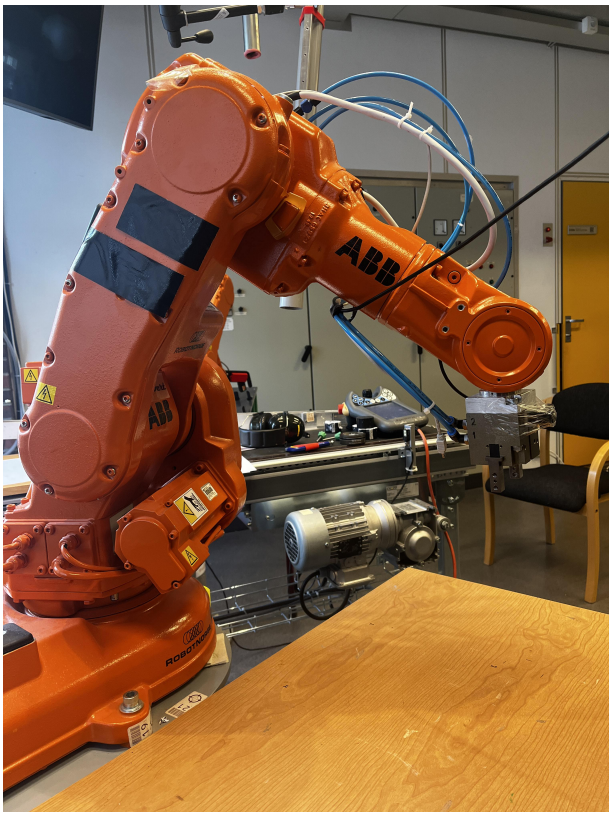
Image Acquisition combined with Norbert robot

ELE610 –RS5

Sondre Lyngstad (256437)

Victoria Lie Fredriksen (265250)

October 2023



Contents

1 Introduction

In this assignment we will be working in both Python 3 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 telling RobotWare where the pucks are, so they may be picked and placed by the RAPID program
Assignment description from [1].

We were to groups (Simon Lundgren - Sigurd Ottesen and Sondre Lyngstad - Victoria Lie Fredriksen) who worked closely on this assignment. Therefore the two groups report will be the almost identical.

2 Interpretation and approach

We used quite some time to understand and grasp this assignment. This was necessary to understand what should be done when and how.

The first step in this lab assignment was to solve the complication on how to communicate with Norbert from python on our computer. The way of solving this was to first make a GET request, and then making a POST request. A POST request works like this: First, we make a simple system in Robot Studio Rapid. This involves a simple target called target_k0 and a while loop. The intention is to check the connectivity between python and the robot by overwriting the uploaded variable target_k0 on Norbert. The Rapid code is shown below.

```
1    PROC main()
2        WHILE TRUE DO
3            MoveJ target_K0 , v200 , z10 , tGripper \ WObj ...
                 :=wobjTableN ;
4            WaitTime 1;
5        ENDWHILE
6    ENDPROC
```

The python code is illustrated below. We import some necessary packages and request master ship for the given IP-address and make a transition from "target_k0" to "newrob_target" The robot now moves to "new_robtarget" when running the code in python. Post request is now achieved

This part turned out to be quite troublesome, and we spent a lot of time to find solutions to this rather simple problem. More on this in the reflection section later on.

```
import requests
import time
from rwsuis import RWS  # Import the RWS module

robot=RWS.RWS("http://152.94.0.38")
robot.request_rmmp()

time.sleep(5)

new_robtarget=[0,0,300]
robot.set_robtarget_translation("target_K0",new_robtarget)
```

The next step in the lab is to make a QR-code scanner, which could return the coordinates of the puck.

Lastly, when all of the above was solved, we had to merge everything (Image acquisition, python and RAPID) together, so that the robot could pick up pucks, wherever they lay on the table.

3 Solutions & Result

3.1 Finding the QR-Code

It was chosen to have a video feed running continuously looking for QR-Code in the video frame. The module is made up of on class initializing the camera with some simple settings like resolution and frame rate of the feed.

Listing 1: Initializing Camera

```
self.cap = cv2.VideoCapture(0)

# Define the new width and height for the resolution
new_width = 1280
new_height = 960

# Set the new resolution
self.cap.set(3, new_width)  # FRAME_WIDTH
self.cap.set(4, new_height)  # FRAME_HEIGHT

# Set the desired frame rate (in frames per second)
desired_frame_rate = 12
self.delay = int(1000 / desired_frame_rate)
```

When the Camera is initialized the video feed will start, but due to some issues with the camera some error handling is necessary.

```
1   def video_feed(self):
2     print("starting Video Feed. Please Wait")
3     while True:
4         ret, frame = self.cap.read()
5
6         if not ret:
7             print("The Camera did not start properly. Program was ...
                  terminated")
8             break  # Exit the loop if no frame is available
```

Some times the camera does not get a resolution ending in a broken feed. To prevent this the *ret* value is checked to see if initialization is done successfully. if not the program is stopped. The *decode* function from the *pyzbar* package looks for QR-Codes in each frame and returns a list with all found QR-codes represented a dictionary with different information. The information in *polygon* contains all the corner points of the decoded QR-code. These points are used to calculate the center of the puck.

```
1     for obj in decoded_objects:
2         points = obj.polygon
3         self.last_known_qr = cv2.convexHull(np.array([point for point ...
              in points], dtype=np.int32))  # Convert to CV_32S
4         # Calculate the center coordinates
5         img_center_x = int(sum(point.x for point in points) / ...
              len(points))
6         img_center_y = int(sum(point.y for point in points) / ...
              len(points))
```

Due to the difference in the coordinates in relation to the origin for the frame and the robots target. Some conversion in need to convert the coordinate in the frame to a usable work object for the ABB-robot

```
1          center_x = img_center_x - (self.resolution_width/2)
2          center_y = img_center_y - (self.resolution_height/2)
3
4          WO_center_x = - center_y
5          WO_center_y = - center_x
6
7          # Convert pixel coordinates to millimeters
8          self.center_x_mm = WO_center_x * self.pix_to_mm
9          self.center_y_mm = WO_center_y * self.pix_to_mm
```

Then the video graphics are updated showing the user the information that has been obtained from the last frame.

```
1          # Display both pixel and smoothed millimeter coordinates near ...
               the QR code
2          label = f'Pixel: X={WO_center_x}, Y={WO_center_y} / MM: ...
               X={self.center_x_mm:.2f}, Y={self.center_y_mm:.2f}'
3          cv2.putText(frame, label, (img_center_x, img_center_y - 10), ...
               cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
4
5          if len(self.last_known_qr) > 0:
6              cv2.polylines(frame, [self.last_known_hull], True, (0, ...
                   255, 0), 2)
```

Figure 1: Video feed with a detected qr-code

If a QR-code is found the value is stored in case there are some frames where the *pyzbar.decode* are note able to find the QR-code.

```
1       if qr_code_detected:
2           self.update_variables(self.center_x_mm, self.center_y_mm)
3
4       def update_variables(self, center_x_mm, center_y_mm):
5           with self.lock:
6               self.center_x_mm = center_x_mm
7               self.center_y_mm = center_y_mm
```

This function is useful, because it allows the graphic to stay relatively stable and allows the user to retrieve this value when needed using the *get_ variables* function

```
1      def get_variables(self):
2          with self.lock:
3              return self.center_x_mm, self.center_y_mm
```

This is a function able to run in parallel with the video feed due to the use of multi threading. Now the user can get the coordinates of the detected QR-code in relation to the robots coordinate system and create a work object to make the robot move to the desired position

3.2 Python, RAPID and Image acquisition merged - Main code

Because we wanted to have the camera running in parallel with the python code controlling the robot, we used the thread method. This is shown in the code below:

```
##### Video Feed In Another Thread #####
updater = qr_code_detection()
video_thread = multi.Thread(target=updater.video_feed)
video_thread.start()
```

With this lines of code, we are able to have the qr_code_detection() running in parallel with the rest of the code.

In the code below we initialize the robot and ask for mastership, so that we can edit the rob-targets.

```
### initialization Of The ABB Norbert Robot ###
robot = RWS.RWS("http://152.94.0.38") # Norberts IP
robot.request_rmmp()                  # Mastership request in manual mode
time.sleep(10)                        # Give time to accept request
```

We tried to use flags called wait_for_python and wait_for_RAPID to keep order and make sure every task was done correctly. In the code extract below we set the start position of the robot, and initialize a flag, before the puck detection part begins.

```
##### Setting New Robtarget and Rapid Variable
wait_for_RAPID = None
new_robtarget = [0,0,15]
robot.set_rapid_variable("wait_for_RAPID", "TRUE")
robot.set_robtarget_translation("target_K0", new_robtarget)
wait_for_RAPID = robot.get_rapid_variable("wait_for_RAPID")
print(wait_for_RAPID)

time.sleep(5)  # To ensure datatransfere before continue
```

In addition, we also print the flag wait_for_RAPID to ensure that the flag is set correctly.

In this next part we implement a while-loop that waits for the wait_for_RAPID variable is changed i.e, the main loop has been run once and the robot is moving to the given rob-target.

```
##### Stopping the script until The main RAPID program is run once
while wait_for_RAPID == "TRUE":
    wait_for_RAPID = robot.get_rapid_variable("wait_for_RAPID")
```

The main program in RAPID is shown in the picture below:

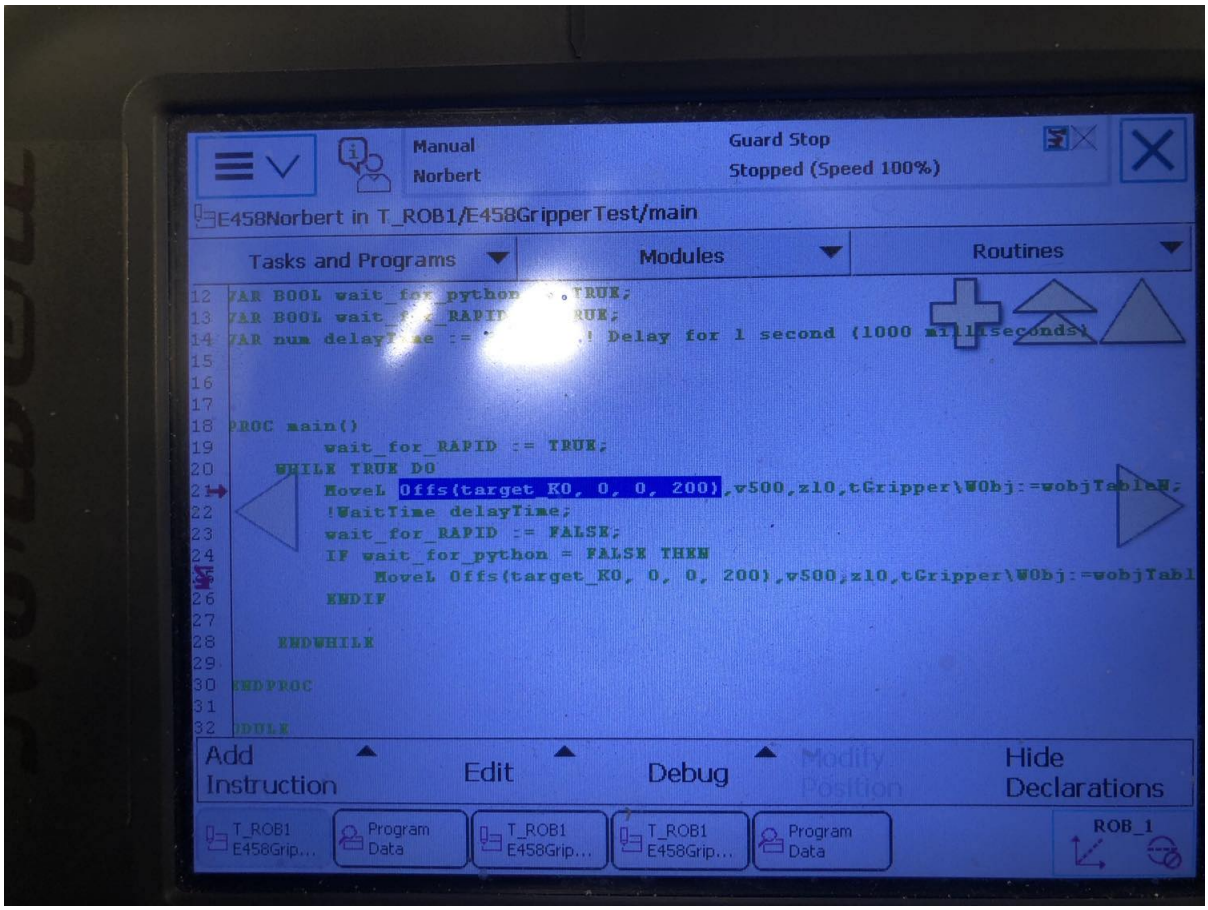


Figure 2: Caption

In this next part, we use our qr_code_detection code, to extract and use the distance in mm from the origin of the picture with axis in relation to the grippers axis. These calculations are shown in qr_code_detection() part. We take these variables called center_x_mm and center_y_mm and add the to the grippers position. To get the grippers position we use the method from the RWS class robot.get_gripper_position(). We save this as (trans, rot) to save and use the position and orientation the gripper has. This is shown below.

```
1 ##### Getting the cordinates for the puck #####
2 center_x_mm, center_y_mm = updater.get_variables()
3 (trans, rot) = robot.get_gripper_position()
4 x = trans[0] + center_x_mm - 55
5 y = trans[1] + center_y_mm
6 z = 15
7 new_robtarget2 = [x,y,z]
8 print(new_robtarget2)
```

We also need to take the cameras offset from the gripper in to our calculations. The camera is mounted 55mm in front of the gripper, this correspond to -55 in x-axis. You can now see from line 4 in the code extract below, that the x-coordinate of the puck is trans[0] + center_x_mm - the camera offset from the gripper. Trans[0] is the x-coordinate of the gripper, extracted in line 3. The same thinking applies for the y-coordinate. We can set the z-coordinate to 15mm, because we know the pucks height, which is 30. In theory, this newly created target should be at the pucks center.

When a new robot target is made, it needs to be written to RAPID. Since the robot is i manual mode a new write access needs to me sent and accepted by the user

```
1 ##### Recuesting new write acsess to make new robtarget the puck position
2 robot.request_rmmp() # Mastership request in manual mode
3 time.sleep(10)       # Give time to accept manual request
```

When permission is granted the target with the new calculated puck position is written and a small pause in the program is added to ensure that nothing goes to fast

```
robot.set_robtarget_translation("target_K0", new_robtarget2)
time.sleep(5) # To ensure datatransfere before continue
```

Then the program retrieve the new target to see if it has been updated correctly.

```
robot.get_rapid_variable("target_K0")
print(robot.get_rapid_variable("target_K0"))
robot.set_rapid_variable("wait_for_python", "FALSE")
time.sleep(5)
print(robot.get_rapid_variable("wait_for_python"))
```

Figure 3 show the new updated target with the puck coordinates

Since Norbert is not calibrated, the best way to show that this the functions work, is to show the new variable saved on the flex pendant and show that i correlates to the center of the disc calculated in qr_code_detection(). A picture of the calculated robtarget is shown in the picture below.

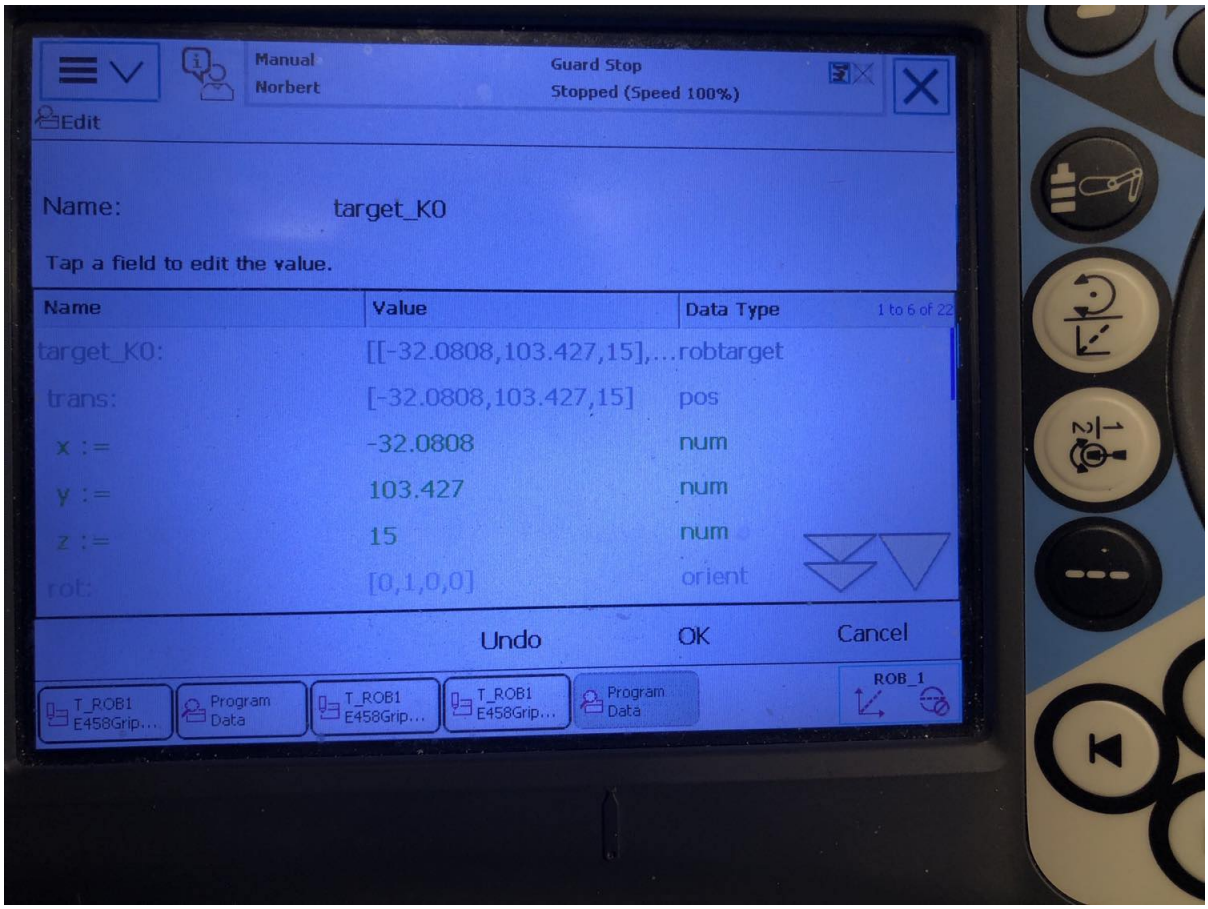


Figure 3: Caption

Comparing the values from figure 1 and 3 it is possible to see that the located QR-code and its corresponding coordinates found using python is successfully transferred to RAPID as a robot target. Note that the x-coordinate for the robot target is 55mm behind the coordinate in the photo due to the position of the camera.

4 Reflection

The 40 hour time limit was quickly filled when working with this task. We encountered many time-consuming problems, which made this task difficult. We will discuss these here.

4.1 Problem 1 - communication between python and RAPID:

The communication between python and RAPID worked poorly in our opinion. An example of this was running the communication test showed in the assignment. This code is shown below in "comTest".

Listing 2: comTest

```
robot = RWS.RWS("http://152.94.0.38")
robot.request_rmmp () # Mastership request in manual mode
time.sleep (5)
new_robtarget = [0 , 0 , 300]
print(new_robtarget)
robot.set_robtarget_translation("target_K1",new_robtarget)
```

We used some time learning the "timing" i.e, when we could hit play on the Flex Pendant to make the robot respond. When figuring this out, we ran the exact same comTest code with the same timing on two different computers, resulting in the robot only responding to one of them.

4.2 Problem 3 - Norbert not calibrated:

At the time we started on this assignment, the Norbert machine was not calibrated. This resulted in it being hard to work with and hard to modify our code. An example of this is the z variable of our calculated Rob-target (where the puck was). We sat this variable to 15 (middle of the pucks height), but when Norbert went for this point, it settled on a height more like 300 in real life. Although we could see that this point was stored on the flex-pendant at height 15. More of this is explained in the solution part, with pictures to prove it.

4.3 Problem 2 - shortage of time:

With the problems of the assignment, earlier explained, and the complexity of this task, the 40 hour time-limit was hit fast. Therefore we present what we have managed to do, even though it is not a complete solution.

References

[1] Karl Skretting. Abb robot assignment 5. Technical report, Universitet i Stavanger, 2023. Utdelt materiale.

Table 1: Hours worked on the project.

Name / date	Sondre Lyngstad	Victoria Lie Fredriksen
18.10	4	3
20.10	3	3
21.10	8	8
22.10	8	9
23.10	8	8
24.10	6	6
25.10	5	5
Sum hours	42	42