Stavanger, March 24, 2025

University of Stavanger

Faculty of Science and Technology

ELE610 Applied Robot Technology, V-2025

ABB robot assignment 5

This assignment is based on the assignment made and tested by Markus Iversflaten and Ole Christian Handegård as a part of their Bachelor thesis spring 2020. Karl did a major revision 2024.

In this assignment you will be working in both Python and RobotStudio (on Norbert). The goal is to identify the position of several, randomly placed pucks in the work area by capturing one (or several) image(s) using Python, and then to tell RobotWare where the pucks are, so they may be picked and placed by the RAPID program.

Approval of the assignment can be achieved by demonstrating the robot program to the teacher and submitting a report on *canvas*. The report should include answers to questions, some small fragments of code and some results as described in section 5.5.

It may be difficult to get as much time as you want on the robot and when you want it. Thus, don't postpone the work on this assignment to the last moment but start as soon as you are ready. Much work can be done from anywhere, prepare and test part(s) of the RAPID code and Python code. Then, reserve two hours slot(s) on Norbert, and use this time effective.

The time limit for this assignment is 40 hours for each student. If you are not able to finish all tasks within the time limit you should include a table giving the number of hours each of you have work on each of the tasks listed in section 5.5. It is a good idea to start on this table when you start the work on this assignment.

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

5 Control robot from Python

You may start from the same Pack-and-go file used in RS2 and RS3, it contains a station similar to the actual laboratory in E458. If you haven't already done so, download Pack and Go file, `UiS_E458_nov18.rspag` ↗, and make sure that file extension is `rspag`. While the task in RS3 was to pick and place pucks from fixed positions the task here is to locate the pucks on the table beside Norbert using a camera and image processing, then pick one puck and place it in a wanted position, perhaps based on user input. User interface is here through the Python program on a PC, the FlexPendant is only used to start the RAPID program[1].

The principle for communication between a Python program and RAPID is quite simple. RAPID is the passive agent and this part is explained in section 5.1 and Python is the active agent and this part is explained in section 5.2. Image Processing is explained in section 5.3 and coordinate transformations in section 5.4. Finally, what you should do is described in section 5.5.

5.1 RAPID code

This section explains the principle for communication between a Python program and RAPID. Python is the active agent and RAPID is the passive agent in this communication. Thus, from the RAPID viewpoint, communication simply happens when a RAPID variable changes its value.

A simple RAPID program may be in just one module. It starts by defining *global* variables that can be accessed from within all functions of the program, in our case here these should include the variables that can be changed from outside, i.e. Python. The two most important ones for communication are:

```
VAR num WPW:=0;   ! What Python Wants
VAR num WRD:=0;   ! What RAPID Does
```

`WPW` is 0 when Python don't want the robot to do anything, the robot just wait. When this variable is set to a positive integer value the robot should start by changing `WRD` to this value, then setting `WPW` to zero indicating that Python may now give another task to do after the `WRD` task is done, finally the robot does the `WRD` task and then start the main loop again. Python may set the value of `WPW` when it is zero if it keeps track of which task the robot already does, `WRD` may be read, and also Python must know which global variables each task uses, and avoid changing these while the task is in progress. It may be more secure if Python set the value of `WPW` when both `WPW` and `WRD` are zero, but this may give unwanted pauses in the robot movements.

[1]Actually this part may also be done from Python

The structure of the MainLoop function should be like below

```
PROC MainLoop()
   TPWrite "MainLoop starts";
   WHILE TRUE DO
      WRD := 0;    ! robot does nothing, waits
      IF (WPW = 0) THEN
         TPWrite "Robot waits for Python to set WPW";
      ENDIF
      WaitUntil (WPW <> 0);
      TPWrite "Python wants to do task WPW = "\Num:=WPW;
      WRD := WPW;
      WPW := 0;
      TEST WRD
         CASE -1: RETURN;          ! Quit MainLoop
         CASE  0: WaitTime 0.1;    ! Hmmm, should not happen
         CASE  1: Task01;          ! do function Task01
         CASE  2: Task02;          ! do function Task02
         ! ... more cases ...
      ENDTEST
   ENDWHILE
ENDPROC
```

The functions Task01, Task02, and onward are the different functions that actually do the wanted task. Typically, these functions use global variables, of which some have been assigned values from Python. It is important that Python first set the values of RAPID variables and then set the WPW value. Also, a global RAPID variable used by the running task function **must NOT** be set while the task function is running.

A common information to pass from Python to RAPID is a position that could be given as a robtarget or as an offset (dx, dy, dz) from a fixed position. The first alternative is the more flexible one as it also contains orientation, and the second alternative is perhaps the easiest. Below is an example where more global variables are defined and the function Task01 is given, it moves the camera to a given position and wait a moment.

```
! below: not complete definition for work object
TASK PERS wobjdata wobjTableN:=[FALSE,TRUE,"",
      [[150,-500,8],[0.707106781,0,0,-0.707106781]], ... ]
! below: not complete definition for tools and speed
TASK PERS tooldata tGripper:=[TRUE,
      [[0,0,114.25],[0,0,0,1]], ... ]
TASK PERS tooldata tCamera:=[TRUE, ... ]
VAR speeddata robotSpeed := v100;
! below: not complete definition for points
CONST robtarget p0:=[[0,0,0],[0,1,0,0], ... ];
VAR robtarget p1:=[[100,100,0],[0,1,0,0], ... ];
! below: numbers for offset position 1
VAR num dx1:=0;  ! global variables for offset
VAR num dy1:=0;  ! that may be set by Python
VAR num dz1:=85; ! minimum distance from camera to table

```

```
PROC Task01()
   TPWrite "Task01 move robot/camera to position 1";
   ! assuming Python set p1 before Python set WPW := 1
   MoveJ p1,robotSpeed,z1,tCamera\WObj:=wobjTableN;
   ! below: not complete alternative using offset
   ! MoveJ Offs(p0,dx1,dy1,dz1), ... ;
   WaitTime 0.1;
ENDPROC
```

There should of course be more global variables (ex. defining a second point) and more functions for the different tasks that should be done, in particular a task, or several, to pick up a puck from a given position and place it in another position. This will also need the function to open or close the gripper as in assignment RS3. Also more test functions should be made, in particular `TestMove()`. The function to move a puck could be similar to the function in RS3 but the arguments should not simply be two numbered positions but two points given as target points or offsets. The `MovePuck` function should also be robust in the sense that it should approach the puck from the side to better avoid hard collisions. The listing below shows some more relevant code in (uncompleted) functions.

```
PROC main()
   CloseGripper(FALSE);   ! open gripper
   MoveJ Offs(p0,0,0,250), robotSpeed, z10,
      tGripper\WObj:=wobjTableN;
   ! starts main loop or a test
   ! MainLoop;
   Test01;
ENDPROC

PROC Test01()
   p1 := ...;  ! normally done by Python, but not in test
   Task01;
ENDPROC

PROC TestMove()
   p1 := ...;  ! normally done by Python, but not in test
   p2 := ...;  ! normally done by Python, but not in test
   MovePuck p1,p2;
ENDPROC

PROC MovePuck ( robtarget fromPnt, robtarget toPnt )
   ! moves a puck from position given by fromPnt
   ! to position given by toPnt
   CloseGripper(FALSE);   ! open gripper
   ...
ENDPROC

PROC CloseGripper(bool state)
   ! Function to open (state = FALSE) or
   ! close (state = TRUE) the gripper
   WaitTime 0.1;
   IF state THEN
      setDO ... ;
   ELSE
      setDO ... ;
   ENDIF
   WaitTime 0.2;
ENDPROC
```

5.2 Python communication code

We can read or write the value of a RAPID variable from a Python program using the ABB package Robot Web Services RWS ↗ and the common Python `requests` ↗ package. The UiS package `rwsuis` ↗ should make this more easy, it is one of the packages that should be installed for ELE610, see section 2.3 of the Fragments of Python stuff ↗ document. To be able to communicate to a robot your computer need to be on the same local wireless network `robot`, see assignment RS2 section 2.8.

To test communication on a more basic level you may use `requests` instead of the preferred `rwsuis` package to establish and test communication. The Python commands could be:

```python
import requests
norbert = "http://152.94.160.198/rw/panel/ctrlstate"
auth=requests.auth.HTTPDigestAuth("Default User", "robotics")
# make an get-request
response = requests.get(norbert, auth=auth)
print(response.status_code)  # an integer
print(response.text)         # a string
# the response may be returned in JSON format
response = requests.get(norbert+"?json=1", auth=auth)
print(response.text)         # a string
```

The **preferred way** of communication is through the package `rwsuis` ↗, which includes the RWS class. Documentation of this package is available on web: ABB Robot with Machine Vision ↗. The `rwsuis` package should be easier to use, for our specific task, than the general `requests` package. An example for a part of the Python program is:

```python
from rwsuis import RWS
help(RWS)  # display useful help on package
norbertIP = "http://152.94.160.198"
robot = RWS.RWS( norbertIP )
wrd = robot.get_rapid_variable( "WRD" )
if (wrd == 0): print( "Robot is waiting for Python to set ’WPW’
   ")
(tr, rot) = robot.get_robtarget_variables( "p1" )
print( f"Translation for ’p1’ is x={tr[0]:.2f}, y={tr[1]:.2f},
   z={tr[2]:.2f}" )
new_rt = [0,0,314]
```

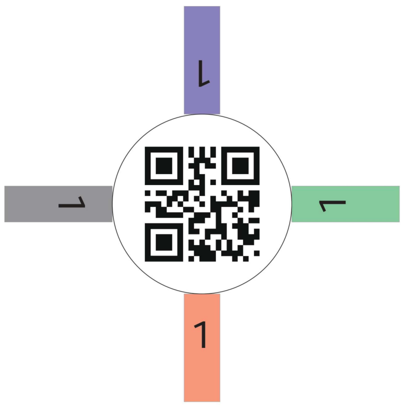

Figure 1: A QR code tag as here may taped on some of the pucks. There are also some pucks where the QR code is printed into the pucks. If you need to print new tags see announcement in *canvas*.

```python
robot.request_rmmp() # mastership manual mode, GRANT on
    FlexPendant
# robot.request_mastership()  # automatic mode
robot.set_robtarget_translation( "p1", new_rt )
robot.set_rapid_variable( "WPW", 1 )
```

5.3 Image processing

To locate the pucks in the work area, you should scan the QR codes on top of them. This will reveal the pucks' positions. To do this, you must first capture an image with the camera in Python, and then put the image through a QR scanner. It may also be a good idea to process the image before passing it to the QR scanner. This will make the image easier to decode, which in turn will make your solution more robust.

5.3.1 Capturing an image

To capture images in this assignment, you will need to use a **uEye XS** camera. You can capture the image either through OpenCV ↗ or PyuEye ↗.

Using OpenCV:
The easiest way to retrieve an image from the camera in Python is through OpenCV. OpenCV has a general API for capturing images, and thus, has only general functionality. This means that many of the cameras parameters will be left untouched. Below is how to capture an image in OpenCV:

```python
import cv2
cap = cv2.VideoCapture(1)
# Change resolution to 1280x960
```

```
cap.set(3, 1280)
cap.set(4, 960)
(ret, frame) = cap.read()
```

Setting the image resolution can also be done in OpenCV, as shown. This will prove useful later.

Using PyuEye:
IDS has created their own API `pyueye` for controlling their cameras. With this, you gain access to all functionality within the XS camera through Python. This also means that the API is harder to use than OpenCV.

For this assignment, OpenCV should be sufficient. If you still would like to use PyuEye, then you may find some help in the functions provided in the IDS Software Suite ↗.

5.3.2 Image preparation

The QR scanner might not be able to decode images with too much noise and/or too little contrast. You should therefore try to reduce noise and increase the contrast in the image.

Reducing image noise:
Reduced image noise can be achieved through image filtering. Some of the most common filters are Gaussian blur, Median filter and Bilateral filter. These all exist in the OpenCV library. You must choose one of these to use, by reading about them in OpenCV tutorial ↗ or elsewhere on the web.
Note: when working with QR codes, some filters are definitely better than others. Try to find the best one!

Increasing image contrast:
Increasing the contrast in the image will make QR codes stand out more. There are several ways to achieve a greater image contrast. Methods worth checking out: `cv2.normalize` ↗, `cv2.equalizeHist` ↗ and `basic linear transform` ↗.

5.3.3 Using the QR scanner

There are several QR scanners for Python. In our experience, `pyzbar` ↗ is both fast, easy to use, and has all the functionality you will need. It should already be installed on the laboratory computers.

The only function you will need from `pyzbar` is `decode`. It simply takes an image as input argument and returns a *Decoded object* (if the image contains QR codes). This contains the pixel indexes for the four corners of the QR code, from these you can find the center and the orientation of the QR code

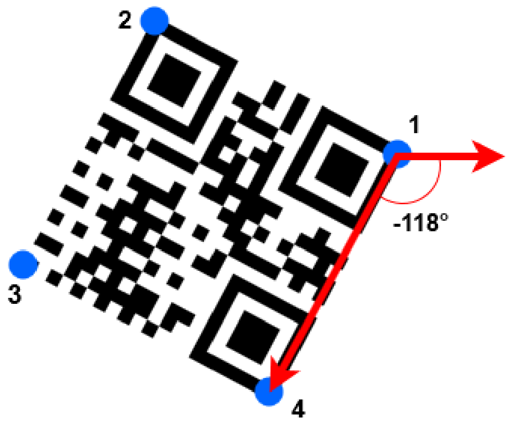

Figure 2: The four corner points for a QR-code (blue dots) as found by `pyzbar` can be used to find the center of the puck by simply averaging. The points can also be used to find the QR code orientation, which is also the puck orientation. Using this information the pucks can be stacked or placed in an orderly manner, so that the QR-codes have the same orientation.

in the image. To access any information from the QR code, simply navigate through the different `Decoded` object parameters:

```python
from pyzbar.pyzbar import decode
...
data = decode(image)
# The object will look like this:
Decoded(
    data=b'Puck#1', type='QRCODE',
    rect=Rect(left=27, top=27, width=145, height=145),
    polygon=[Point(x=27, y=27), Point(x=27, y=172), Point(x
    =172, y=172),Point(x=172, y=27)]
)
print( f"First point is (x,y) = ({datapolygon[0][x]},{
    datapolygon[0][y]})" )
```

5.4 Coordinate transformation

The complete theory of coordinate transformation is not trivial, the student needs a basic understanding of mathematics (geometry) and optics (camera) and the teacher (or textbook) needs to make a good and clear explanation accompanied by sufficient clear illustrations. A decent presentation is given by FDX-labs ↗. A clear mathematical presentation is given by the Intelligent Motion Lab ↗ which is a part of the University of Illinois. You may also find other web pages or videos explaining this transformation.

A linear transform is often sufficient, and here we also simplify by keeping the camera in a fixed height above the table (ex: 250 mm) and camera is assumed to be above center of table, if camera is translated the table coordinates should have the same translation. Using *homogeneous* coordinate representation an image point is a column vector $\mathbf{p}_i$ and a table point a column vector $\mathbf{p}_t$. Here, camera is assumed to be in a fixed position above the center of the table `(0,0,250)`. Then coordinate transformation consists of rotating, scaling and translation which can be collected and represented by a matrix T

$$
\begin{aligned}
\mathbf{p}_t &= T \cdot \mathbf{p}_i \quad \text{or} \quad \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} && (1) \\
x_t &= T_{11}\, x_i + T_{12}\, y_i + T_{13} && (2) \\
y_t &= T_{21}\, x_i + T_{22}\, y_i + T_{23} && (3)
\end{aligned}
$$

Assuming the camera is in same height and the same orientation (direction) but translated to `(xCam,yCam0,250)` the table coordinates will have the same translation, i.e. $x_t = x_t + xCam$ and $y_t = y_t + yCam$.

If you have a set of K known table and image coordinates the matrix T can be found using linear algebra. The table coordinates are collected in a $3 \times K$ matrix P_t, and the image coordinates are collected in a similar $3 \times K$ matrix P_i. These are related by $P_t = T \cdot P_i + E$, where E is a $3 \times K$ error matrix. The unknown 3×3 matrix T, and the error matrix, can be found by solving this matrix equation such that the error matrix is minimized in a least squares sense, i.e trace norm also known as Frobenius norm. In MATLAB the solution is simply found by `T = Pt/Pi;` and in Python the function `numpy.linalg.lstsq` ↗ can be used. Collecting point coordinates at a different height (ex: 500 mm) will give another transformation matrix.

5.5 What to do

You should do the tasks listed below. You may skip the optional ones if you have used the hour budget, note that an hour list should be included.

a. Make the first version of the RAPID program by including the parts explained in section 5.1. The missing program parts should be filled in. The `MovePuck` function and the `CloseGripper` function can initially be as in assignment RS3, but should eventually be modified to be robust.

   The tasks to do should be to pick a puck from one (known) position and to place it in another (known) position. The third task to do is to go to a position 300 mm above the table and stop, and stay there when the image is captured and analyzed.

b. Test Python communication as explained in section 5.2. You may do the commands directly from the command line interface (CLI) in Python but it is better to have them in a small script (program). Use `rwsuis` and connect to Norbert, you don't need to load your own program into Norbert but simply use the one already loaded, most likely there is a `robtarget` defined, this can be found using the FlexPendant and the Data program there. Your Python script should read the values of one RAPID `robtarget` variable and print it on the standard output. You don't need mastership to do this.

   Check that you get the same values in Python as the variable in RAPID. The report should contain the lines in your Python script and a confirmation that the code works as expected.

c. Use IDS camera on the gripper to to capture images. This can be as image acquisition in assignments IA2 and IA3, or simply using OpenCV as explained in section 5.3. Take some images of the QR-code tagged pucks placed on the table besides Norbert. Get images of single pucks, both where QR-code is on paper attached to the puck and where QR-code is integrated in the puck and images of several pucks. Take some images from different heights ranging from 150 mm to 500 mm. Store these images on your laptop, or a memory stick.

   The report should contain one example image.

d. Write Python code that uses `pyzbar` to get the coordinates for the pucks in the images taken in point above. You may need appropriate preprocessing, see section 5.3, to make this work properly. The Python code should print out the image coordinates for the center of each puck in all of the images.

   The report should contain the puck location in the example image shown in point c above. It should also include a summary of the results, i.e. how many pucks out of all the were correctly located.

e. You should now capture a set, or perhaps two sets, of images to use for estimating the transformation matrix as explained in section 5.4. You may use the coordinates of `tGripper` as the coordinates of the P_t matrix, as the camera is fixed on the gripper. Thus, the transformation matrix T is the transformation matrix from image (pixel) coordinate to table coordinate relative to the gripper.

   You may use the robot to place one puck in origin of the table coordinate system, and perhaps also some pucks in some other know positions. Take a set of images, the camera direction should be straight down, using different offsets (ex: dx and dy in $\{-200, 0, 200\}$ mm) at a fixed height (ex: z=200 or 250 mm). This should give at least 9 points where coordinates are given (or found) in both image coordinate system and table coordinate system adjusted by camera position. These points can be used to estimate the transformation matrix $T^{(200)}$ where the superscript indicate the height of the gripper/camera. You may also do this for a different height and estimate $T^{(500)}$. The report should contain the values, with appropriate precision, of the estimated matrices $T^{(200)}$ and $T^{(500)}$.

f. You should find a transformation matrix for any height z by interpolation between two known matrices by

$$T^{(z)} = \frac{z_2 - z}{z_2 - z_1} T^{(z_1)} + \frac{z - z_1}{z_2 - z_1} T^{(z_2)}. \tag{4}$$

Use this equation to find $T^{(350)}$ and compare this to the matrix found by the method used in the previous point for height 350 mm. The report should contain the values, with appropriate precision, of matrix $T^{(350)}$ both from linear interpolation and estimated directly from images.

g. Now you should make all parts of the solution work together. Both the Rapid program allowing the robot to be controlled, and the Python program controlling the robot, and the image acquisition and processing in Python should work together.

The first test is to place one puck on the table, in a position unknown to the robot. Then run the solution. From Python make the robot go to a position where an image should be captured, you may have an overview from 500 mm height and perhaps a more detailed view from 200 mm height to more precisely locate the puck. When the puck is located go and get it and then place it in a default position. The report should contain a confirmation that the code works as expected. You should also demonstrate this part to the teacher in the robot laboratory.

h. The second test is to use three or four pucks randomly placed on the table. Let your solution stack the pucks in a default position. Improve the program by making it robust. The report should contain a confirmation that the code works as expected. You should also demonstrate this part to the teacher in the robot laboratory.

i. Finally, if there is still time left, you should collect the Python code made in this assignment into a program `appImageViewer5X.py` where `X` is your group name. This program should inherit `appImageViewer2.py`, or your own (improved) variant `appImageViewer2X.py`, similar to the way `appImageViewer3.py` and `appImageViewer4.py` do, see documentation videos and the Fragments of Python stuff ↗ document. The program should continuously show the last image captured and perhaps also indicate the results of image processing. How you want any user interaction is much up to you to decide.

The report should contain a brief summary of the new features, added menu actions, of this image viewer program. You should also demonstrate this part to the teacher in the robot laboratory.

j. This point is **optional**. To get a more accurate and more general transformation matrix a tool for the camera is needed. The tool `tCamera` should be as the `tGripper` but with an offset in position (x,y,z), the orientation (direction) of `tCamera` can be as for `tGripper` as this is almost correct.

k. This point is **optional**. The tool `tCamera` may not be as accurate as it should be in the RAPID program. The offset may be slightly off from correct values, also the orientation may need some adjustments, the camera might not be precisely attached to the gripper tool and also the camera sensor and lens may be misaligned in the camera housing (the specification says that there is a production tolerance of more than one degree).

   In RobotStudio the position and orientation of the camera tool is shown as an axis cross that should be located where the center of camera lens is and oriented in the direction that the camera "sees". When the camera is placed in a point directly above the center of a puck, and the point has orientation z-axis straight downwards, the puck should appear in the center of the image. When the point is moved straight up or down to size of the puck in the image is changed but the center of the puck should still be in the center of the image.

   You should yourself think of a method to adjust the RAPID tool, you may consult RAPID and RobotStudio documentation, and perhaps also look for hints on the web. It may be an idea to start by exaggerate the alignment to clearly see the effect, for example rotate the orientation of the tool 10 or 15 degrees around its x-axis or its y-axis.

   Don't use to much time on this task. The report should contain a brief description of the procedure you did to align the camera tool, and the resulting definition for `tooldata tCamera` used in RAPID.