Project description and guidelines

IN 2022 Robotics, BSc Course, 2nd Sem., Dr. Julien Nembrini, Mikkeline Elleby

Handout on Thursday April 25 2023 Week 8 to 14 Due on **Monday May 29 2023, midnight**

Presentation on **Thursday June 1st 2023**

The remaining part of the semester will be devoted to a group project which consists of a *task* and a *challenge*. The challenge can be freely defined by the students but must be presented to the teaching team for approval before starting work on it.

Important dates :

April 27 Final approval on challenge description

May 29, midnight Hard deadline for submitting the final project (incl. presentation draft)

June 1st, 13h15 Final presentation

The techniques exercised during the first part of the semester are sufficient in combination to solve the task proposed. All behaviors needed are variants of Explorer/Lover behaviors, wall following or object detection controllers. However, it is allowed to use other techniques if wished. As during the first part of the semester, all robots should use the exact same code (unless especially agreed for the challenge).

Grade The final project is evaluated on the basis of :

- the task code/algorithm (25%)
- the challenge code/algorithm (25%)
- the report (30%)
- the video (10%)
- the final presentation (10%)

Important elements include :

- robustness and efficiency of the algorithm
- clarity and readability of the code
- structure and clarity of the report
- orthography and grammar

Hand in :

Upload your code, report, video and presentation (in PDF format) in a zip file on Moodle before May 29, (midnight). There will be no submission accepted beyond this deadline.

You may amend your presentation between the deadline and June 1st, but in this case **you must resubmit it in PDF format per email before 12h00 on June 1st**.

Zip files not following the naming convention will not be considered.

Task

The task consists in defining a test run for a robot. Using a fixed arena setup, it will test in sequence the quality of :

1. the camera using object recognition.
2. the motor control by traveling in straight line.
3. the ground sensors by crossing a line
4. the proximity sensors by turning on the spot near an obstacle.
5. the sensori-motor coupling by following a wall.
6. one additional sensor

For each of these steps a performance plot will be produced at the end of the run, which allows to differentiate the robots according to their overall quality.

Environment

In the task, the robot arena, through which the e-pucks navigate, is of squared shape of dimension 60 by 60 centimeters, with blocks on its edge (see Figure 1). 4 colored blocks of a different color (black, red, blue and green) are positioned in the middle of each edge of the arena, the green block being thinner. On the green block edge, the blocks are perfectly aligned and both ends of this edge are right-angled corners. Two lines (black and violet) are located perpendicularly on the axis going from the center of the arena to the green block.

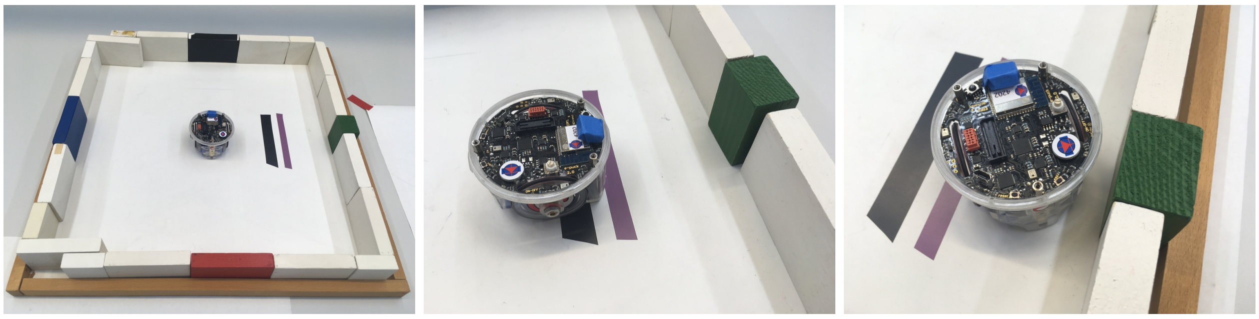

FIGURE 1 – Task arena with robot at starting position (left), robot in phase 2 “forward” with blue weight on the front (middle), robot in phase 3 “turning” close to the wall (right)

At the beginning of the test, the robot starts in the center facing the black block, as in Figure 1. The robot is weighted in the front to prevent from oscillating (using the blue weight on Figure 1). The robot then goes through the following phases (in order) :

Phase 1 : Turn on the spot and detecting blocks. Stop in front of the green block after two turns.

Phase 2 : Move forward towards the green block and stop when detecting an obstacle in front.

Phase 3 : Turn on the spot while recording proximity sensor values (two turns).

Phase 4 : Follow the wall on its right until the green edge corner, turn around and follow the wall on its left until the opposite corner, and stops.

An additional sensor must be tested, either during one of these 4 phases or in a new one.

Phase 1, "Detection"

Task to achieve : turn on the spot while recording object detections, and stop in front of the green block.

1. The robot starts in front of the black block
2. The robot makes 1 and 3/4 turn and stops as precisely as possible in front of the green block
3. The robot records all object detections

At the end of the test, box plots of all `x_center` positions and widths for each block type are produced (see Figure 2).

Phase 2, "Forward"

Task to achieve : move forwardwhile recording green block detections and ground sensor values, until reaching an obstacle in front.

1. The robot moves forward at low speed without direction steering and crosses the 2 lines on the ground
2. When it detects an obstacle in front, the robot stops at about 1-2 cm from it
3. The robot records all green block detections and ground sensors values.

At the end of the test, a box plot of all green block `x_center` positions is produced, together with a scatter plot correlating detected green block widths and `x_center` positions to evaluate how linear the relationship is (see Figure 3).

Phase 3, "Turning"

Task to achieve : turn on the spot while recording proximity distances and stop with the obstacle on the right of the robot.

1. The robot turns on the spot
2. The robot makes 2 and 1/4 turns and stops as precisely as possible with the obstacle on its right
3. The robot records all proximity sensor values.

At the end of the test, box plots of all IR proximity sensor values are produced (without values produced when no obstacle is present), together with a line plot of the same values (see Figure 4).

Phase 4, "Wall follow"

Task to achieve : follow the wall on the right-hand side until reaching the corner and turn back to follow the same wall on the left-hand side to the next corner.

1. The robot follows the wall on its right until reaching an obstacle in front
2. The robot turns on the spot and stops as precisely as possible with the wall on its left
3. The robot follows the wall on its left and stops when reaching an obstacle in front

At the end of the test, box plots of the relevant IR proximity and `ds` values are produced, together with line plots of the same values (see Figure 5).

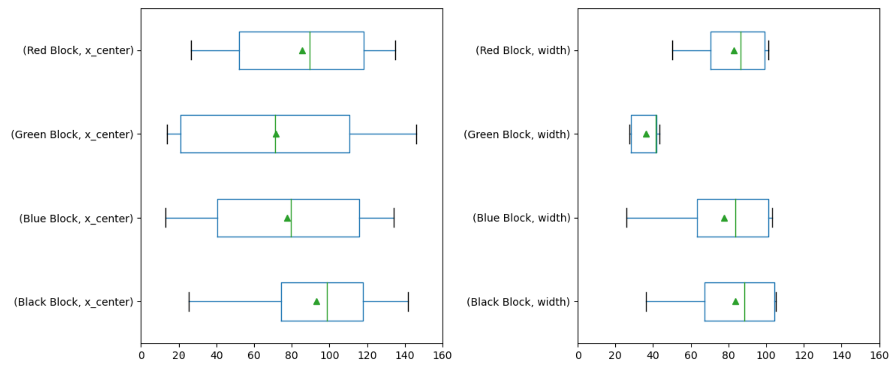

FIGURE 2 – Example detection phase analysis graphs

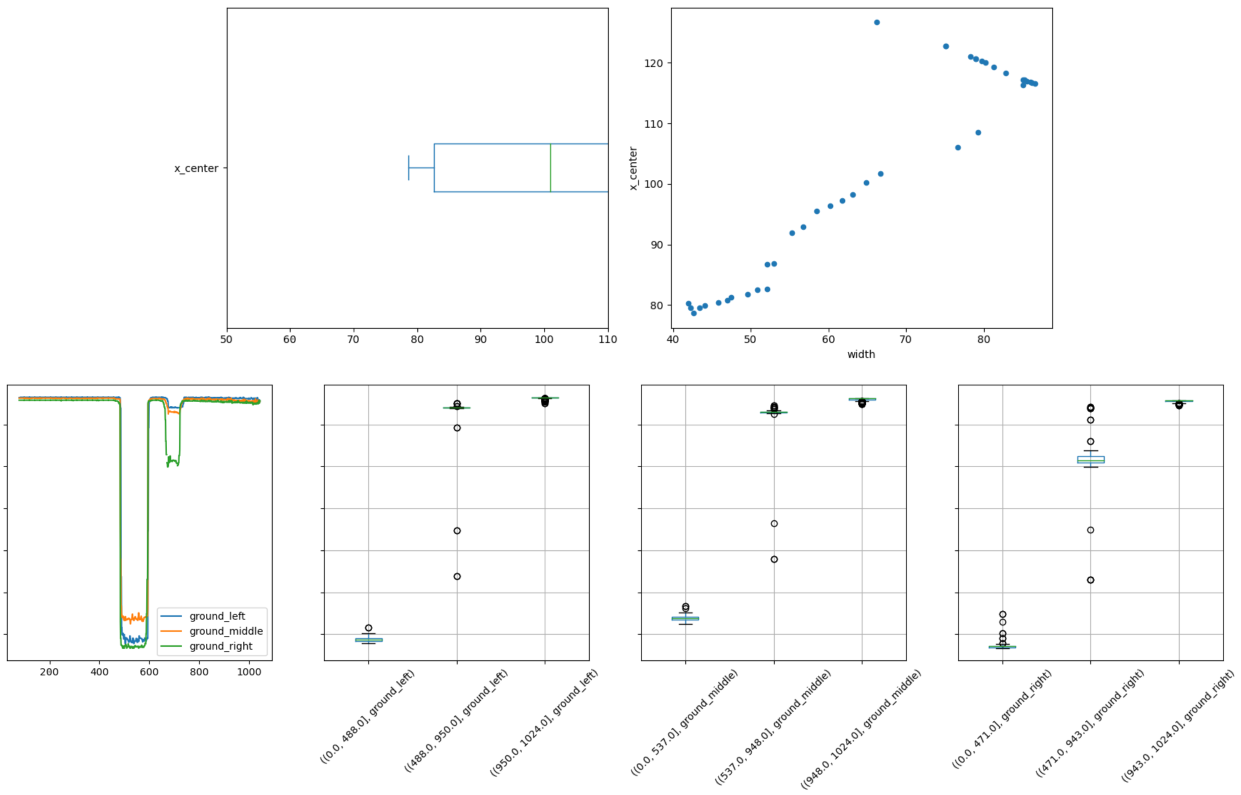

FIGURE 3 – Example forward phase analysis graphs

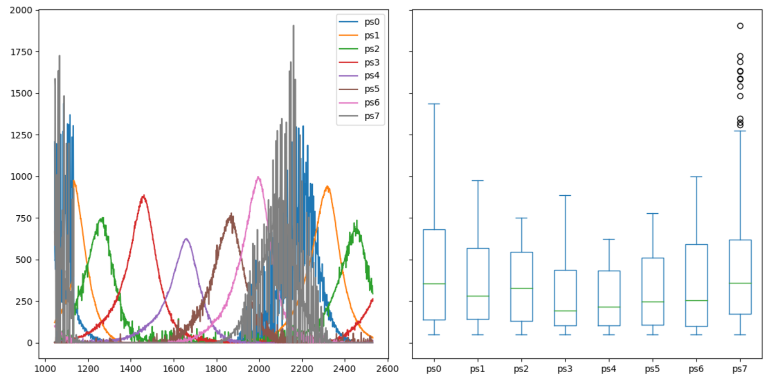

FIGURE 4 – Example turning phase analysis graphs

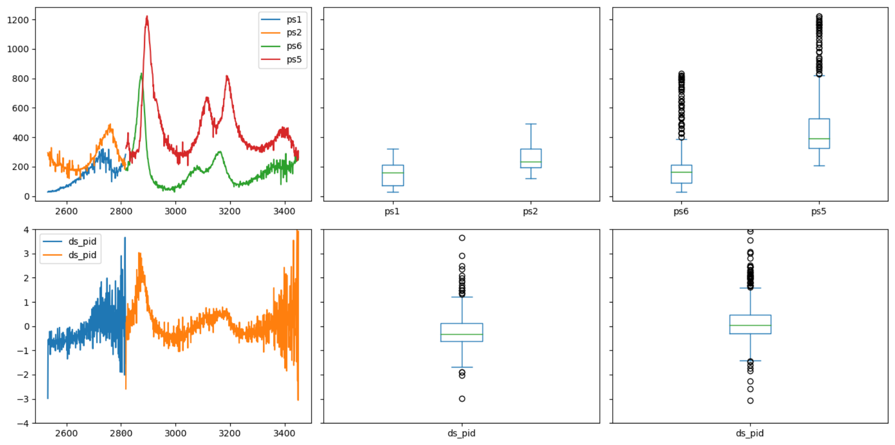

FIGURE 5 – Example wall follow phase analysis graphs

Graph production

Additional sensor testing

Choosing between the microphones or the time-of-flight, an additional sensor type must be tested, either during the described 4 phases or by adding another phase to the task. At the end of the test, an analysis of the performance of this sensor type must be produced in the form of a graph.

A python script is provided to produce the graphs of Figures 2 to 5. The graphs for the additional sensor type must be developed by the group.

Robot comparison

When the test run controller is complete, a comparison of 3 robots is conducted to display performance differences. The controller should be robust to robot differences. This comparison is discussed in the final report. All robots should be tested using the exact same code.

Challenge

The challenge consists of defining, designing and solving an additional challenging task.

The challenge can be freely defined by the groups, but must be described in a short email (5 to 6 sentences at most). This idea will be discussed with the teaching team to get approval before starting work on it. A final discussion on project proposals will take place on **April 27**. Projects involving multiple robots are encouraged. Possible challenges are :

— autonomous “cars”
— use object recognition API
— neural network to improve robot behaviour
— communicate with other media (sound, light, etc)
— **your proposal**

Each group will have to choose a different challenge. If two groups choose something similar, variants that are sufficiently different will have to be proposed. Alternatively, multiple teams working on similar projects that involve competitive robots may test their controllers against other groups.

Guidelines

You are free to choose which strategy you implement for solving the task and the challenge, but keep the following guidelines in mind :

— You are expected to discuss the problem, including coding and testing, in group of three persons. Please inform the teaching team if you wish to change group before starting the project.

— **Report :** each student has to write her/his own report in Latex, maximum 12 pages + appendices (including this document as Appendix A). The report should present the project (task + challenge) as a whole. An empty template is provided on moodle

— **Code :** one code per group. We ask you to hand in a modular, readable and commented code. Your code will be thoroughly read.

— **Video :** one video per group, max 4 minutes. An uninterrupted and accelerated shot of the whole task process (all phases, one robot) must be included. Any acceleration must be annotated in the video and further annotations for clarity are encouraged. LED patterns should be visible.

— **Presentation :** each student has to do her/his own presentation (max. 4 minutes) focusing on a particularly interesting aspect of the project (task or challenge), different for each student. One presentation file per group is encouraged.

— Navigating the environment can be done using the lover/explorer behaviors, wall follower and object recognition controller (and possibly others). Be careful to the robustness of the controllers you may choose to use.

— The task controller difficulty is in making robust transitions. It needs to be able to cope with malfunctioning robots, e.g. a robot without exploitable camera images. A prompt asking a testing person to reposition the robot is possible.

— Make your solution as robust (catch error situations) and efficient (achieving the task in a short time) as possible.

— Document your code (events, methods, algorithms, etc.), draw a FSM of your controller.

— Use the LED patterns described in the appendix in order to indicate to an observer in which phases the robots are. Add the necessary custom patterns to explain your code, and document them using the same tabular presentation as in the appendix.

Appendix : LED patterns

The following is a list of the LED patterns that must be used. For each pattern, the indexes of the LEDs are given. The LEDs have to be turned on long enough to be visible in the video.

LED position	Corresponding action
Red LED 0	Phase 1 : ON
RGB LED 7	Phase 1 : Shows the color of the detected block (white if black)
Red LED 4	Phase 2 : ON
RGB LED 7	Phase 2 : Shows green if green block detected
Red LEDs 2/6	Phase 3 : ON
Red LEDs 2 or 6	Phase 4 : ON on the side of the wall follower
Red LEDs 0/2/6	Phase for additional sensor (if applicable)
All Red LEDs (0/2/4/6) + body LED	Phase 4 : Task completed
Specific to your code	Add to better explain your code

Appendix : Example code for recording sensor values

Below is a code snippet to record your sensor/detection data in a format compatible with the provided python script to produce the graphs in Figures 2 to 5. The different states are `DETECTION`, `FORWARD`, `TURNING`, `WALLFOLLOWER_RIGHT`, `WALLFOLLOWER_LEFT`.

Note that an additional sensor requires modifying this code.

```
#open file for writing
data = open("task.csv", "w")

if data == None:
    print('Error opening data file!\n')
    quit

#write header in CSV file
data.write('step,state,x_center,y_center,width,height,conf,label,ground_left,ground_middle,ground_right,
     ds_pid,ps0,ps1,ps2,ps3,ps4,ps5,ps6,ps7\n')


def logDETECT(f,state,step,item):
    line = [step,state]
    line.extend([item.x_center,item.y_center,item.width,item.height,item.confidence,item.label])
    line.extend([np.nan] * 12)
    f.write(', '.join(map(str, line))+'\n')


def logGROUND(f,state,step,gs):
    line = [step,state]
    line.extend([np.nan] * 6)
    line.extend([gs[0],gs[1],gs[2]])
    line.extend([np.nan] * 9)
    f.write(', '.join(map(str, line))+'\n')

def logPROX(f,state,step,ps):
    line = [step,state]
    line.extend([np.nan] * 10)
    line.extend([ps[0],ps[1],ps[2],ps[3],ps[4],ps[5],ps[6],ps[7]])
    f.write(', '.join(map(str, line))+'\n')


def logPID(f,state,step,ds,ps):
    line = [step,state]
    line.extend([np.nan] * 9)
    line.extend([ds,ps[0],ps[1],ps[2],ps[3],ps[4],ps[5],ps[6],ps[7]])
    f.write(', '.join(map(str, line))+'\n')


# example use (where stepcounter counts all iterations since the controller start)
for item in detections:
    logDETECT(data,"DETECTION",stepcounter,item)
```