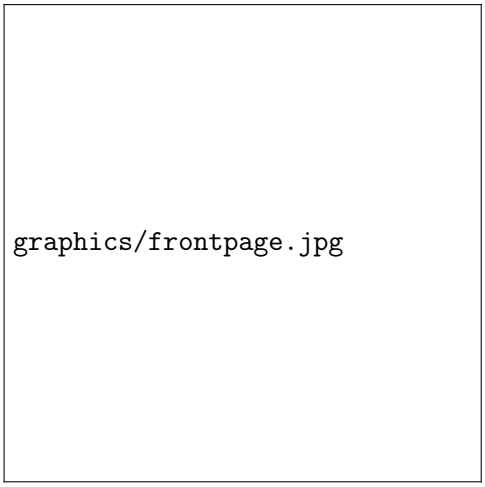

Advanced Robotics Exam Handin

Jonas Nim Røssum (jglr@itu.dk),
Gergo Gyori (gegy@itu.dk),
Ruben Oliver Jonsman (rubj@itu.dk),
Laurits Bonde Henriksen (lauh@itu.dk)

Advanced Robotics (Autumn 2024) - KSADROB1KU
IT University of Copenhagen

1st of November 2024

Abstract

This report outlines the development of a robot designed to solve ...

Contents

1 Problem: Tag

This project entailed programming a robot that was able to play tag. Our task was to develop a seeker robot and an avoider robot. In the game of tag, one robot is assigned to be the seeker and the others are avoider robots. The seeker starts in the center of the board and the avoiders in the corners of the board. In the center of the board, a zone is marked with reflective silver tape. In this zone, an avoider is safe. However, there is only space for one avoider at a time. Avoiders must leave the safe zone, if it receives a signal from another avoider.

2 Technical Requirements

1. You are free to use any approach, sensor and robot (taught/used in the course)
2. It is expected that you employ some form of robot learning or evolutionary robotics, but maybe this is not where you want to start.
3. A seeker robot has to turn its LEDs red and if in the safe zone orange.
4. An avoider robot has to turn its LEDs blue, but if it is safe green and if it is tagged purple (due to embarrassment).
5. A seeker transmits "1" and if an avoider receives this it is tagged and has to stand still.
6. An avoider transmits "2" and if an avoider in the safe zone receives this it has to leave immediately and can first after 5 seconds begin to transmit "2".
7. It is essential that you focus on how to make a robot that performs well also in practice.

3 Robot hardware

Our robot is a closed-loop, behaviour-based Unmanned Ground Vehicle (UGV), also known as a mobile robot [1]. The hardware assembly consists of a Raspberry Pi with a camera, mounted with a 3D printed mount to a Thymio robot. The system gets power from a power bank, that fits between the Raspberry Pi and the Thymio robot.

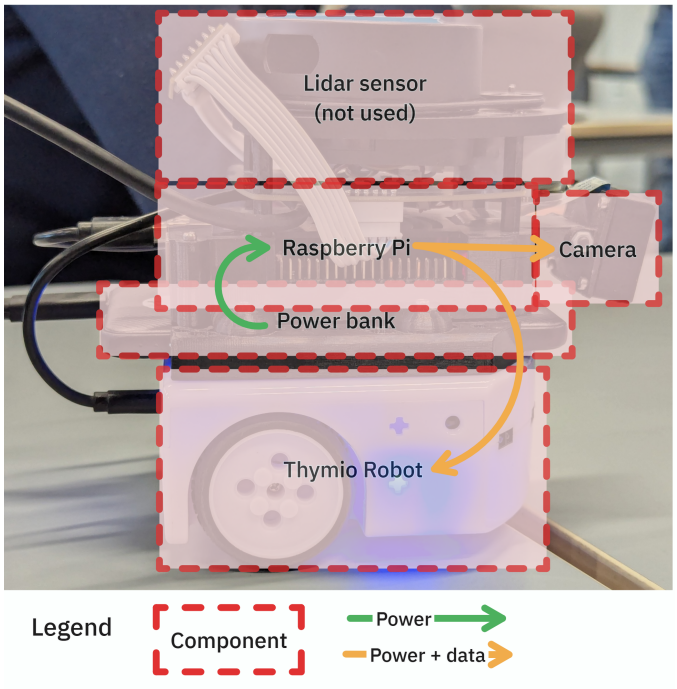

Figure 1: Component Diagram of the robot, side profile

3.1 Raspberry Pi

The Raspberry Pi is flashed with <image>.

It is in charge of running the robot controller code, that communicates with the Thymio via. a Micro USB cable (see Figure 2.

[1] [2], section 1.4

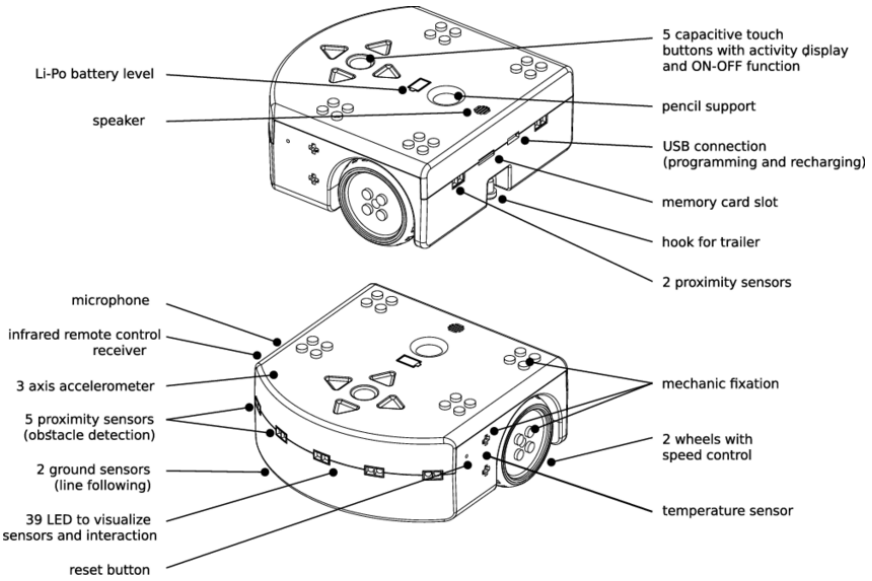

Figure 2: Main elements of the robot

3.2 Thymio Robot

3.3 Camera

3.4 Power bank

We chose a continuous track system with three wheels on either side, each with a diameter of 30 mm, surrounded by a belt with a circumference of 300 mm (see Figure **??**). Two wheels on each side support the weight of the robot, while one is an idler wheel, to increase the tension in the belt (see Figure **??**).

3.5 IR sensors

The robot contains 7 proximity sensors on the side [is it true?], 5 in the front [What does front mean?] and 2 in the back. Our implementation doesn't rely on proximity sensor values, so the details are not described in this report. Instead we describe the RX/TX implementations in the below section since all of 7 sensor are capable to send and receive [RX/TX] values..

RX/TX The sensors are able to send receive signal in a [TODO: some technical details about the capabilities]. The implementation of updating these values need Aseba language [1]

[TODO: maybe add some example code?]

3.6 [TODO: CAMERA] Measurements of Color Sensors

The color sensors measure the red, green and blue light reflected off the surface under them. When following a line, the sensors should be situated above the light grey surface between the lines, since they are placed on the sides of the robot, further apart than the tape line is wide. Therefore, they sense a high average color reflectivity. In contrast, when reaching an intersection, at least one sensor will hover over a tape line, sensing a lower color reflectivity. To calibrate these threshold values, we measure the mean reflectivity of the red, green and blue output of the sensor. Each color sensor was measured while situated above the light grey surface, the intermediate surface on the edge of a line and the black surface on the tape line (see Table **??**).

3.7 Ground sensor

2 ground sensors are on the bottom part of the robot

4 Simulation

The simulation is explained in three steps, the setup, the behavior of the seeker robot and lastly the behavior of the avoider robot.

4.1 Setup

We create a simulation in PyGame, the simulation is from a top-down perspective and we create a square map with walls coloured in red. We also have a square safezone in the center of the map colored in grey. A robot is a pixel-art png of a Thymio robot, see Figure 4. To easily know the heading of the robot, a line is drawn from the robot direction forward. The line is colored red if its a Seeker and blue if its an Avoider. Each robot also has a view frustum simulating the camera using the Raspberry Pi. We set the range of the simulated camera (view frustum) to 2000 units. The view frustum's base width is 100 units and the top width is 1000 units, see Figure 3 Initially all the robots spawn at the same location slightly offset from the safe zone. After 50 episodes all the robots are initialized in random locations and the seeker in the safe-zone, see Figure 5 for visual depiction. When testing the trained model, one Avoider is spawned in each corner and the seeker is spawned in the safezone. The simulation is structured such that it is possible to run n-amount of games concurrently. We opt for this strategy instead of increasing the number of Seekers and/or Avoiders in a single game as this could lead to unstable training. All Avoider and Seeker robots has a random orientation at each episode. We define initial episode length to be ten seconds, then for each episode the length of the episode increases by 0.1 capped to one mintue. After each episode, all concurrent games are reset to initial starting positions. Without the dynamic episode length updates the robots learned to spin in place and by implementing the dynamic updating it allowed it to forget the spinning behavior in early episodes and then later learn to avoid the Seeker.

4.2 Seeker

The Seeker is a fairly simple robot. It moves randomly until it sees an Avoider robot in its view frustum (camera). This then informs the robot whether the Avoider is to the right, left or center of the Seeker. The Seeker then uses this information to move towards the Avoider at max speed. When all robots are caught the Seeker will stop and do a celebratory drift in place. When the Seeker hits a wall it backs up and turns 180°.

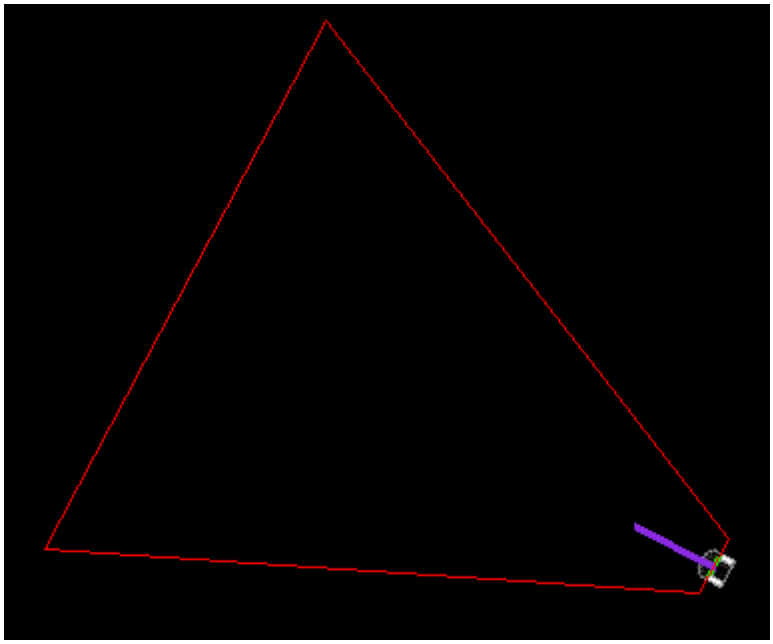

Figure 3: Enter Caption

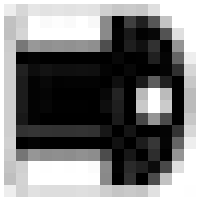

Figure 4: Thymio simulation png

4.3 Avoider

We define three states the Avoider robot can be in. It can be caught, safe and default. Safe is when it is in the safe-zone, default is when the robot avoids the Seeker. When it hits a wall it goes into caught state as the robot is now allowed to wander out of the map boundaries. The Avoider is the robot that utilizes any form of learning. We decide on genetic learning as we can run multiple instances of the game at the same time.

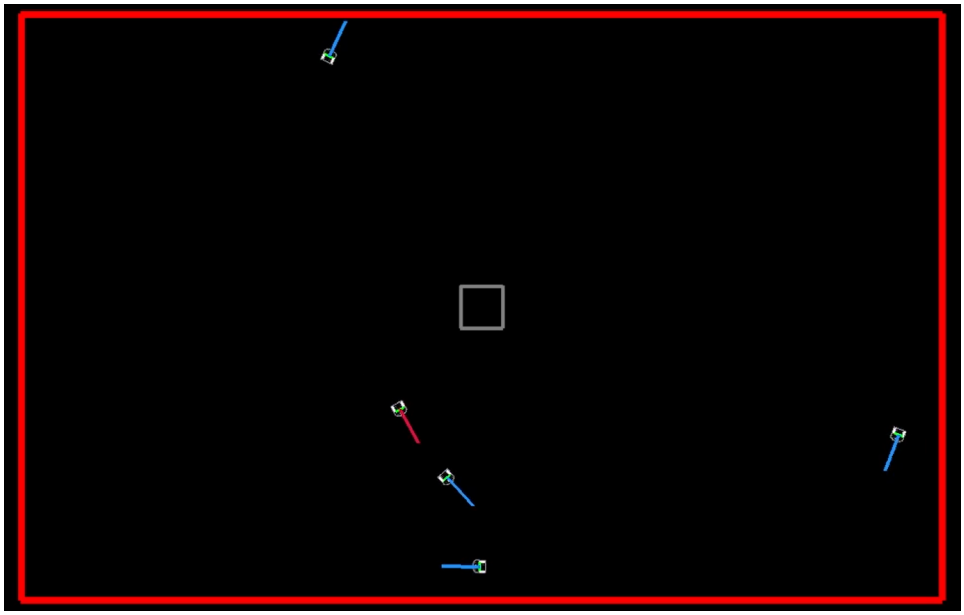

Figure 5: Simulation Setup

4.3.1 Model

We utilize Pytorch to structure the model. The model takes in boolean flags for whether the Seeker is to the left, right or center, whether a robot is within its field of view. It also gets the floor color of where the Avoider currently is. The robot is also provided with the euclidean distance metric to the nearest point on a wall. Lastly, it is also provided with the euclidean to the nearest robot within the view frustrum (camera). We opt for a simple MLP where the output layer has a *tanh* activation function applied to its output to get values from -1 to 1. This value is then the proportional speed the robot should drive with. Negative values correspond to backwards rotation and positive values correspond to forwards rotation. This allows us to run the simulation with larger speeds than the physical robot is able to run with without widening the reality gap. This allows us to perform more episodes in shorter amount of time.

4.3.2 Learning

The genetic learning works by firstly, copying the $top_n = 1/3 \cdot$total population, robots into the next episode. For the robots not copied over, we crossover

two random robots based on rank selection. Specifically the resulting model will have every second parameter from the first parent robot and the rest from the secondary parent robot. After crossover, all models including models copied over to next episode, has a 20% chance to mutate a single weight slightly ($\theta' = \theta + 0.01 \cdot Z,$ where $Z \sim \mathcal{N}(0, 1)$).

The fitness function $R(t)$ defined as follows:

$$R(t) = \begin{cases} -25, & \text{if floor\_value = wall,} \\ R_{\text{base}} - 100, & \text{else if state = caught,} \\ R_{\text{base}} - 0.005 \cdot |S_L - S_R| + D, & \text{otherwise.} \end{cases}$$

$$R_{\text{max}} = T_{\text{total}} \cdot SF, \quad \text{where } SF = 0.01$$

$$R_{base} = \frac{T_{\text{survived}}}{T_{\text{total}}} \cdot R_{\text{max}}$$

$$D = \sqrt{(x - x_{\text{last}})^2 + (y - y_{\text{last}})^2}.$$

R_{base} is the reward for the amount of time survived without penalties. R_{max} is the max possible reward, if the robot was to survive the entire episode. T_{survived} is the time the robot survived. T_{total} is the full episode length. $|S_L - S_R|$ is the absolute difference between the left and right motors. D is the distance traversed since the last frame. $(x_{\text{last}}, y_{\text{last}})$ is the last recorded position of the robot.

The fitness function is calculated for each Avoider at each frame in the simulation. The fitness value returned by the fitness function is summed to a total and this summed value is the value that is being used for our selection strategy. See figure X, for our final models fitness curve over time and see video for a single episode test using the final model.

5 Robot software

Simultaneously with developing our simulation of the task, we did several hardware tests, where we got individual components of the robot to work with small Python scripts.

5.1 Reality gap

After getting the simulation to work, we then started migrating our simulation to the physical robot and bridging the reality gap, by bringing together our Python scripts and replacing our simulated sensors. One key flaw of this was that our seeker in the simulation assumed that we could detect the distance from the robot to the edges of the board. This proved a challenge in the migration, as we did not have a script made that was able to detect this with any of our hardware. We briefly experimented with a proxy for this using the camera and OpenCV, but this proved too unreliable. We therefore decided to instead derive this information by using the floor sensor Section **??**, and if the edge was detected, it would correspond to a distance of 0, and otherwise, an arbitrarily high distance. This in turn resulted in the robot starting to just follow the edges, as it had no longer a notion of when it was almost at an edge, which it had learned to rely on in the simulation.

5.2 Computer Vision

Our computer vision system worked very similar for both the avoider and the seeker program.

They both used OpenCV HSV color ranges to detect contours areas from the camera feed. We calibrated these to respectively detect either blue or red light emitted by the LED's of the other robots. We then extracted the centroid of these contour areas. These would only count as a robot detection if they were larger than a given threshold. We then calculated the normalized position of the centroid in the camera frame in an interval from -1 (left edge of the frame) to 1 (right edge of the frame). For the seeker, we used this normalized position to control the steering of the robot. For the avoider, we used it as binary detection. If the seeker program detected a seeker robot, it turned around and went the other direction.

5.3 Floor sensor

5.4 Integration: Putting it all together

6 Robot Component Experiments

In terms of experiments, we focused on HW capabilities and their implementations conventionally [?]

6.1 RX/TX singals

We tested the distance and angles of which we were able to get caught (recive a 1 on the infrared) and catch others.

[TODO: Angels, distance, etc]

The setup for the distance experiment used two robots, one transmitting 1's and one receiving 1's. The robots were placed 50 cm apart and then slowly moved closer until we registered the 1 signal on the receiver. The distance for when the 1 was received was then recorded using a measuring tape between the robots.

The setup for the angles

6.2 Camera

different RGB colors which are defined like [32, ,32] - blue etc etc...

We applied threshold for focusing on ...

6.3 Ground sensors

We have splitted the raw ground sensor values into 3 group [TODO: Add code]

7 Demo: Robot Competition

On Tuesday, the 12th of december, the team participated in a competition.

To succeed, the winner of a round is the avoider that stays alive the longest or all the avoiders still alive after 3 minutes.

8 Discussion

9 Conclusion

...

10 References

[1] Mobsya Association. *Aseba Language*. `https://wiki.thymio.org/en:asebalanguage`. Accessed: 2024-11-21. n.d.

[2] Robin R Murphy. *Introduction to AI robotics*. 2nd edition. The MIT press, 2019.

11 Appendix

A Appendix example 1.1

Figure 6: Example Appendix 1.1