS has named 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. Remember to store the work you do at regular intervals, in particular the RAPID code, the mod-files.

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. The RAPID code should be included in the report, and it usually makes up the largest part of 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

This is done as in assignment RS2, section 2.1.

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

3.2 Clean up

In this assignment you only need Norbert, the table and the puck on the table beside Norbert, and the gripper attached to Norbert. Delete everything not needed and check that simulation still works.

Unfortunately the release of a puck may not work as it is supposed to do. The problem seems to be related to the controller, i.e. the RobotWare version, version 6.13 should work with the downloaded pack-and-go file and this version is installed on the stationary PCs in E464. However, the newer version 6.14 which will be used when newest version of RobotWare is installed have given us some problems. It may be that it works with only some of the pucks, but I don't remember if this was a newly created puck (see section 3.4) or the puck from the pack and go file, anyway you may try and if one works replace the ones that don't work with copies of the one which works. You should also change the color. Another solution could be to use the older (or newer?) version of RobotWare and/or RobotStudio.

Run a simulation and check that the pick and place simulation still works.

3.3 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 `closeGripper(..)` 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 any position, and put it back in initial position.

```
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;
```

3.4 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. 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]]
    ];
```

3.5 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. 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`.

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

h. Optional point: The margins are narrow when a puck is gripped. An open gripper has a span of 52 mm and the pucks has diameter of 50 mm, thus the gap is only 1 mm for each finger, and there may also be tape or leftovers from tape that narrows the gap even more. Thus, collision between the gripper and the pucks are likely to happen. To make these collisions less severe a good idea is to move the gripper down not over the puck, but 30 mm to the side of the puck in direction orthogonal to the direction of the gripper gap, and then move the gripper horizontally the 30 mm back to the puck center. A hard collision will now be avoided and if collision happens the result will be that the puck is pushed a little bit to the side. Implement this approach in the `movePuck(..)`-function.

You don't need the `getPuck(..)` or the `putPuck(..)` functions any more.

3.6 A function that moves one stack

Make a function that moves one stack, all pucks in a given position, to another position. If the new position already has some pucks, the new pucks should be placed on top of these. Use the function `movePuck` to do the work, that is the `moveStack(..)`-function should call `movePuck(..)`-function as many times as there is pucks in the `fromPosNo` argument. 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
```

When you are satisfied with the `moveStack(..)`-function use it to move the stack in position 1 to another position and back. You could make a function for this as well, ex: `moveStackTwice(..)`. The `moveStackTwice(..)`-function could have three arguments: `fromPosNo`, `middlePosNo` and `toPosNo`.

3.7 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. Note 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
```

3.8 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. The following instructions are useful: `TPReadFK`, `TPReadNum` and `TPWrite`. 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.

Before the menu is shown it can be a good thing to display the current program state, TPWrite can be used to write current state on the FlexPendant, as:
`Number of pucks in pos. 1-7: 5,0,0,0,0,0,0`
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.

3.9 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`, and I think the name after the last software update is `Norbert-Devservers6.15.08`. Anyway, load the final program to Norbert. The actual gripper and the virtual gripper use different IO-signals, so you need to make the relevant lines in the gripper module `gripper.mod` correct. The names of the actual I/O-signals can be found on the I/O menu on the FlexPendant, I think they are named as `AirValve1` and `AirValve2` so these names should be used in the gripper module `gripper.mod`. An alternative is to use only one I/O-signal, the one named `Gripper`.

When you are pleased with your program you may store the program in a catalog named `RSprog39` or just the module `mainModule39.mod` as a file, 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 by the instructions given from the FlexPendant you show this to the teacher, and you submit your brief report, mainly the RAPID code, on *canvas*.