IT UNIVERSITY OF COPENHAGEN

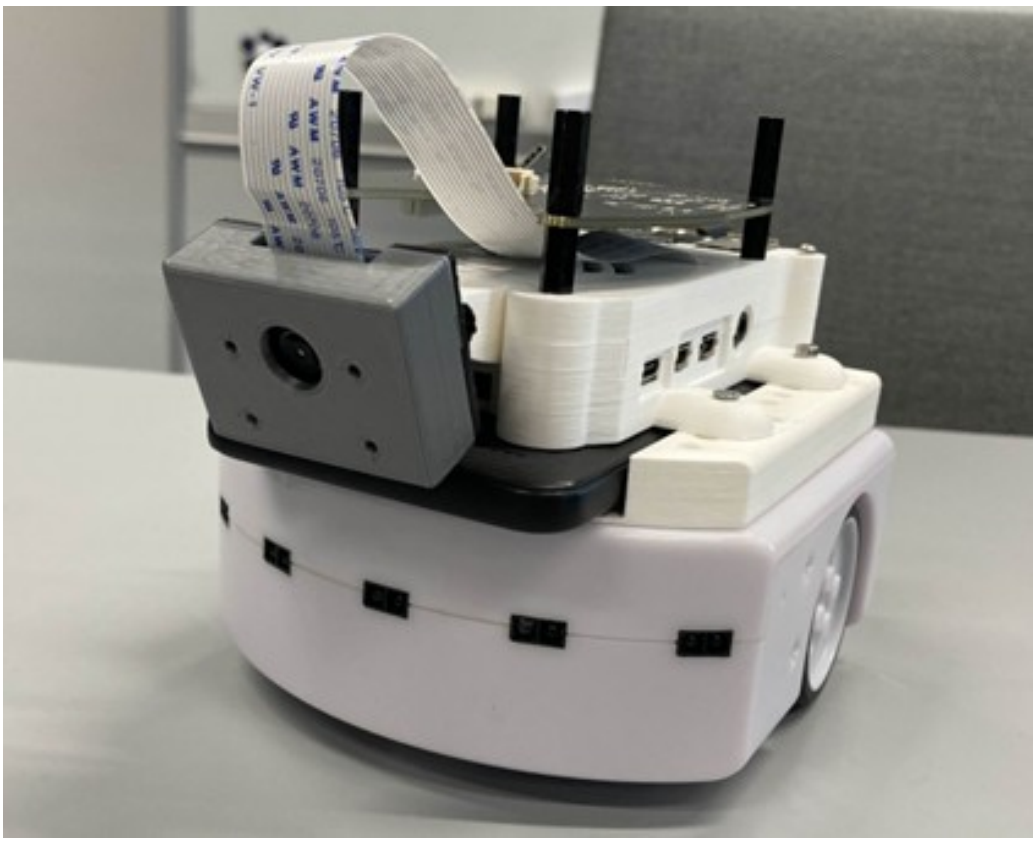
Advanced Robotics
Project Exam
Group 6 (Sixers)

Chrisanna Kate Cornish (ccor@itu.dk)

Gino Franco Fazzi (gifa@itu.dk)

Jan Lishak (jlis@itu.dk)

KSADROB1KU



1 Problem Statement

This study focusses on the development of robots designed to participate in a game of tag[1]. In its simplest form, the game involves one player (the *seeker*) pursuing other players (the *avoiders*) in an effort to "tag" them, thereby removing them from play. The objective is to create a seeker robot and an avoider robot, which may either be distinct entities or a single robot capable of switching between behaviours.

The game setup places the *seeker* at the centre of an arena (see Figure 1), while avoiders begin at the corners. A central *safe zone*, identified by a gray floor, provides temporary immunity to avoiders, preventing them from being tagged. However, this safe zone accommodates only one avoider at a time. If an avoider occupies the zone and another enters, the former must vacate, allowing the latter to take its place.

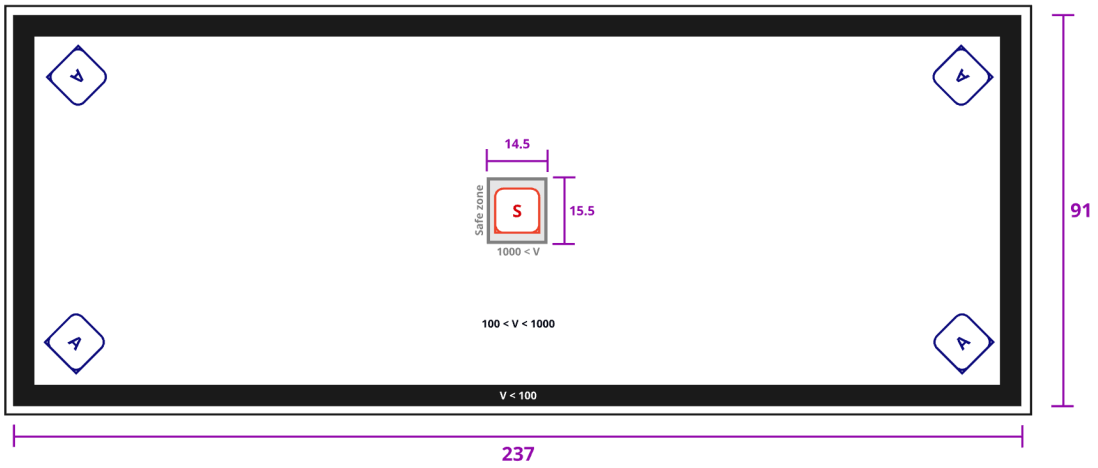


Figure 1: Diagram of the game arena, with dimensions (in purple), showing the placement of the robots (S: *Seeker*; A: *Avoider*) and the tested values for ground sensors on each surface. The sensor readings are denoted as V, with corresponding value ranges indicated.

Robots roles in the game can be identified from their displaying LED light colours: the *seeker* turns its LEDs red, and when in the safe zone, orange. *Avoiders* turn their LEDs blue, and when in the safe zone, green. When an avoider has been tagged, LEDs switch to purple.

The game is played in several rounds, once with each robot as a seeker and the remainder as a avoiders. Victory conditions are defined as follows: the avoider that survives the longest is deemed the winner, or alternatively, all avoiders that evade the seeker for a total of three minutes are considered successful.

2 Robot description

For our experiments we use a Thymio II robot loaded with 9 infrared (IR) proximity sensors (5 in the front, two in the back, and to in the bottom; reach $\sim$ 10cm), 1 three-axis accelerometer, 39 LED lights, and a Li-Po 3.7V, 1.500mAh, battery (autonomy between 3 and 5 hours).

The software controlling the robot runs from a top-mounted Raspberry Pi computer, and is written primarily in Python, using the `tdmclient` library, and the Aseba program for IR communication. This hardware is powered by a portable charger with 5.000mAh and 18.5Wh capacity, connected via a USB-C to USB-A cable.

A front-mounted camera[2], connected to the Raspberry Pi, serves as an additional sensor. It is contained in a case that allows forward and backward tilting, enabling adjustments to the camera's angle for optimal viewing of the horizon/floor.

[1] See https://en.wikipedia.org/wiki/Tag_(game)

[2] Camera Module 3, 12MP IMX708 Quad Bayer sensor with High Dynamic Range mode

Our robot was also originally equipped with a 360° laser distance sensor mounted on top of the Raspberry Pi. We removed this sensor, as it was unnecessary for our tasks, and provided an extra weight that would hinder our performance, both in terms of speed and energy consumption. We had considered using it to detect other robot positions, but the combination of the inbuilt proximity sensors and the camera were sufficient for this purpose. We also noted it would only be effective if all other robots were also equipped with it, due to its relatively high positioning on the bot, and this could not be guaranteed. This follows the design principles of *cheap design* and *redundancy*[2].

The official specifications of the Thymio II, along with all the provided hardware can be found in Appendix A. The final configurations of the robot can be seen in Figure 2.

(a) Front/camera set up (b) Side view

Figure 2: Thymio robot with raspberry pi and camera mounted, final configuration.

2.1 Hardware tests

We tested the robot to assess its capabilities and to capture potential gaps between technical descriptions and real-world execution, particularly with regards to the sensors and motors, in an attempt to better quantify the reality gap.

2.1.1 Surface Reflection

Experiments were conducted to determine the value ranges detected by the ground sensors across different surfaces: open arena surface, black tape (representing the arena boundaries), and the safe zone. After multiple trials, the most consistent intervals observed were as follows:

Black tape: < 100

Safe zone: > 1000:

Arena surface: All other readings fell within this category.

It is important to capture these values, since IR sensors are sensitive to differences in surface conditions and ambient lighting. For example, light-coloured or shiny surfaces reflect more light, whereas dark or matte surfaces absorb more light. Likewise, bright natural or artificial light sources can cause noise or inaccurate readings. These readings will prove essential for the robot's navigation system, as discussed in 2.2.

2.1.2 Proximity detection

The horizontal IR sensors are capable of detecting both reflected IR beams and ambient IR signals. To evaluate their performance, tests were conducted to measure light reflection in relation to object proximity. The results are presented in Figure 3.

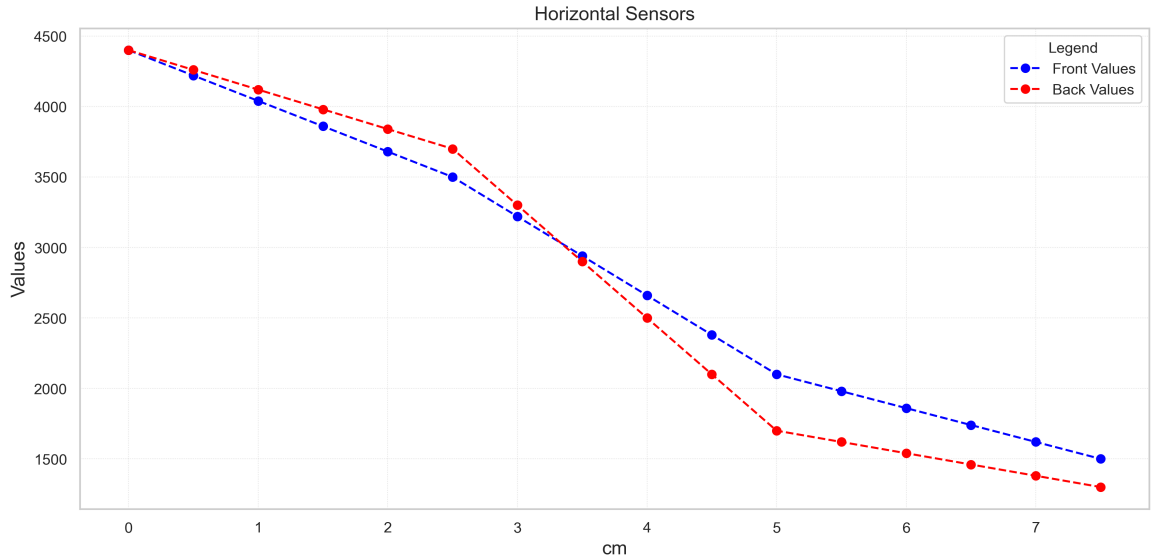


Figure 3: Relationship between object proximity (in cm) and the output values of the horizontal IR sensors. The front sensors (blue) and back sensors (red) exhibit slight discrepancies, potentially caused by variations in lighting conditions or inherent hardware sensitivities.

None of the sensors detected an object at more than 7.5cm, in contrast to the 10cm suggested by the specifications[3]. As discussed in 2.1.1, the reasons for this discrepancy can be varied, including lighting conditions (light source and reflection), object conditions (curvature, angle, and smoothness of objects), hardware conditions (sensor calibration, dirt in the sensors), or others (ambient humidity, movement of the robot, IR emissions from nearby robots, etc).

2.1.3 Infrared signal detection/emission

The robot was able to capture the other robot's signal from a distance of 20.5cm from the front and 23.5cm from the back in our tests with a second Thymio unit (see Figure 4). Later tests under different lighting conditions and with a different second unit found it was able to send and receive signals with a distance of up to 30cm.

2.1.4 Mounted camera

Several tests were conducted to optimize the performance of the mounted camera. These tests included evaluating various resolutions, exposure times, image formats, colour transformations, image flipping, and size adjustments. Additionally, the camera was tested at different angles, ultimately being positioned at approximately 60° to enhance the detection of nearby robots and limit false-positive detection from the surroundings (for example, from people wearing jeans).

There is an inherent trade-off between image resolution and response time, particularly when processing images in real-time for decision-making. This balance highlighted the importance of determining the most suitable configuration to achieve both accurate detection and rapid reaction, ensuring optimal performance during live operations.

2.2 Behavioural Module

The robot is equipped with a behavioural module, inspired by [1], which uses input from its front, back, and ground sensors to govern navigation. This reactive sensor-motor coordination applies a tightly couple perception (sensors) to action (motors) basic heuristic, following an inhibitory approach. The navigation behaviour is rule-based, and determined as follows:

- Surface detection, by use of ground sensor readings (see 2.1.1):

[3] Thymio Specifications: wiki.thymio.org

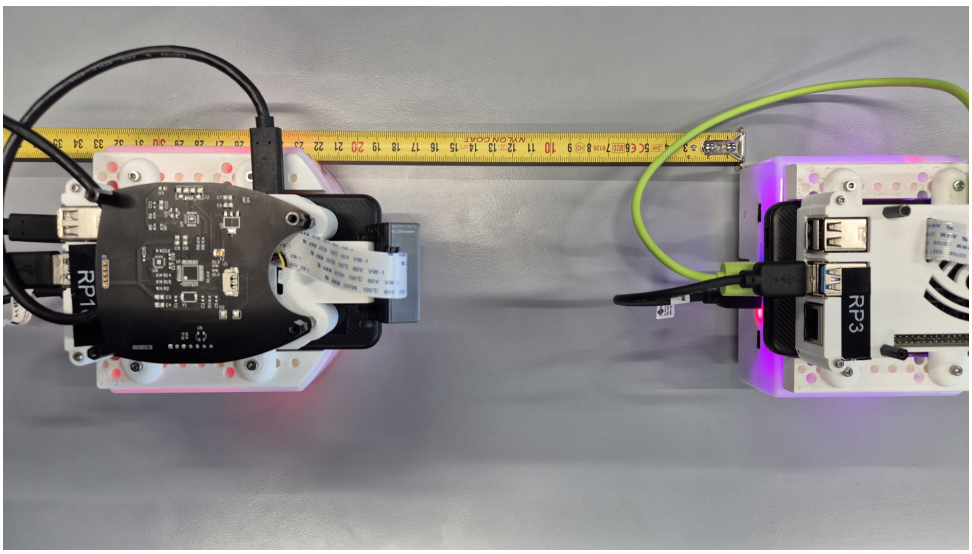


Figure 4: Testing IR signal transmission. The robot correctly “tagged" the avoider at a distance of $\approx 20cm$.

  - Both ground sensors detect a black line (front): Randomly set the left (or right) motor to move forward, and the opposite motor to move backward (at a slower pace first, then at full speed). This causes the robot to turn either right or left randomly, introducing unpredictability into its movement and making it harder for other robots to anticipate its behaviour.
  - Left (or right) ground sensor detects a black line: If only one ground sensor detects the black line, set the corresponding motor (left or right) to move forward and the opposite motor to move backward (at a slower pace first, then at full speed). This causes the robot to turn away from the detected line.
  - Safe Zone: Stop and change LEDs to Green.
  - No black line detected: Move forward at full speed when *avoider*; move at full speed, or rotate slightly (left or right at random) after 4 seconds, when *seeker*. This last rotation is meant to allow the robot’s camera to capture different sections of the arena when in *seeker* mode.

- Obstacle detection, by use of horizontal sensor readings (see 2.1.2):

  - When all front sensor values exceed a threshold (heuristically set at $2,000$), interpret there is an obstacle in front and turn 180°. When an obstacle is detected, a "cool-down" period (2 seconds) is introduced, to prevent the robot repeatedly turning around when surrounded by other robots.

Some of these behaviours are largely independent (they run in parallel), and some others follow a certain hierarchy —e.g., first react to black line so to stay within the arena, then react to close-by robots.

It can be noted that the behaviour of the *avoider* is simplistic (`KISS`[4] design principle), as it tries to isolate itself from any other robot. This behaviour is intentional, under the assumption that the more densely populated a zone is, the less room for movement, less capability for escape, and the higher the chances of being "tagged" (a lonely robot is a safe robot). Therefore, there is no need for an explicit abstract representations knowledge of the environment.

[4]Keep It Simple Stupid

2.3 Infrared Signal

The robots employ an Infrared (IR) signal system for communication, enabling interaction and coordination during gameplay. This system is implemented through an Aseba code script, which bridges the Python interface with the Thymio robot's sensors.

Aseba is a simple imperative programming language with an event-driven architecture, where code execution is triggered asynchronously by events. These events may originate externally (e.g., user-defined events from other Aseba nodes) or internally (e.g., sensor updates).

The robot compiles a specific Aseba script based on its assigned role, defining:

- The type of signal to be transmitted (encoded as an integer): 1 when *seeker*, and 2 when *avoider*.
- The time interval for sending messages (200 ms)
- A variable to store incoming messages.

Upon receiving an IR signal from a neighbouring robot, the system processes the input and adjusts the robot's behaviour accordingly. This communication mechanism is vital to the gameplay and enables the following functionalities:

- Identify when it has been "tagged" (receiving message 1 from the *seeker*) or when another robot attempts to enter the safe zone (receiving message 2, while *avoider*).
- Signal another robot as "tagged" (sending message 1, while acting as a *seeker*).

2.4 Computer Vision Module

The mounted camera (detailed in Section 2) is employed to detect and track other robots, when the robot is in *Seeker* mode. This functionality is implemented through a Computer Vision module, which leverages the `picamera` and `cv2` Python libraries. This module is in charge of creating a `Picamera2` instance, that provides convenient access to the Raspberry Pi's camera system.

We began the development of the computer vision module by considering the resolution of the camera. Reducing the camera resolution significantly decreases processing time. In our case, high resolution is not advantageous, as we are primarily interested in detecting the presence of other robots, rather than capturing fine details of the images. Using the `Picamera2` library, we determined that the lowest resolution that utilizes the full sensor size is 1640x1232. While smaller resolutions are available, employing the full sensor size provides the widest field of view (FOV), enabling us to detect other robots at more acute angles.

It was also important to use manual exposure and white balance settings. Later in the processing stages, we will heavily rely on a specific HUE or VALUE to identify objects. Using automatic settings would introduce unpredictable noise to the processing if the colours shifted over time.

The camera module captures images as three-dimensional pixel arrays. Once an image is acquired, it undergoes preprocessing to facilitate robot detection through colour segmentation. The key steps in this pipeline include:

- The image is resized to 616x820 pixels to reduce processing time while keeping the key details.
- Image Transformations: The raw image is processed using a Gaussian blur to reduce noise and is converted from the RGB colour space to HSV (Hue, Saturation, Value) colour space. HSV separates the colour information from lighting/brightness, making segmentation easier. This conversion enhances the ability to isolate specific colours in the environment.
- Colour Masking: A mask is applied to isolate blue elements in the image, using a fine-tuned colour range ($[110, 50, 50]$ to $[130, 255, 255]$). This allows the system to identify regions of interest that may correspond to other robots in *seeker* mode.
- Contour Detection: The contours of the masked regions are analysed, and the area of each contour is calculated.

- Threshold Assessment: Contours with areas exceeding a predefined threshold (set to 6,000 px) are classified as potential robots. Additionally, an image with the detected contour is saved for debugging purposes (see Appendix B for images from the competition).
- Centroid Calculation: For detected elements, the centroid of the contour is computed to determine the robot's next movement.

Colour Masking must be tuned to the object we are trying to find. An example of this is shown in Figure 5, where the mask was tuned to detect green cups. To do this, we start by having all the sliders corresponding to the HUE values at their maximum and lowering them by small steps until the cup started to fade a little. There are two more parameters that can be tuned: the amount of Gaussian blur applied and the minimal area of a contour to be considered as a possible object.

One major problem in find these correct values is that the camera program needed to be running on the Raspberry Pi, which does not have any display output. To solve this, a VNC server was installed on the Raspberry Pi, which allowed us to view the live output of the camera and adjust the values in real-time.

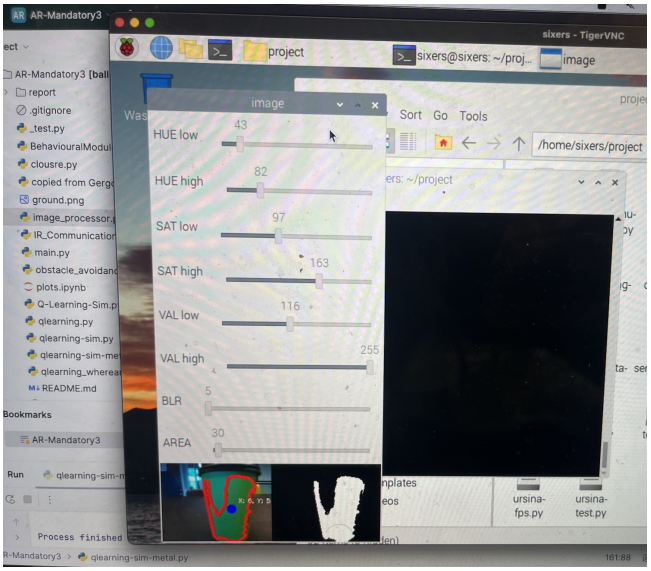


Figure 5: Remotely adjusting threshold parameters using VNC

2.5 Q-Learning

To enable the robot to track and follow other robots while in *seeker* mode, we implemented Q-Learning, a reinforcement learning algorithm where the robot learns through interactions with its environment. The robot receives rewards for achieving desired objectives, allowing it to iteratively improve its behaviour.

Given the complexity of using all sensor and camera inputs as the state space, combined with the motor speeds as actions, the number of possible state-action combinations would be computationally prohibitive. We applied solutions to reduce the state space, following [3] and decided to rely only on the camera vision module as input for the algorithm to reduce the state space.

The output from the Computer Vision Module provides the 2D coordinates of the centre of the detected object, which in our case is the robot. These coordinates are initially in higher resolution and are then scaled down to a normalized 0-10 range for both the X and Y axes. This scaling process results in a 2D coordinate system within a 10x10 range, providing only 100 possible positions corresponding to the robot's centre as detected by the camera.

The constituent elements of the Q-Learning algorithm are detailed below.

Input space

For the input space of the Q-learning, we tried both X and Y coordinates of the object detected by the camera. But in the final version, we only used the X coordinate because it contains the most important information about the left-right position of the object.

Action space

We only allowed the algorithm to choose two actions: steer left or steer right. The robot would be programmed to always move forward, which is usually the most optimal action.

Reward function

The first reward function we tried was:

$$max(X) - abs(X_{center} - X_{object})$$

This function would give the robot 10 points if it was exactly in the middle of the screen and fewer points if the object was closer to any of the sides. If the object was at the extremes, it would get 0 points. In other words, the reward function is how close the object is to the middle. To further improve the function, it was made exponential, which would amplify the rewards closest to the centre. This helped the Q-learning discount rate have a better effect on the learning. The final used function is:

$$(max(X) - abs(X_{center} - X_{object}))^2$$

Q Learning hyperparameters

In Q-learning, success if most often related to selecting the correct input and action space, along with a suitable reward function, and less about hyperparameter tuning. We used standard parameters with a learning rate (α) of 0.1, which determines the extent to which new information overrides old knowledge. The exploration rate (ϵ) was set to 0.2, meaning that, on average, 1 in 5 actions were chosen randomly. This gave us a good balance of exploring the different state-action pairs in a short time.

Simulation

The training was done in a 3D simulated environment. The simulated environment was run in another Python thread, and the rendered images were sent to the main thread running the Q-learning algorithm. We used Python classes as interfaces, which made it simple to change the output from the rendered simulated camera later to use the actual input from a camera. Using the simulated environment made it much easier to have controlled training.

The training consisted in spawning a tennis ball (see Figure 6) at a random location, allowing the robot to move towards it. If the robot couldn't find the ball, the simulation would automatically reset after 15 seconds. In the case the robot would get right next to the ball, it would also automatically reset. This way, the robot was able to train continuously without any pauses or unwanted noise, which wouldn't be possible in the real world. We observed that the Q-values converged after about 20 minutes of training. This setup allowed us to design and refine the reward function, define the state and action spaces, and test the algorithm under controlled conditions, as other robots were rarely available for training.

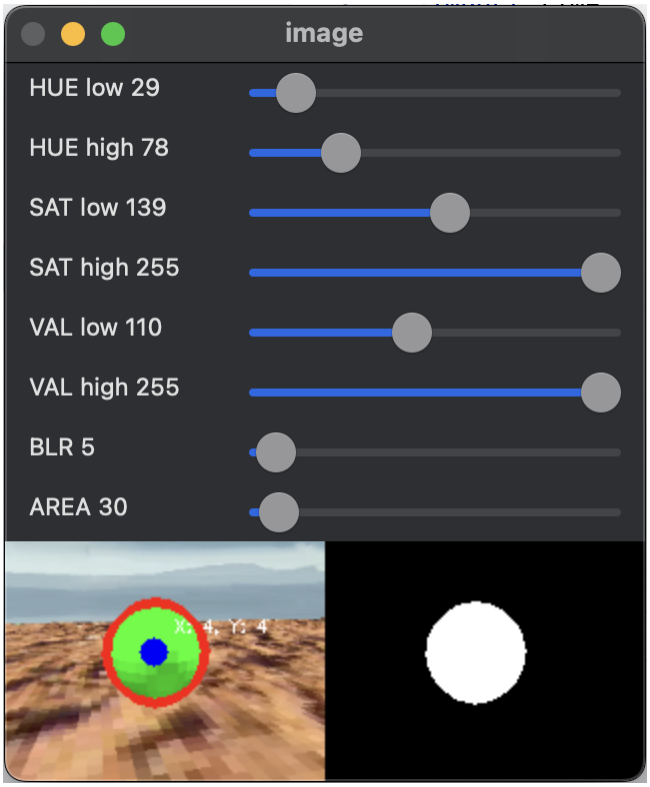


Figure 6: Capture of the robot camera input in simulation, for tracking a green tennis ball.

Moving to the real world

Since the simulation used the exact same interface for controlling the robot, it was easy to migrate the learned algorithm to the real robot. The simulation also used the same image preprocessor, so to follow robots instead of a green ball, only the parameters for the image preprocessor needed to be tuned to match the colour of the robot in *avoider* mode.

As a first step in translating the simulated environment to the real world, we tested our tracking algorithm with a green disposable coffee cup (see Figure 7), as a way to bridge any potential reality gaps.

(a) Cup localisation. (b) Robot view.

Figure 7: Initial Q-learning set-up where the robot learnt to move towards the coffee cup. The contour of interest is marked in red, and its centre in blue.

By following this structured approach, the Computer Vision module enables the robot to efficiently identify and track targets, facilitating dynamic and responsive behaviour during gameplay. In our

different testing phases, we were able to spot blue robots at a distance of $\sim$ 50cm.

It is important to note that the algorithm does not attempt to distinguish between individual robots; instead, it treats all moving robots as potential targets. This lack of specificity can lead to unexpected behaviour, such as the robot frequently switching targets, which may hinder its ability to tag any particular robot. Nevertheless, this was not a particular issue during the competition.

3 Competition Results

During the competition, our robot performed satisfactorily overall, though several issues arose, particularly when it acted as the *seeker*.

In the first four rounds, the robot assumed the role of an *avoider*. It successfully navigated the arena, stayed within the boundaries, and avoided collisions with other robots when they came into close proximity. In three of these rounds, the robot was "tagged", after which it correctly changed its colour and stopped in place as per the rules.

In the fifth round, the robot switched to the role of a *seeker*. It correctly displayed orange when starting in the safe zone and switched to red upon leaving it. Additionally, it successfully detected and pursued at least two different *avoiders* (as evidenced by images captured by the robot during the competition; see Appendix B), and noticeably changed its behaviour when pursuing a target. However, it failed to tag any robots due to a messaging issue in the communication system. An unnoticed bug introduced during a last-minute code change (see Section 4.2) caused the robot to transmit signals as an avoider (specifically sending "2" instead of "1"), even while acting as a seeker. Although the issue was identified and corrected mid-competition, we lost connection to the robot causing it to continue to execute its last received instruction - to move forwards at full speed - and we were unable to re-establish connection before the time limit elapsed.

Despite this unexpected code error, another limitation was greater than we had anticipated. While in *seeker* mode and attempting to tag one robot, another robot was often "hidden" behind the first, effectively using it as a shield. This shielding strategy was observed frequently during the competition and posed a significant challenge for *seekers*. Most robots, including ours, were unable to distinguish between multiple robots in a specific area. Without the ability to identify or track individual spotted robots, *seekers* would follow the most visible one while ignoring the shielded robot, often moving on to a different area of the arena without "tagging" the hider. Addressing this issue would require a more sophisticated computer vision module capable of recognizing and tracking multiple robots in real time, thereby improving tagging efficiency. However, such a solution would involve a far more complex approach than the one employed in this competition.

When comparing our robot's seeker performance to others in the same mode, we observed that it was slower in movement. This was a deliberate design choice favouring precision over speed, as the space was relatively narrow and we needed a direct line between our emitters and their receivers. While this strategy worked well in most cases, it could be adjusted to achieve a better trade-off between speed and precision for future implementations.

As a result of these challenges, our robot tied for last place, with zero successful tags recorded.

4 Discussion

4.1 Design principles

The robot was designed with careful adherence to the Three-Constituents Principle (Figure 8) proposed by [2]. This framework guided the configuration of the robot by considering its intrinsic characteristics and capabilities, analysing the environment and its constraints, and focusing on achieving the designated task successfully.

4.1.1 Agent

The robot's design fulfils all essential attributes: it is autonomous (operates without real-time control from the operator), self-sufficient (powered by its own energy source), embodied (equipped with sensors,

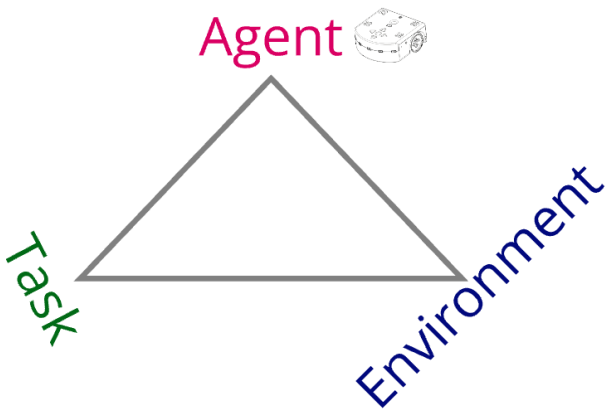


Figure 8: Pfeifer's Three-Constituents Principle: Agent, Task and Environment.

a camera, and motors), and situated (interacting with its environment despite lacking an explicit map).

4.1.2 Task: Evaluation Criteria

The evaluation criteria for the competition evolved from an initial objective —"the winner of a round is the *avoider* that stays alive the longest or all *avoiders* still alive after three minutes"— to the final, explicit objective announced before the competition: "the winner is the robot that tags the most *avoiders* in the shortest time." Despite this change in competition rules, the objectives remained interconnected. For instance, if a robot remains untagged during a round, it effectively reduces the number of robots tagged by the *seeker*, thereby impacting its score.

Additionally, a number of secondary objectives were implicitly defined, all directly aligned with the main competition objective. These included:

- Ensuring the robot navigates the arena safely without exiting its boundaries.
- Demonstrating expected behaviours, such as appropriate light colour signalling and correct actions in the safe zone (e.g., vacating the safe zone upon receiving a signal from another robot).
- Accurately tracking and following *avoiders* when in *seeker* mode.

These criteria were largely straightforward to evaluate through direct observation during experimental trials, with the exception of the IR communication system, which was not directly observable. Unfortunately, due to time constraints, it was not possible to conduct extensive trials of the competition. However, several rounds were completed to assess the robot's performance under the *avoider* role.

We believe that the results obtained provide a fair and realistic representation of the system under study, despite the code error that influenced the robot's performance during the competition.

4.1.3 Environment: Assumptions

The results of this experiment are reproducible only under the specific conditions of the competition. The competition was conducted in a brightly illuminated room, with the arena placed on a table approximately 80cm above the floor.

During robot training, it was observed that environmental factors significantly influenced performance, particularly lighting conditions and the arena's surroundings. For instance:

- When the arena was elevated on a table, higher light reflection affected the sensor readings. Additionally, proximity to windows on a Danish winter afternoon caused the computer vision module to occasionally misidentify blue patches of sky as potential blue robots.

- When the arena was placed on the floor, shadows from nearby objects and passers-by altered surface reflections and disrupted the computer vision module, as the perceived colours became much darker.

4.2 Limitations

Another limitation in our implementation is that the camera is not used while running in *avoider* mode. Its potential use in this case would be to identify the *seeker* (marked by red LEDs), and trigger a special avoidance behaviour, however this adds complexity.

We also considered that a potential effective *avoider* strategy could be to identify a tagged robot and then hide behind it, however this is a complex task that requires identifying and monitoring the *seeker* position and adjusting its own position with regards to the seeker, the shield-bot and the limits of the arena. Whilst it would likely be effective given there can only be one seeker at a time, it is needlessly complicated given a simpler, effective enough solution exists.

Due to time constraints, the final code used in the robot was not properly tested before deployment, allowing a bug to go unnoticed and affecting the robot's performance during the competition (see Section 3. This highlights the importance of thorough testing in software development.

4.3 Other issues

One significant issue arose when bystanders wearing blue clothing were flagged by the computer vision module as potential robots (an example can be seen in the images in Appendix B). We had compensated for this by tuning the blue tones accepted by the image processing module to minimise this but the change of lighting conditions for the competition itself meant we had to widen the range and we were unable to fine-tune sufficiently.

Another issue that was only apparent under competition was the battery packs were relatively loose on the robots and easily dislodged on impact, partially or fully obscuring the camera view of the affected robot (examples can be seen in Appendix B). The cables connecting components together also obstructed the view, and became a hazard as the robots became entangled on one another.

These environmental factors highlight the system's sensitivity and the need for controlled conditions to ensure consistent results. Future trials should address these limitations to draw more robust conclusions about the robot's performance.

5 Conclusion

This project successfully developed a robot that was capable of playing a game of tag, using principles of robotics design, sensor integration, image processing and Q-learning. Whilst acting as an *avoider*, it behaved according to design. When in *seeker* mode, we encountered some challenges, both from a bug in the implementation and hardware/environmental factors, indicating a need for more testing and improvements for better performance.

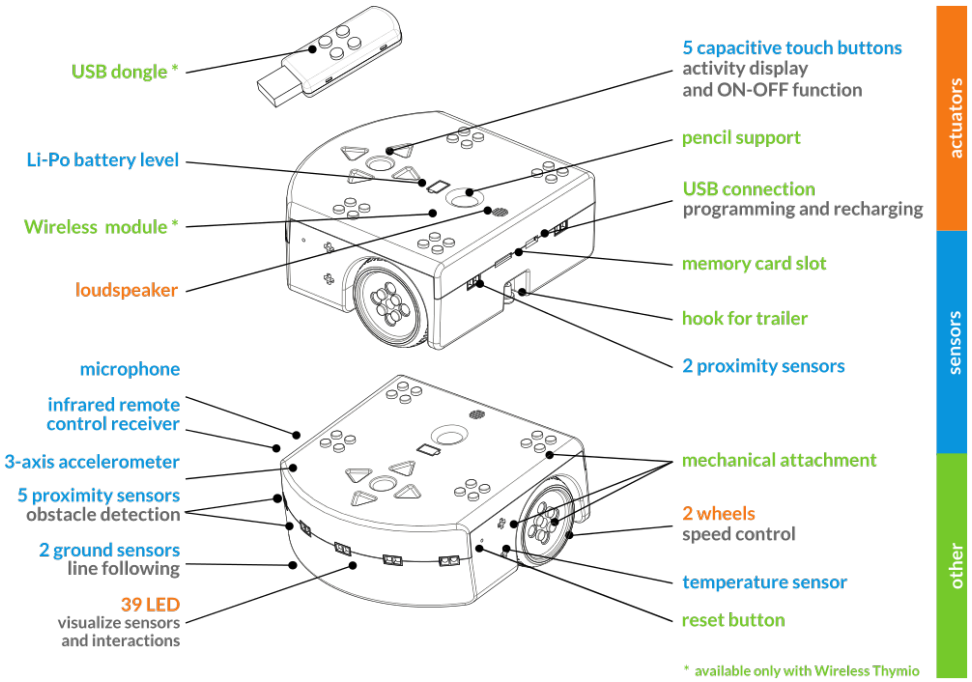
References

[1] V. Braitenberg (1984) "Vehicles: Experiments in Synthetic Psychology"

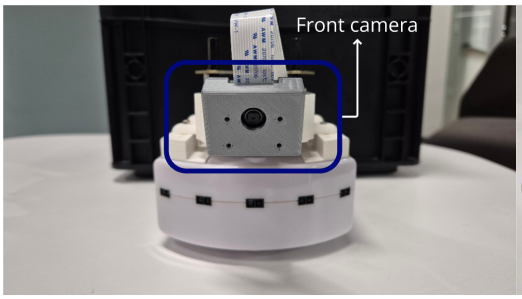
[2] R. Pfeifer and F. Iida, and J. Bongard (2005) "New Robotics: Design Principles for Intelligent Systems"

[3] M. Mataric (1994) "Reward Functions for Accelerated Learning"

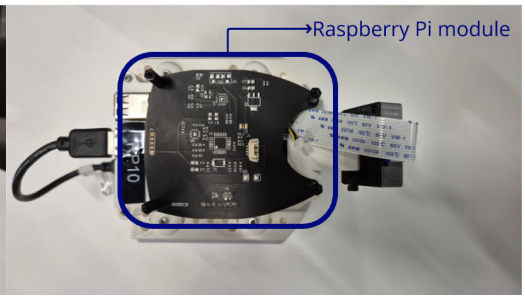
A Robot Specifications



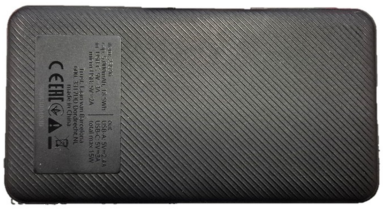
(a) Thymio Specifications (wiki.Thymio.org)



(b) Front camera.



(c) Raspberry Pi module.



(d) Power bank.

Figure A.1: Full Thymio II specifications (a), and all provided hardware: Front Camera (b), Raspberry Pi module (c), and power bank (d), powering the Raspberry Pi hardware.

B Competition

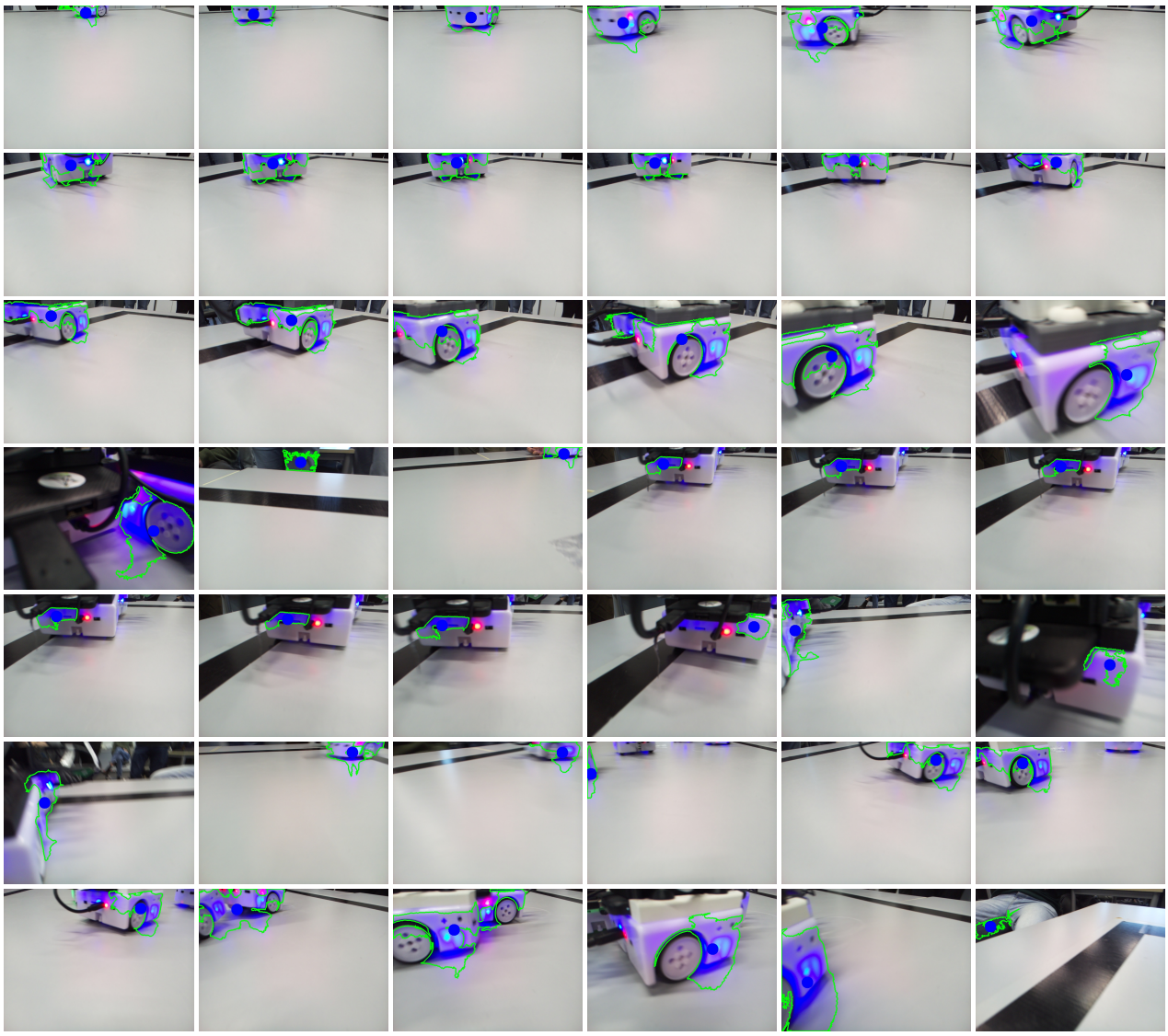


Figure B.1: Images from the Computer Vision module during the competition. The green outline is the largest contour detected and the blue dot indicates the centre point, which was used for targeting by the robot. Misidentification can be seen in column 2, row 4 where a bystander's jeans were seen by the robot. An obscured view due to a dislodged battery can be seen in column 1, row 5, and column 6, row 6.

C Code Repository

All code for this project can be found in the GitHub Repo