Robotics Project II

Marco B. Gabriel, Group 12

May 24, 2021
IN.2022 Robotics 2021, BSc Course, 2nd Sem.
University of Fribourg
marco.gabriel@unifr.ch

Abstract

This report covers the second project of the Robotics Course, which consists of two parts. The first part is a given task, in which three e-puck robots have to find and hide behind colored blocks, before gathering in the center knowing whether their hideout was exposed or not. This can be solved using the behaviours developed in project I. The second part is implementing the game 'tag' with up to four robots, including an improved color-follower behaviour that can be used for both the catcher and the runners. We succeeded in implementing both parts, but especially the last phase of the given task offers room for improvements.

Keywords: e-puck, robot, robots, behaviour, communication, game tag, tag, color recognition, exposed, hidden, catcher, runner, game, color-follower, robotics

Contents

Chapter 1

Introduction

Robots are machines, that use sensors to get information about their environment and algorithms to react autonomously to their environment in a mechanical manner. They have been used for research and to automate work in industrial processes for decades now, are still used today, and probably also in the future.

The Robotics Course introduces its students to various important topics in the research field and industry of robotics. This second project requires us to work independently and as a group to solve a given task and a challenge chosen by ourselves. The task is designed to be solvable using a combination of the already discussed techniques of the first report [2]. The task consists of three robots finding a colored corner and hiding in it, then moving to the center of the arena where they either circle around a line or stop inside, depending on whether their hiding spot was hidden or exposed.

As a challenge, we decided to implement the well-known game called *tag*, where there is one catcher, whose duty it is to catch one of the runners. The runners also need to behave in an elaborate manner, not just randomly moving around. A catched runner converts to being a catcher, whereas the catcher turns into a runner. The game includes a *power-up*, namely a speed boost for the runners. The main goal is to achieve a solution that leads to an interesting unfolding of the game.

Our solution for the task consists of the previously established behaviours line-follower, wall-follower, color-recognition and a color-follower, which is a slight modification of the line-follower. Our final solution is satisfactory, however, the last part could be implemented in a more robust way. The positioning of the corners needs to be more or less consistent for our last part to work. All other parts are completely functional and robust to a sufficient degree.

For the challenge, we made use of color-recognition, color-follower and Braitenberg behaviours, but modified them to a significant extent. We changed the color handling to not use letterboxes disguised as sensors, but count the number of colored pixels and calculate their mean coordinate, to find out where to drive. Additionally, we added black as a color to allow for four robots in the game at once. The implementation turned out pleasing, with especially the color-follower making a good figure. We only see potential to slightly improve the catcher behaviour as a whole, and to come up with a more intelligent runner behaviour. The problem with the current runner is, that it loses sight of the catcher by design, therefore limiting the amount of useful information it can gather about its environment.

Chapter 2

Task

2.1 Problem statement

The task's environment consist of a closed black line in the center of the robot arena, which measures 160 x 160 centimeters. Outside of the black shape, there are colored blocks forming a right angle. These constructions will be called *corners*. The corners have the colors red, green and blue, and every corner can be oriented in two ways. It's opening, the 90-degree-angle, can either point towards the center of the arena and therefore also towards the other corners, or away from it.

The task requires three robots to be controlled in the arena. Each robot has to hide in one of the corners, but each corner can only be occupied by one robot. The robots then have to find out whether they are exposed, e.g. they can see another colored corner from where they are hiding, or hidden. How these sates might look like is shown in Figure 2.1.

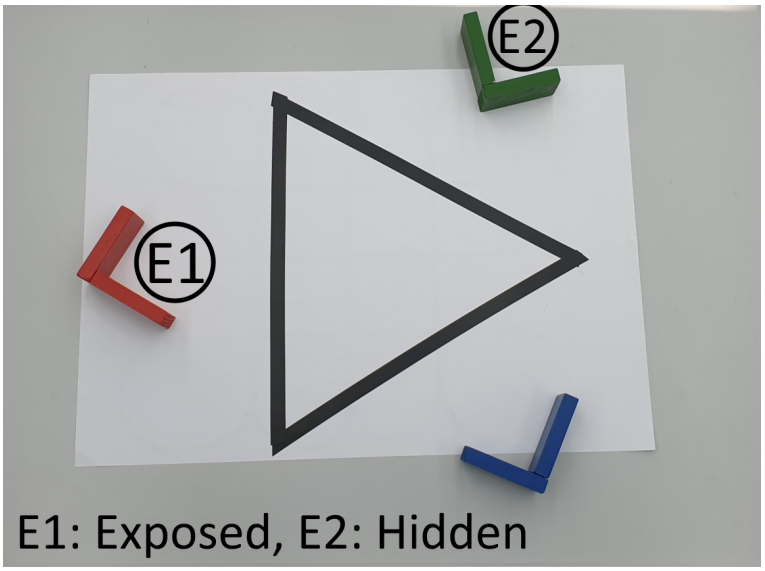

Figure 2.1: The robot arena with a visualisation of hidden and exposed robots

Hidden robots should then proceed to navigate inside of the closed line and stop there, whereas exposed robots should follow the line in a clockwise manner. Once all robots reached either state, they must simultaneously decide that the task is finished and signal this with an appropriate LED pattern. Specified LED patterns also have to be used throughout the task to indicate different states of the robots.

2.2 Solution Strategy

As suggested in the project description, which can be read in Appendix A, we split the task into different phases and sub-phases. We then implemented one (sub)phase after the other and added them to the controller. We tested the controller after every significant addition and depending on the test's outcome we corrected and improved our previous implementations or continued with the next sub-phase.

In phase 1, the robots first turn on the spot until they detect a colored corner. This is done by a function turning the robot around and analysing the colors of the center pixels of periodically taken images. Once it detects a color, each robot announces the detected color to the others, to avoid multiple robots selecting the same corner. This means that the robots need to listen for messages and keep in memory, which colors are already taken, and which ones are still available. Before selecting a color that it detects, the robot has to check if the color is still available. When it successfully detects and selects a color, it drives towards the selected corner. A simple version of this behavior can be achieved by slightly modifying the line following behavior of the previous project[2] to use the robot's camera instead of it's ground sensors. After arriving at the corner, the robot follows it using a PID-wall-follower, also from the previous project. To detect, when it reached the angle, we have to use the proximity sensors.

Phase 2 consists of the robot scanning it's environment and deciding whether it is hidden or exposed. For this, we can use the same color detection function from phase 1 with the addition of not detecting the color of the own corner.

For phase 3, we need the robot to circle around the line, if it was exposed. We accomplish this by letting the robot driving straight to the color that it saw, until it reaches the line. Given the positions of the corners, it is guaranteed to come across the line sooner or later. Once on the line, a derivation of the already established line-follower of project 1 can be used. Only a minor change is needed to make it only move along the line in a clockwise manner. If it was hidden, the robot follows the wall for a predefined amount of steps. This duration is an approximation of the time it takes the robot to move not quite to the closest point of the corner to the line. After this, it executes a curve away from the wall it was following, until it senses the black line. After reaching the black line, it drives in a straight line for a short number of steps and stops, so it should be far enough inside the line, given that the corner was placed approximately in the middle of a side of the shape created by the black line.

2.3 Implementation

To implement the task, we use a main file with a controller function, which handles switching between the different phases, the communication between the robots and getting an image from the camera. The concrete behaviours are located in other files for better clarity.

For the communication, two predefined variants of messages are used. One is the *FIN_MSG*, which a robot sends when it is finished, e.g. circling around or stopped inside the black line. The controller keeps track of the number of *FIN_MSG* it receives, shown on line 1 in Figure 2.2. If the message is a color, it means that one of the other robots has seen and selected the received color. The received color's state is then set to False in the array of available colors on line 2. In the unfortunate event of two robots selecting the same color at the exact same moment, both robots would claim the color before they receive the other robot's message. This is prevented with a simple solution: If the received color is the same as the own color, a collision must have happened. This is resolved by both robots resetting the color's status to available (True) and waiting a random amount of steps before searching for a color again. This resembles a simplified version of a back-off algorithm also used in computer networks.

```
c2i = {'r': 0, 'g':1, 'b':2} # dict to translate colors to indexes
available_colors = [True, True, True] # array of the available colors
# ... inside the robot.go_on() loop:
if msg == FIN_MSG: number_finished += 1 # a robot finished
elif msg in c2i: available_colors[c2i[msg]]=False # a color got taken
elif msg == own_corner_color: # other robot took the same color
    STATE = FIND_COLOR # go back to searching for a color
    available_colors[c2i[msg]]=True # reset status of color
    collision_delay = random.randint(0, 10) # back-off delay
```

Figure 2.2: Code for the communication in the basic task

Getting the current image from the robot's camera is also implemented in the main controller function. This simplifies the handling of the image data. If an invoked function needs the image, the red, green and blue arrays are passed as arguments. This design has the advantage, that getting the image is only done in one place of the code and only once at most per iteration, making it easier to turn off the camera or to limit the frequency of updating the current image data. This part is implemented as visible in Figure 2.3

```
i += 1
if i%5==0 and CAM_ON: # limiting the frequency of updating the camera
    robot.init_camera(".")
    red,green,blue = robot.get_camera() # storing image for later use
if not CAM_ON: # turning camera off if not needed
    robot.disable_camera()
```

Figure 2.3: Code for the camera in the basic task

The functionality of detecting colors is needed in different situations. Therefore we implemented a customiseable function to do so. It consists of two parts: The color detection on a pixel level, and the color detection in a part of an image. The former works as shown in Figure 2.4 by checking the numerical differences between the color channels to distinguish grey tones from colors, and in the case of it being a color, returns the color with the highest value. To reduce false positives when the robot is close to an object, specifically when it is in the angle checking whether it is exposed, there are lower bounds in place to prevent very dark colors from counting as colors. We implemented this limitation, because otherwise the robots sometimes reported to be exposed while the camera faced into the angle of the colored corner.

```
seen_color = "" # is considered as False (e.g. no color)
if abs(int(r)-int(g))<threshold_grey: wv += 1
if abs(int(g)-int(b))<threshold_grey: wv += 1
if abs(int(b)-int(r))<threshold_grey: wv += 1
if wv > 1: return seen_color # color is grey-ish
elif (r>g and r>b and r>80): seen_color = "r"
elif (g>r and g>b and g>70): seen_color = "g"
elif (b>r and b>g and b>70): seen_color = "b"
return seen_color
```

Figure 2.4: Code for the color detection on pixel level in the basic task

The color detection on a part of the image, for example a letterbox, is done using an averaging function which is shown in Appendix B, Figure 4.2. The function returns the average of all pixels inside a given rectangle, which can then be passed to the previously shown function which only works on single pixels, since the average can be considered as just one pixel.

After detecting a color in phase 1, a color-follower behaviour is used to move the robot towards the colored corner. The color-follower is comprised of the line-follower from the previous report[2] and a small modification replacing the ground sensors with the camera as shown in Figure 2.5. Instead of using the three ground sensors, the current image from the camera is divided into three rectangles, left, right and center. The robot will therefore always curve towards the side on which it sees the color, or drive straight if it sees the color only in the middle or in all areas, resulting in a robust color-follower.

```
color_box_left = pixel_average((20, 50, 40, 60), r, g, b)
color_box_right = pixel_average((110, 140, 40, 60), r, g, b)
color_box_center = pixel_average((70, 90, 40, 60), r, g, b)
# instead of
gs_values = r.get_ground() # = [ground_left, ground_center, ground_right]
```

Figure 2.5: Modification to turn a line-follower into a color-follower in the basic task

Once close enough, the wall-follower of the project 1 [2] is used, until the robot detects itself to be in the angle. The angle-detection works by checking the values of multiple proximity sensors. It is shown in Figure 2.7. We use a counter to only detect the angle after a specified time, to prevent the robot from thinking it is in the angle at the moment when it first encounters the corner. At this moment, the robot is not yet aligned correctly relative to the wall, which can lead to false positives. For detecting an angle, we either use the front sensors angled away from the wall (sensor 1 or 6), or a combination of a forward facing sensor (0 and 7) with a sensor facing the wall at the backside (3 and 4). Potential scenarios are shown in Figure 2.6.

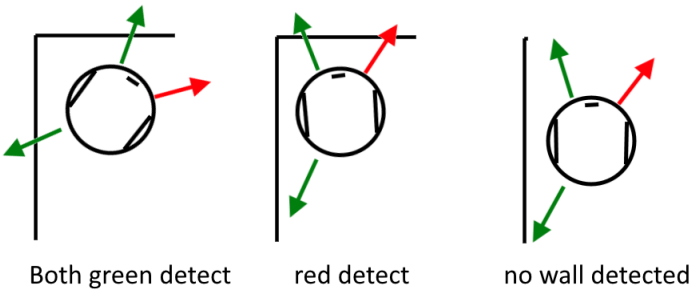

Figure 2.6: Different situations to detect an angle

```
corner_counter += 1
if corner_counter > 300:
    if (side == left and ((ps[1]>70) or ((ps[7]>70) and ps[4]>70))): RETURN = CORNER
    if (side == right and ((ps[6]>70) or ((ps[0]>70) and ps[3]>70))): RETURN = CORNER
```

Figure 2.7: Angle-detection in the basic task

Arriving in the angle, the robot now turns around itself to check whether it sees another color than its own, which would mean that it is exposed. If it is exposed, we immediately run the line-follower behavior, which drives straight forward before detecting a line, which means it drives towards the seen color. That way, the robot should sooner or later come across the line and follow it in a clockwise manner. We changed the line follower to only have the left ground sensor off the line and the two others on the line, to make it only drive in a clockwise manner.

Hidden robots however make it a bit harder to implement a robust behaviour. We decided to let the robots first follow the wall for a specified amount of steps, before then curving away from the wall. This curve should be guaranteed to lead the robot to the middle, but the environment has to fulfil certain conditions, which means that the behaviour is not that robust. A depiction of a scenario with a compliant environment is shown in Figure 2.8. The green stars resemble points, where the robot changes its behaviour. At the first star from the right, it stops following the wall, at the second star it moves straight instead of a curve, and it stops at the third. The robot starts to move straight after detecting the line and stops after a given amount of steps.

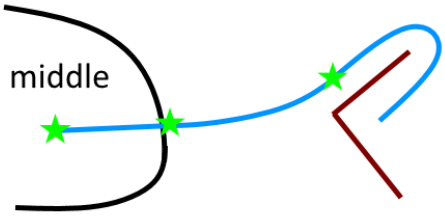

Figure 2.8: Path a hidden robot takes

Circling around the line or stopped in the middle, the robots wait for the other robots to communicate that they are finished with the task. Once the robots received the confirmation from all other robots, they switch to the finish state, indicated with two green and two yellow LEDs.

2.4 Validation

At first, we had problems getting the wall-follower to run properly. The problem was, that some of our variables were in the wrong scope. They should have been global, but they were local and therefore freshly declared in every iteration of the robot.go_on() loop. Also, we did not implement the line-follower correctly. Sometimes it neither moved for- nor backwards and did also follow the line anti-clockwise. We fixed it by hard coding the left ground sensor to be off the line.

The color detection was unreliable at first too. Robots thought they were exposed while their camera was directed into their own color. We fixed this issue by adding a minimum brightness to detect colors. The last problem that we encountered was with the communication. Robots would send multiple times that they were ready to finish, which caused some robots to enter the final state before others. Also, it was possible for exposed robots to enter the final state before having reached the line.

The solution could be improved by making the process of the hidden robots moving inside the line more robust. We had the idea to use the other colored corners as guiding landmarks to guide the robot towards the center, but the result did not satisfy our expectation, it was too unreliably. But we are confident that it is possible to implement it that way with more time to refine the code. Another improvement would be to incorporate an explorer behaviour to prevent hidden robots from crashing into each other, like they did in our video [3] at 0:43 available on https://youtu.be/IkN0H6pND9A.

But all in all the implementation turned out to work, especially the communication part that prevents multiple robots from selecting the same colored corner worked perfectly since the beginning. That part can be seen at 0:52 in the video [3].

Chapter 3

Challenge

3.1 Problem statement

The challenge is implementing the game "tag". The game consists of multiple robots, with one of them being the *catcher* and all others so called *runners*. The catcher's task is to catch an arbitrary runner, which in turn tries not to get catched. Unlike when played among humans, a runner is catched when the catcher manages to move closer to a runner than a given threshold, instead of actual touching. After catching a runner, the catcher turns immediately into a runner, whereas the catched robot turns into the catcher after a short delay. The delay prevents players from immediately catching whoever catched them, which would result in an uninteresting situation. There is the possibility for runners to get *power-ups*, in this case a speed boost, to make their situation not completely hopeless. The speed boost is given to the runners when they hear a loud noise. Both, the runner and catcher behaviour should be robust and clever, to allow for a good game. The arena should have a size which allows for an interesting unfolding of the game, with only the robots inside it.

3.2 Solution Strategy

To solve the challenge, we start with implementing the catcher behavior, because it is the essential basis of a tag game and it can be tested with non moving runners. We then continue with the communication and the runner behavior. After testing that the game including catching, running away, and changing roles works as intended, we implement the speed boost power-up.

The first step in solving the challenge is a mechanism to differentiate the robots. Every runner has to know which robot the catcher is, and the catcher needs to know which runner it catches. We allow this functionality by taping colored paper strips around the robots, so a color-follower behavior will work on robots. To let the robots know which color they are, we just define that the order of the IP addresses that we pass to the program is red, blue, green and black, with the red robot being the catcher at the start.

The robots having distinct colors enables us to use color-follower behaviour for the catcher and a color-avoidance for the runners. The color-avoidance behaviour is basically a reversal of the color-follower. The robot tries not to see a color, so when it sees one, it drives a backwards curve to get the color out of it's field of view again.

To detect when a runner is catched, we use the catcher's front proximity sensors. To prevent walls from being catched, we also check that the color seen by the camera matches the target's color. Communicating who got catched is simply done by sending the color of the catched robot. That way, the other runners also immediately know which color the new catcher has.

To determine when the runners receive a speed boost, we use the microphones and check if their values are above a given threshold.

3.3 Implementation

We first added colored paper stripes to the robots. A depiction of the robots equipped with colored paper stripes is provided in Appendix B.1 Figure 4.3. We also decided to turn on three of the RGB LEDs to show the robot's own color, to make it possible to differentiate the robots in videos taken from the bird's perspective. Furthermore, the catcher is recognisable by having a red LED in the front and the back turned on. It also shows the color of its current target as the color of its front left RGB LED.

Considering the catcher behaviour, we decided to let it use different states. First, it turns around searching for a runner. If it does not find a runner in a given interval, it changes the state to explorer for a specified amount of steps, with the intention of repositioning the robot, since it did not see any runners from the previous location. At the new location, the catcher turns around again. When it detects a runner, it switches to the color-follower state, in which it stays until it either lost the runner or catched it. A detailed version of the catcher behavior is shown in Figure 4.5 in Appendix B.1 in form of a finite state machine.

The runner detection and the color-follower is essentially done the same way. We implemented a function which analyses a given image and returns the color of most of the pixels, the number of pixels that were this color and the sum of the horizontal coordinates (relative to the center) of this color's pixels. Mentioned function is shown in Figure 3.1. We do not consider the pixels in the upper half and lower quarter of the image. The used *is_color()* function is the same as the one sown in Figure 2.4 of the basic task. However, we did add the color black to it's repertoire by inserting the code fragment shown in Figure 3.2.

```
coord_sum = {"r":0, "g": 0, "b": 0, "n":0}
n = {"r":0, "g": 0, "b": 0, "n":0} # number of pixels with the corresponding color
for x in range(60, 90): # discard upper half and lower quarter of image
    for y in range(y_range:=160):
        if pixel_color := is_color(red[x][y], green[x][y], blue[x][y]):
            coord_sum[pixel_color] += (y-y_range/2)
            n[pixel_color] += 1
most_color = max(n, key=lambda key: n[key]) # selects the color with the highest n value
return (most_color, n[most_color], coord_sum[most_color])
```

Figure 3.1: Function that does the image analysis

```
black_threshold = 75
if wv > 1: # True if color is a grey tone
    if r<black_threshold and g<black_threshold and b<black_threshold:
        return "n"
```

Figure 3.2: Code fragment to add black to the color detection

To detect whether there is a runner visible in the image, we just check if the number of pixels of a color is bigger than 800. We chose 800 as a threshold, since robots normally caused about 1800 pixels to be the right color, whereas the value for an empty arena was below 100. A reference image is shown in Figure 4.4.

For the color-follower, we adapted a Braitenberg [1] Lover behavior. We calculate the mean coordinate of the colored pixels and use this value as our sensor data. The code is shown in Figure 3.3. We square the mean coordinate and the maximum coordinate offset in order to make the robot react strongly when the coordinates are close to the maximum offset, but weakly when they are close to zero. This way, the robot does not overcorrect it's direction when the target is straight ahead, but also remains agile when the target is close to being out of sight.

```
POS = coord_sum/n # mean coordinate relative to the center
MAX_POS = 80 # maximum coordinate offset from center
prefix = 1 # prefix to prevent negativeness from being eliminated by the squaring
if POS < 0: prefix = -1
exponent = 2

ds_left = (NORM_SPEED * prefix*-(POS**exponent)) / MAX_POS**exponent
ds_right = (NORM_SPEED * prefix*POS**exponent) / MAX_POS**exponent
speed_left = NORM_SPEED - ds_left
speed_right = NORM_SPEED - ds_right
```

Figure 3.3: Code of the color-follower

The next step is the *catch()* function, which determines when a runner is catched. We rely on one of the two front proximity sensors reporting a value higher than our threshold to determine when the robots are close enough. Additionally, the most represent color in the bottom quarter of the image has to be the color that the catcher was following previously. This prevents the catcher from accidentally catching the walls. This part is quite simple, as shown in Figure 3.4.

```
ps = robot.get_calibrate_prox()
color, n, _ = get_most_color(red, green, blue, letterbox_x=(90, 120))
return (((ps[0]) >= CATCH_DISTANCE) or ((ps[7]) >= CATCH_DISTANCE)) and (target_color ==
    color) and (n > 50)
```

Figure 3.4: Code of the catch() function

To announce a successful catch to the other robots, the catcher sends the color of the catched robot to all other robots. The runners use this message to either change their own state to catcher after a given delay if the message equaled to their own color, or update their knowledge of which color is currently the catcher if the color was not equal to their own. We use this knowledge to only avoid the color of the current catcher.

The last thing we implemented was the runner behavior. It contains two possible movements, on the hand side a normal Braitenberg [1] Explorer using the proximity sensors and on the other hand a derivation of the color-follower used by the catcher. When the runner is close to a wall (or any other obstacle), it uses the Explorer behaviour. This prevents the robot from driving into walls, and it is also a simple way of moving away from the catcher, if it can be sensed by the proximity sensors. Furthermore, the Explorer is also used when the catcher cannot be detected by the camera. However, if the catcher is detected by the camera and there are no obstacles nearby, the color-avoidance behavior is executed.

```
exponent = 1
# ...
robot.set_speed(-speed_left, -speed_right)
```

Figure 3.5: Modification of the color-follower to get a color-avoidance

The modification done to the color-follower to get the color-avoidance behavior is shown in Figure 3.5. The exponent is set to 1 instead of 2, since we do not especially want the runner to drive away from the catcher in a smooth, straight line, but actually want to encourage fast turns. Obviously, the more important modification is how the calculated speeds are applied to the robot's wheels. To get the avoidance, we just multiply them by -1. The result is visualised in Figure 3.6.

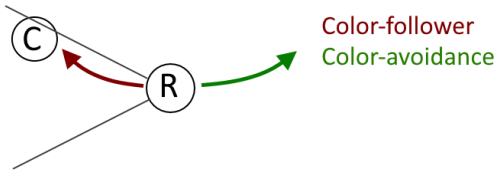

Figure 3.6: Comparison between color-follower and color-avoidance

The last thing to implement is the speed boost. We made, that the runner gets a speed boost, when the average value of all four microphones is greater than the sound threshold, which we set to 30. We chose 30, because the microphone readings when only the robot is making noise are on average consistently below 30. A graph of the measured noise levels can be seen in Figure 4.1 in Appendix A. For the concrete speed values for the runners and catcher, we used 1.5 and 3 respectively. The catcher having twice the speed of the runners leads to a fast paced game with a lot of catching situations. However, we set the speed boost for to runners to be a factor of 2.5, so when they get the speed boost, they are faster than the catcher and can actually escape.

3.4 Validation

In general, our solution turned out to work well. The catcher is capable of spotting, following and catching runners, and they are in turn capable of avoiding the runner to their possibilities. Naturally, the runners do not have much chances of actually escaping the catcher, because their speeds are intentionally set to be rather slow to allow for many catching situations. The runner behaviour might be improved by always pointing the camera at the catcher while trying to flee, to not lose sight and therefore information about where the catcher is. But since the only chance for runners to escape, is to get out of the catcher's field of view, we believe that the current implementation is at least as good, especially with the speed values that we chose. In the video [3], at 1:28, the blue robot can be seen to try to avoid the catcher, but it keeps driving forwards and backwards, because it is surrounded by a wall in the back, and the catcher in the front.

The catcher behavior is robust and efficient enough in our opinion. The only improvement which could be done in my opinion, is to also check the camera while in explorer mode. But I do think that the potential benefit is rather small. In the video [3], a nice example of the catcher behaviour can also be seen at 1:28.

We are very proud of our color-follower and are astonished by it's performance in general. However, there were situations where it did not detect the black runner or it thought it catched the black one, whereas it actually catched another color. We were able to fix the latter issue by increasing the distance for when the catching is done, because the camera records a too dark image when the robots are too close, leading to it being detected as black, even if it is not. Sometimes, the increased catching distance looks a bit too far, like at 1:40 in the video [3], where the catcher catches the blue runner from quite far away. Additionally, we made it only possible to catch the color that was being followed. We tried to improve the color recognition of black, but there are still some situations where it does not work perfectly.

The speed boost implementation does also satisfy our expectations, but in the aftermath, I would have added an LED pattern to make it easier to recognise when the robot receives the speed boost. The speed boost is demonstrated using the speakers of a mobile phone at 2:16 in the video [3].

Chapter 4

Conclusion

This report is intended to explain the second project of the Robotics Course. The project is split into two parts, the task and the challenge. For the task, we succeeded in implementing a solution which lets robots drive to colored corners, hide in them and move to the center of the arena. We made use of a variety of behaviours already covered in the report of the first project [2], such as line-follower, wall-follower and more. Additionally, we had to implement an angle-detection, color-follower and the communication. After fixing some mistakes, like a semantic error in the wall-follower, we arrived at a final version. Our final solution is capable of solving the task, although there is still room for improvements, especially the last phase can be further optimised for robustness, since ours is partially hard-coded. We mention a promising approach in section 2.4, but did not have the necessary resources to already implement it successfully.

Concerning the challenge, we succeeded in implementing a working version of the game tag with up to four robots. The course of the game is interesting, which was our main goal. Our catcher behaviour is robust and able to follow the runners, even if they try to escape. For this following behaviour, we redesigned the color-follower to use the mean coordinates instead of letterboxes to determine its path. This resulted in a smoother and more agile movement. On the other hand, our runner behaviour does actively avoid the catcher, but there might be ways to make it better. A speed boost for runners is also included, which works by exposing them to a loud noise. Further properties that could be improved are the catcher's behaviour when not detecting runners for a given duration and the detection of the color black, which is still not perfect. A small but neat improvement would be to indicate the accelerated runners with an LED pattern.

I am looking forward to learn what challenges the other groups came up with, what problems they faced and how they were able to implement their solutions. It will remain interesting if future students can benefit of our experience and how technology will change, since i can only imagine what else would be possible with an e-puck version 3.

Bibliography

[1] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1986. https://books.google.ch/books?hl=en&lr=&id=7KkUAT_q_sQC&oi=fnd&pg=PA1&dq=Vehicles+Braitenberg Last visited: 22.05.2021.

[2] Marco B. Gabriel. *Robotics Project I*. 2021. https://weebit.ch/robotics_project_1.pdf Last visited: 20.05.2021.

[3] *Video of the basic task and the challenge*. https://youtu.be/IkNOH6pND9A version 1 Last visited: 22.05.2021.

Appendix

Appendix A Experimental Results

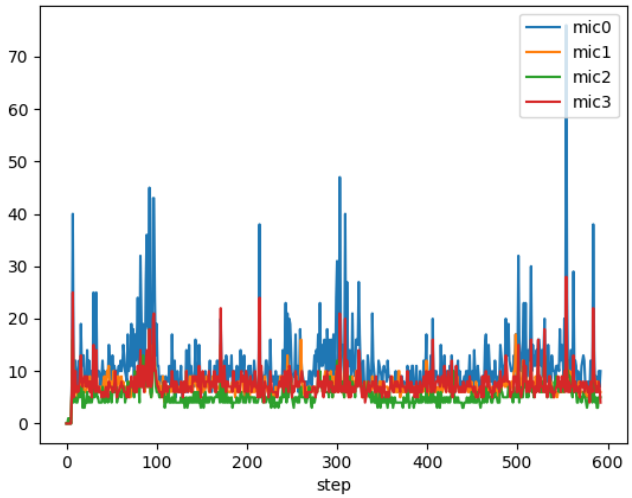

Figure 4.1: Measured noise level of only the robot moving

Project description and guidelines

IN 2022 Robotics, BSc Course, 2nd Sem., Dr. Julien Nembrini, Vincent Carrel

Handout on Thursday April 15 2021 Week 7 to 13 Due on **Monday May 24 2021, midnight**

Presentation on **Thursday May 27 2021**

The remaining part of the semester will be devoted to a group project which consists of a basic 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 20, midnight Short description of challenge idea, sent per email to the teaching team

April 22 Challenge idea discussion

April 25, midnight Final short challenge description, sent per email

May 24, midnight Hard deadline for submitting the final project (incl. presentation draft)

May 27, 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 or PID 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 project code/algorithm (25%)
- the challenge code/algorithm (25%)
- the report (30%)
- the video (10%)
- the final presentation (10%)

Important elements include :

- dynamicity, 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 24, (midnight). You may amend your presentation between the deadline and May 27, but in this case **you must resubmit it in PDF format per email before 12h00 on May 27**.

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

Basic task

The basic task consists of letting 3 robots (1) hide behind two colored blocks forming a corner (2) check if they are indeed *hidden* or *exposed* (3) if hidden : gather inside the closed line in the center ; if exposed : circle around the closed line in a clockwise manner.

Environment

In the basic task, the robot arena, through which the e-pucks navigate, has a height and width of 160x160 centimeters (see fig 1). The robots can start in any position in the arena. Colored blocks are positioned in groups of two forming a right angle corner, denoted as *corners*. There are only three corner in the arena, each of a different color (red, blue and green). The corners' position and orientation can vary. In the middle of the arena is a closed line. Corners are always located outside the line.

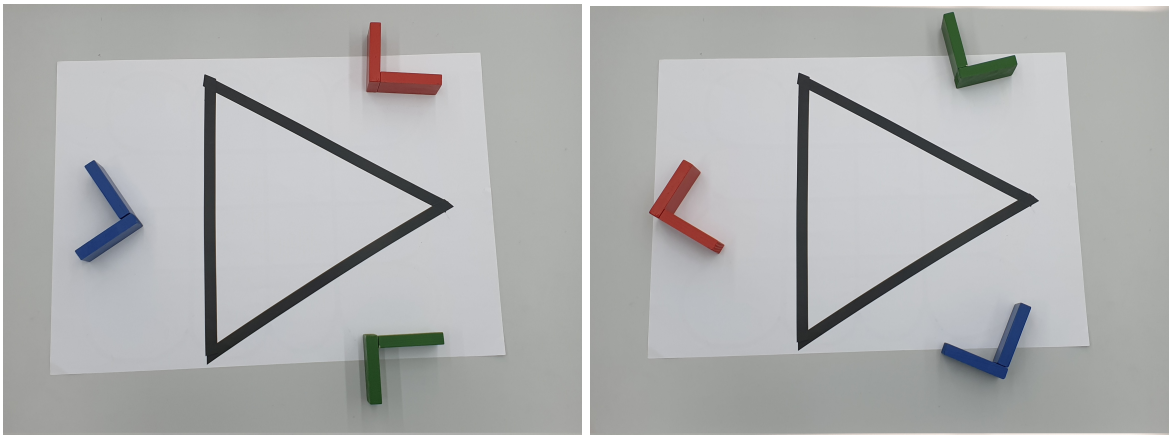

FIGURE 1 – Project arena (two possible corner configurations)

We assume that 3 robots are in the arena and that the communication is reliable. Each robot is initially placed randomly in the arena. Each robot then goes through the following phases (in order) :

— Phase 1 : find one colored corner and hide in its angle.
— Phase 2 : check if it is *hidden* or *exposed* by looking at the surrounding environment.
— Phase 3 : go inside the central closed line if hidden or circle around it if exposed.

Phase 1, Find and Hide

Task to achieve : select a colored corner, and hide in its angle.

1. Each robot selects a different colored corner (the coordination between robots may involve communication).
2. The robot moves toward the selected corner.
3. Once the corner reached, the robot goes around it and stops in its angle.

Phase 2, Observe

Task to achieve : each robot checks if its position is *hidden* or *exposed*.

1. Each robot scans its environment, without moving.
2. If it detects another colored corner, it is *exposed*. if it does not detect another corner at the end of its scan, it is *hiddden*. Only colored corners count, another robot should not be detected as a corner.

Phase 3, Gather

Task to achieve : go inside the central closed line if hidden, or circle around it if exposed.

Depending on its status (hidden/exposed), the robot navigates to :

- hidden → the inside of the central closed line.
- exposed → the central closed line and circle around it in a clockwise manner.

When all robots have reached either of these states, they must collectively and robustly decide that the task is completed and signal task completion with the appropriate LED pattern.

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) that will be sent before **April 20 midnight**. This idea will be discussed with the teaching team on **April 22** to get approval before starting work on it. A final short description will be sent per email before **April 25, midnight**. Projects involving multiple robots are encouraged. Possible challenges are :

— autonomous “cars”
— collectively solve a maze
— recognize objects/forms
— neural network to improve robot behaviour
— **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. If you are hesitating between multiple ideas, feel free to include all of them in your initial mail.

Guidelines

You are free to choose which strategy you implement for solving the basic 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 his own report in Latex, maximum 12 pages + appendices (including this document as Appendix A). The report should present the project (basic 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 basic task process (phases 1 to 3) must be included. Any acceleration must be annotated in the video and further annotations for clarity are encouraged.

— **Presentation :** each student has to do 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.

— The e-pucks need to collaborate and communicate. 3 e-pucks have to be used for the task.

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

— Do not use any hard coded behavior. The e-pucks should be able to complete the basic task even if their starting positions change or a different arena is used (all the robots basically know is that the arena is divided by a closed line within which they start and that colored blocks are disposed along this line).

— Make your solution as dynamic (independent of initial configuration), robust (catch error situations) and efficient (achieving the task in a short time) as possible.

— Document your code (events, methods, algorithms, etc.).

— Use the LED patterns described in the appendix in order to indicate to an observer what the robots are doing. If you add custom patterns, document it 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
RGB LED 7	Show the color of the block attributed to the robot
RGB LED 3 and 5	Show the block's position (BLUE if hidden, GREEN if exposed)
Red LED 0	On when the robot is following a line on the ground
Red LED 2 and 6	On when the robot is following a wall on the corresponding side
Red LED 4	On when the robot is in the corner
Choose your favorite	Task completed
To be specified	Others if necessary

Appendix B Source Code

```
for x in range(box[2], box[3]): # the box tuple is given to the function as an argument
    for y in range(box[0], box[1]):
        r_total = r_total + r[x][y] # calculating the sum of the color values
        g_total = g_total + g[x][y]
        b_total = b_total + b[x][y]
        n+=1
return (r_total/n,g_total/n,b_total/n) # returning the average color values
```

Figure 4.2: Code for the color detection on image level using a letterbox, in the basic task

B.1 Miscellaneous

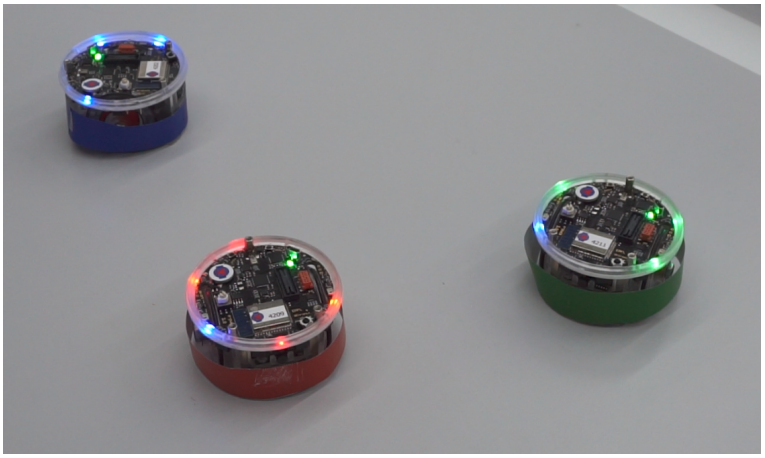

Figure 4.3: Image of the robots with colored paper stripes

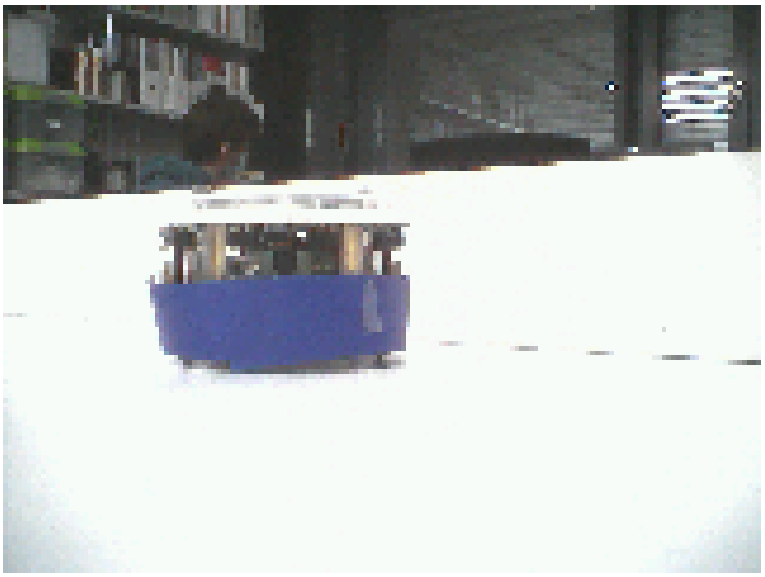

Figure 4.4: Image of a robot with a blue paper stripe from an E-Puck's perspective

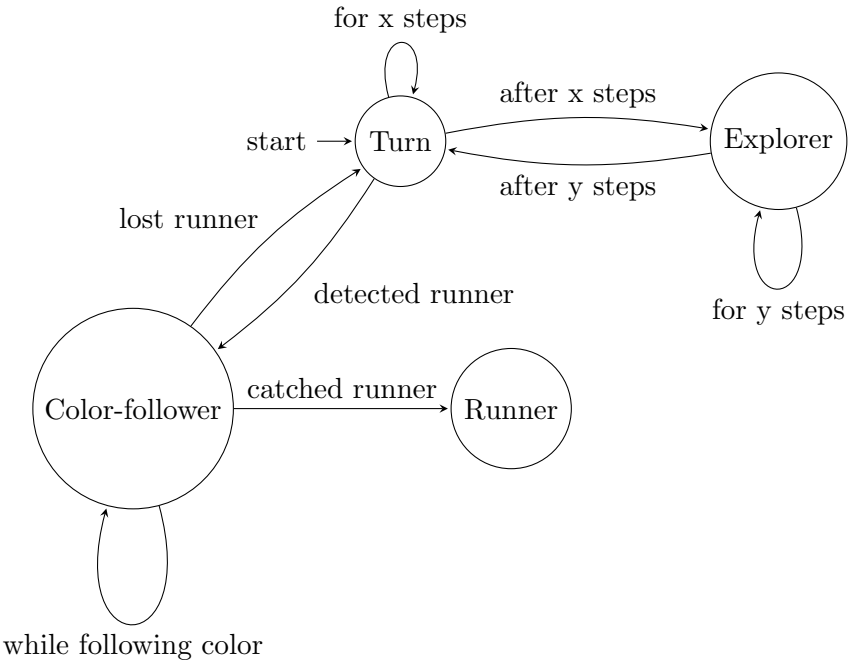

Figure 4.5: Catcher behaviour