

ABB robot Assignment 5

Antón Maestre Gómez and Daniel Linfon Ye Liu

- 1) 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 tool tCamera should have a different offset than the tGripper but may initially have the same orientation. Simply use a ruler to measure the offset as well as you can.

After taking the necessary measurements, the position of the tCamera is:

```
TASK PERS tooldata tCamera :=[TRUE,[[ -52.5,0,34.25],[0,0,0,1]], [1,[-0.095984607,0.082520613,38.69176324],[1,0,0,0],0,0,0]];
TASK PERS tooldata tGripper:=[TRUE,[[0,0,114.25],[0,0,0,1]], [1,[-0.095984607,0.082520613,38.69176324],[1,0,0,0],0,0,0]];
```

In addition, after completing test01 with an assigned p1 value, we can verify that it is performed correctly:

```
PROC main()
  CloseGripper ( FALSE ) ; ! open gripper
  MoveJ Offs ( target_K0 ,0 ,0 ,250) , robotSpeed , z10 ,
  tGripper \ WObj := wobjTableN ;
  ! starts main loop or a test
  ! MainLoop ;
  Test01 ;
ENDPROC

PROC Test01()
  p1 := [[100 ,100 ,150] ,[0 ,1 ,0 ,0] , [0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9] ] ; ! normally done by Python , but not in test
  Task01 ;
  TPWrite " Test01 done successfully! ";
ENDPROC

PROC Task01()
  TPWrite " Task01 move robot / camera to position 1";
  MoveJ p1 , robotSpeed , z1 , tCamera \ WObj := wobjTableN ;
  WaitTime 0.1;
ENDPROC
```

```
T_ROB1-> Task01 move robot / camera to position
1
T_ROB1-> Test01 done successfully!
```

- 2) 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.

```
from rwsuis import RWS

norbertIP = "http://152.94.160.198"
robot = RWS.RWS ( norbertIP )

( tr , rot ) = robot .get_robtargget_variables ("p1")
print ( f"Translation for 'p1' is x = { tr [0]:.2f} , y = { tr [1]:.2f} , z = { tr [2]:.2f}" )
```

Translation for 'p1' is x = 0.00 , y = 0.00 , z = 500.00

Name:	p1	
Tap a field to edit the value.		
Name	Value	Data Type
p1:	[[0,0,500],[0,1,0,0],[-1,...	robtargget
trans:	[0,0,500]	pos
x :=	0	num
y :=	0	num
z :=	500	num
rot:	[0,1,0,0]	orient

- 3) This point is optional. Test writing access from Python into RAPID variables. When the Python code has printed out the values of a RAPID robtargget as in the previous point, the code should be expanded. Change the values of the robtargget in Python and write it back to the robot. Mastership is required as explained in section 5.2. Check that you get the same values in RAPID as the variable in Python. The report should contain the relevant lines in your Python script and a confirmation that the code works as expected.

```
1  from rwsuis import RWS
2
3  def set_p1(x, y, z):
4      input()
5      robot.set_robtargget_translation("p1", [x, y, z])
6
7
8  norbertIP = "http://152.94.160.198"
9  robot = RWS.RWS ( norbertIP )
10 robot.request_rmmp()
11
12 set_p1(100, 200, 500)
13 ( tr , rot ) = robot.get_robtargget_variables ("p1")
14 print ( f"\nTranslation for 'p1' is x = { tr [0]:.2f} , y = { tr [1]:.2f} , z = { tr [2]:.2f}" )
15 robot.set_rapid_variable( "WPW", 1 )
```

```
Translation for 'p1' is x = 100.00 , y = 200.00 , z = 500.00
```

Name	Value	Data Type
p1:	[[100,200,500],[0,1,0,0]]...robtarg	
trans:	[100,200,500]	pos
x :=	100	num
y :=	200	num
z :=	500	num

- 4) Use the IDS camera on Norberts gripper 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.



500 mm

- 5) 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.

To get the coordinates for the pucks we have used these functions:

```
def take_picture():
    arr = [0]
    while len(arr) < 20:
        arr = ueye.get_data(pcMem, width, height, bpp, pitch, False)
    frame = np.reshape(arr, (height.value, width.value, bpp.value//8))

    return frame

def find_pucks(image):
    frame = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, frame = cv2.threshold(frame, 90, 255, cv2.THRESH_BINARY)
    return decode(frame)
```

If we print the results:

```
Pucks detected:

1. **Puck #3**
- Type: QRCODE
- Position:
  - Left: 634, Top: 309
  - Width: 196, Height: 198
- Polygon (Coordinates):
  - Point 1: (634, 496)
  - Point 2: (821, 507)
  - Point 3: (830, 318)
  - Point 4: (639, 309)
- Quality: 1
- Orientation: DOWN

2. **Puck #1**
- Type: QRCODE
- Position:
  - Left: 1954, Top: 1245
  - Width: 198, Height: 193
- Polygon (Coordinates):
  - Point 1: (1954, 1249)
  - Point 2: (1959, 1438)
  - Point 3: (2152, 1437)
  - Point 4: (2149, 1245)
- Quality: 1
- Orientation: DOWN

3. **Puck #2**
- Type: QRCODE
- Position:
  - Left: 1244, Top: 775
  - Width: 190, Height: 190
- Polygon (Coordinates):
  - Point 1: (1244, 961)
  - Point 2: (1432, 965)
  - Point 3: (1434, 778)
  - Point 4: (1248, 775)
- Quality: 1
- Orientation: DOWN
```

- 6) Having fixed the values for the camera tool 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 robot to place one puck in the origin of the table coordinate system, and perhaps also some pucks in some other known 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 indicates the height of the 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 $T(500)$.

We have calculated $T(500)$ with this function:

```
def calculate_T():
    # Coordinates of the center of the table in pixels
    x_center, y_center = 1320, 1162

    # Image points (in pixels)
    P_image = np.array([
        [1200, 1446, 1200], # X
        [1195, 1162, 1129], # Y
        [1, 1, 1]
    ])

    # Table points (in mm)
    P_table = np.array([
        [7.09, 0.0, -7.09], # X
        [26.0, -26.0, 26.0], # Y
        [1, 1, 1]
    ])

    T = P_table @ np.linalg.inv(P_image)
    print('Matrix T:\n', T)
    return T
```



```
Matrix T:
[[-1.41052975e-16  2.14848485e-01 -2.49653939e+02]
 [-2.11382114e-01  1.07552856e-16  2.79658537e+02]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

- 7) Place one puck on the table somewhere. Make Python code that moves the robot to positions where images are to be taken from, capture the image and locate the puck. Then the Python code should tell the robot where to pick up the puck and move it to the center of the table. 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.

```
def get_puck_position(x_image, y_image, T):  
    image_point = np.array([x_image, y_image, 1])  
    table_point = np.dot(T, image_point)  
    x_table = table_point[0] / table_point[2]  
    y_table = table_point[1] / table_point[2]  
  
    return x_table, y_table
```

Confirmation:

```
T_ROB1-> Task02: Move the puck to the center  
T_ROB1-> Moving the puck...  
T_ROB1-> The puck is in the center of the table  
T_ROB1-> Robot waits for Python to set WPW
```

Clear	Don't Show Logs	Don't Show Task Name
 T_ROB1 E458Grip...		ROB_1 1/3 

- 8) Place several pucks on the table somewhere. Now stack the pucks in the center of the table. An optional feature is to align the orientation of the pucks when stacking, see Figure 2. 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.

First, we take a photo at a height of 500mm and get the positions of the pucks

```
T = calculate_T()
robot.request_rmp()

input('Permiso para tomar la foto')

# move to overview
set_p1(0, 0, 500)
move_to_point_a()

# take picture, find puck
img500 = take_picture()
cv2.imwrite('A5_images/puck_image_500mm.jpg', img500)

pucks = find_pucks(img500)
numPucks = len(pucks)

if len(pucks) < 1:
    print("ERROR: no pucks in image")
    exit(-1)

print("Pucks detected with their center coordinates:\n")
image_points = []

# For every puck, get position
for i, obj in enumerate(pucks, start=1):
    data = obj.data.decode("utf-8")

    print(f"{i}. **{data}**")

    puck_x, puck_y = get_center_of_puck(obj)
    puck_matrix = np.array([puck_x, puck_y, 1])

    x_fisico, y_fisico = get_puck_position(puck_x, puck_y, T)
    print('Position: (', -x_fisico, y_fisico, ')')
    image_points.append((-x_fisico, y_fisico))
```

```
1. **Puck #2**
Position: ( 2.421342424242523 -146.32054878048783 )
2. **Puck #5**
Position: ( -3.854381818181947 206.23901788617877 )
3. **Puck #7**
Position: ( 102.6352696969696 102.15148983739822 )
4. **Puck #8**
Position: ( 101.59755151515155 -87.42806504065052 )
```

Then, we send a signal to the RAPID with stack_pucks (WPW = 2) and tell it the positions of the pucks.

```
robot.request_rmp()
input('Permission to process pucks')

stack_pucks()
for x,y in image_points:
    wait_for_rapid()
    print('Processing Puck')
    set_p1(x,y,0)
    robot.set_rapid_variable('image_processed', 'TRUE')

robot.cancel_rmp()

ueye.is_ExitCamera(hCam)
finish_rapid()
```

Confirmation:

```
T_ROB1-> Task03: Move all pucks to the center
T_ROB1->Successfully moved puck number 1
T_ROB1->Successfully moved puck number 2
T_ROB1->Successfully moved puck number 3
T_ROB1->Successfully moved puck number 4
T_ROB1->All pucks are in the center of the table
T_ROB1-> Robot waits for Python to set WPW
```

Clear

Don't Show
Logs

Don't Show
Task Name

T_ROB1
E458Grip...

ROB_1
1/3

Working hours

Date	Time	Student working
October 24th	LAB: 10:15 - 15:15	Anton & Daniel
October 25th	LAB: 9:00 - 14:00	Anton & Daniel
November 7th	LAB: 9:30 - 14:00	Anton & Daniel
November 8th	LAB: 9:30 - 14:00	Anton & Daniel
November 11th	LAB: 16:30 - 18:30	Anton & Daniel
November 12th	LAB: 17:00 - 19:00	Anton & Daniel
November 14th	LAB: 17:00 - 19:00	Anton & Daniel
November 15th	LAB: 17:00 - 19:00	Anton & Daniel
November 19th	LAB: 11:30 - 13:30	Anton & Daniel
November 21th	LAB: 17:00 - 19:00	Anton & Daniel
November 22th	LAB: 17:00 - 19:00	Anton & Daniel
November 26th	LAB: 11:00 - 13:00	Anton & Daniel
November 27th	LAB: 11:00 - 12:00	Anton & Daniel

Total hours = 40.