Advanced Robotics - Report 2
(Group 3)

Balázs Tóth (bato@itu.dk), Boris Karavasilev (boka@itu.dk), Michele Imbriani (miim@itu.dk)

October 24, 2022

Contents

1 Introduction

The goal of this mini-project was to become familiar with different types of sensors, learn about the implementation of robotics simulations and gain experience with SLAM on a complex system consisting of several components. We used a Thymio II robot which was equipped with several light sensors, a lidar, a camera and a Raspberry Pi as a controller. To gain practical experience with the topics taught in the lectures we had to program our robot to complete a search and rescue mission. Our robot has to explore a classroom environment and find a tennis ball using its camera. After the tennis ball is found the robot shall return to its starting position. During our tests, we found out that our robot is able to locate the tennis ball but due to the inconsistent map of the room it is not feasible to return to the starting position.

2 Exploring the area

The robot should be able to explore the area and avoid walls and obstacles as it drives around.

2.1 Understanding the agent's surroundings

To explore the area around the agent we decided to use the simplest approach possible that fulfils our primary goal to encourage it to navigate towards the direction where the probability of seeing the ball is highest. This can be achieved using different approaches of varying complexities: when deciding what approach to adopt, we relied on core design principles such as principle #1 (the Agent-Task-Environment design principle) and #7 (good ecological balance). Knowing the relative simplicity of both our environment (backside of our classroom, rectangular shape, normal ambient illumination, simple and not too cluttered space) and our task (navigate while avoiding obstacles, detect a coloured object), we decided that an equally simple level of intelligence would be appropriate for our agent, and thus, a simple approach would be suitable.

We implemented a "scanner" system that scans in 8 directions surrounding the robot in the map constructed from the lidar data to gather information about the agent's surroundings: this is used to make informed decisions on which direction to navigate next.

This scanning system (shown in Figure 1) is used for two different purposes:

- to find the direction with the most amount of free space (this serves as a heuristic to navigating towards unexplored areas and increases the chances of finding the ball),
- to scan for walls or obstacles within a certain distance, thus finding the "legal" directions (i.e. a direction without walls or obstacles) that the agent can move in.

In the first step, the values provided by the LiDAR sensor are used to compute the direction with the least amount of "wall probability": the probability values for wall presence are summed up for all the squares in each of the 8 directions. The result of this sensing is a list of 8 values (one for each direction), where a small value represents a high presence of walls or obstacles, while a high value represents the presence of open areas. One of the advantages of this approach is that it is a simple way to use the knowledge learnt about the environment: summing the wall values in the RIGHT direction *BEFORE* the agent explored behind the wall (2) will lead to a lower value being assigned to the RIGHT direction (since it will think there are many obstacles behind the wall), thus discouraging it from going in that direction, while summing the wall values in the right direction *after* the agent explored behind the wall (2) will lead to a much higher and more

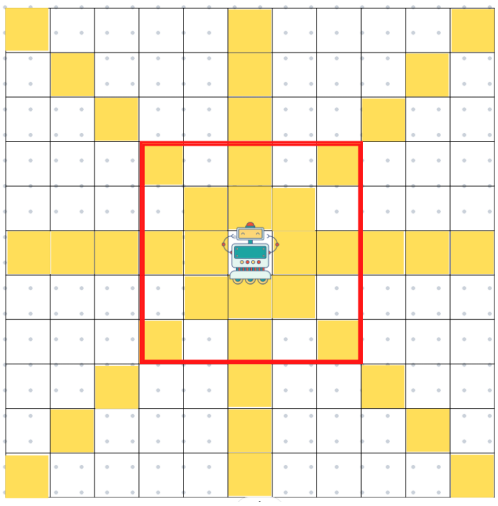


Figure 1: Robot scanning system. (red square: example of possible scanning radius)

realistic value. Of course, the clear disadvantage is that the agent will be more prone to navigate towards areas that are already known, rather than explore the area and find new open spaces. This, however, can be easily accounted for by introducing an exploration factor, which (with a modifiable probability value) would make the agent select a random, legal direction that is not the BEST direction.

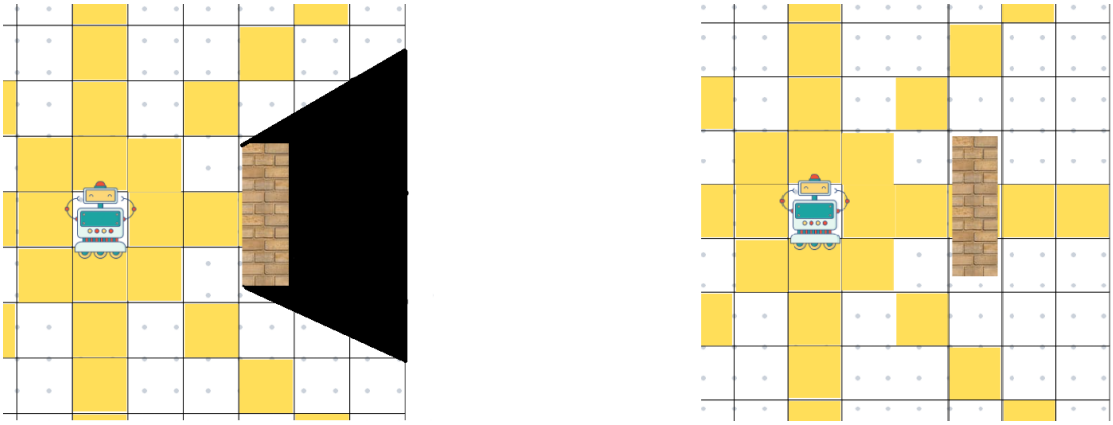


Figure 2: Agent's knowledge before (left) and after (right) exploring behind the wall.

A different scanning radius can be used for the two scanning purposes, and different behaviours can be achieved by tweaking the parameters to find the right combination. During our experiments, we noticed that a higher radius for sensing the direction with the most open space leads to a more exploratory and less predictable behaviour. This is likely due to the fact that the further the agent "looks", the higher the likelihood of walls and obstacles being present is in all 8 directions (i.e. as the radius becomes larger, eventually all obstacles will be scanned). Additionally, due to how the computation of the best open space was implemented (sum of the LiDAR's probability values of walls for each direction), the larger the radius, the more discouraged the agent will be from exploring the walls of the room, since being close to a wall would lead to summing many "wall values" for a certain direction.

Furthermore, we also noticed that a lower radius for the minimum allowed distance to a wall leads to our

agent getting stuck in corners or in between obstacles much more often. A higher value, on the other hand, would make the agent very sensitive to obstacles, making the exploration of the area much more difficult and lengthy.

Based on our experimentation we found the sweet spots for the "sensing open area" radius and "minimum wall detection distance" to be 20 and 5 respectively. By combining these two information (most optimal direction, and legal directions) we are able to instruct our agent on how to explore the area.

Overall, there are clear downsides in our approach. For start, because the agent is discouraged from moving towards areas with walls or obstacles, it is less likely to explore the edges and the corners of the room it is placed in. Additionally, as mentioned above, due to how sensing an open area was detected the robot will be more likely to move towards known areas than to explore new territory. This can lead to a stall situation, where the central area of the room has been explored without the yellow ball being detected. Additionally, while it is true that the robot is able to use the new knowledge about the discovered environment, the fact that the sensing directions are only 8 means that some information about the surrounding environment might not be detected. This is especially true at longer distances between the robot and obstacles, as shown in 3.

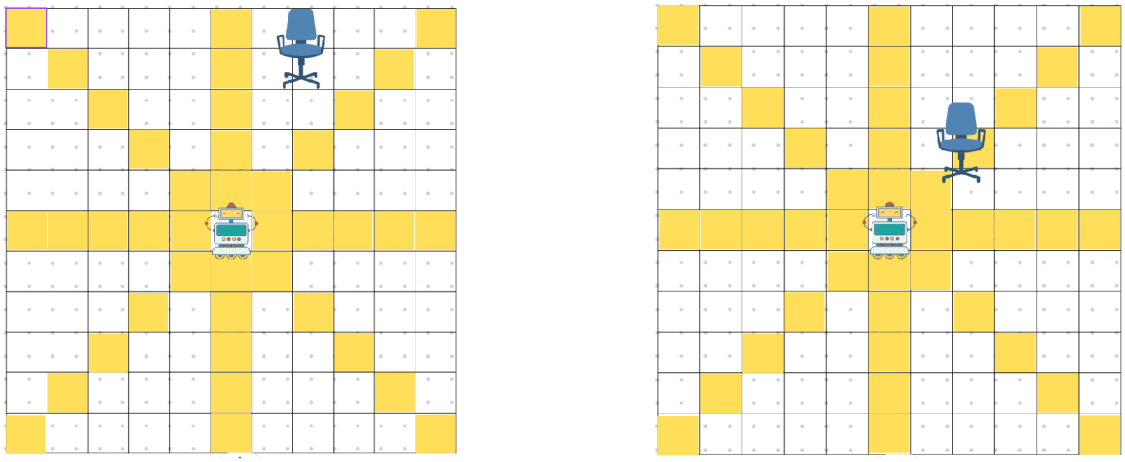


Figure 3: An obstacle in the environment might not be detected by the agent until it is in its proximity.

Despite the discussed downsides of this approach, the advantages (mainly its simplicity and effectiveness at locating the yellow ball) make for a strong approach for our task. As discussed and reasoned in section (REF TO SECTION ABOUT FAULTS OF HARDWARE), we were not able to experience and test the capabilities of our implementation to its full potential. Nevertheless, the limited resulting behaviour was promising, and showed potential for further improvements.

2.2 Moving the agent

Once the agent has discovered the area and marked the target coordinates on the map, it has to perform a set of calculation to determine the angle it has to turn and the distance it has to travel.

The coordinate system the agent has to navigate in consists of 250 * 250 tiles where each tile represents a 6 * 6 centimeter area. When the agent is initialised, it is placed in the middle of the coordinate system which corresponds to x=125, y=125.

2.2.1 Calculations

To calculate the correct angle the agent has to turn and the disctance it has to travel, we use the pythagoras theorem and basic trigonometry.

The first step is to create a triangle by using the current coordinates and the target coordinates. The adjacent and the opposite side of the triangle can be determined by the difference of the target coordinate and the current coordinate.

$$(adjacent = targetX - currentX) \tag{1}$$

$$(opposite = targetY - currentY) \tag{2}$$

After that, we can determine the hypotenuse by using the pythagoras theorem, which is the distance the robot has to travel.

$$Hypotenuse = \sqrt{adjacent^2 + opposite^2} \tag{3}$$

Once we have determined all the sides of the triangle, we use the following equation to determine the degree the robot has to turn:

$$Degree = \arctan \frac{opposite}{adjacent} \tag{4}$$

We discovered that there are some edge cases where the degree calculation does not work due to the fact that it is not possible to divide by zero. In those cases, we hardcoded the result.

We ran 8 tests where we gave 8 different coordinates as an input to the calculation and we precalculated the results. All 8 test cases were calculated correctly by the agent.

2.2.2 Moving the agent to the target

Once the target degree has been calculated, we have to decide that which way the robot has to turn. It is necessary to determine the direction of the turn, since we don´t want the robot to turn 359 degrees when the same result can be achieved by turning 1 degree to the opposite direction. The direction is simply decided by checking if the target degree is smaller or larger than 180. If it is smaller, then the agent starts to rotate clockwise and continues the movement until it reaches the desired position. If the target degree is larger than 180, it starts to rotate counterclockwise.

When the agent has turned into the right direction and is facing the target, it starts to move forward until the x and y coordinates of the agent match the target coordinates.

Unfortunately the test results of the turning and moving did not turn out as we expected since the x and y coordinates from the lidar sensor were not reliable enough, therefore the agent had no chance to reach the target coordinate. We run some experiments to verify the reliability of the lidar sensor which can be seen in the Experiments chapter.

3 Detecting a Tennis Ball

For the detection of our distressed robot mate (tennis ball), we made use of various image processing techniques for performing the most accurate object detection in the quickest time possible. Because of the frame rate of the RaspberryPi's camera module being 30 fps (relatively high), in order to achieve real-time detection (or, at least, with the least delay possible) it was essential to sufficiently lower the amount of computation load put on the CPU during image analysis without losing relevant information about the environment.

For this purpose, the following image transformations were applied: 1) Gaussian blur, 2) RGB to HSV colour space conversion, 3) colour segmentation via thresholding (using hand-picked HSV values corresponding to the yellow tennis ball), 4) open and close morphological operations. Finally, the area of the largest contour present in the binary image is used to decide if there is a tennis ball in the image.

This sequence of image processing operations is performed continuously on every frame captured by the camera.

The algorithm is very sensitive to artificial lights: when introduced in the frame, artificial lights skew the colours of the objects present in the scene, thus making the yellow colour of the ball fall outside of our specified threshold range. This, however, is not a significant issue since the camera is attached at 0° angle from the ground, thus making the chances of artificial lights being present in the frame minimal. Additionally, knowing the environment that the agent will operate in (backside of classroom) removes the need of handling such situation, since artificial lights are not present in such environment. A similar argument can be put forward for the presence of natural light, though it has a much less significant effect on the correct detection of the yellow ball regardless.

All in all, our friend-in-distress algorithm successfully detects a yellow ball in a well lit environment in distances between 3cm and 250cm, which is a sufficient distance range for our problem.

4 Pathfinding

Even though the created map of the room from the lidar data was not stable enough in order to be usable to guide the robot back to the starting position we propose several pathfinding approaches that could be implemented if the map was static through the lifetime of the robot.

4.1 Saving visited coordinates

The simplest way to drive the robot back to the starting point is to save the visited map tile coordinates. Once the tennis ball is found, the robot could simply follow the list of visited coordinates in reverse order. This solution guarantees that the robot reaches the starting point without hitting an obstacle; however, it will very likely not find the optimal solution.

4.2 A* search

A* search can be used in combination with the previous method. The robot would still save the visited coordinates, but it would be able to find the shortest path within the already visited tiles.

5 Experiments

Because there are many variables in our robotic system that can influence the ability of the robot to complete its task we performed a series of experiments to determine the amount of imprecision of our sensor readings and establish a baseline.

5.1 Position and Orientation Drift

One of the most important data that our robot collects is its position and orientation. Therefore, we measured how these values drift when the robot remains static both in a stable and in changing environment.

5.1.1 Static Environment and Static Robot

For this test, we placed our robot in a static environment seen in figure 4 and the robot remained static as well. Despite these circumstances, the position of the robot drifted by 2 tiles on the x-axis and by 8 tiles on the y-axis in 5 minutes. The orientation of the robot drifted to a negative 14 degrees from 0 within the same time frame. We took measurements every minute for 5 minutes and the results are summarised in table 1.

Elapsed Time (min)	0	1	2	3	4	5
Position (x)	125	125	126	127	127	127
Position (y)	125	123	123	122	120	117
Orientation (deg)	0	-5	-8	-10	-12	-14

Table 1: Drift in position and orientation over 5 minutes

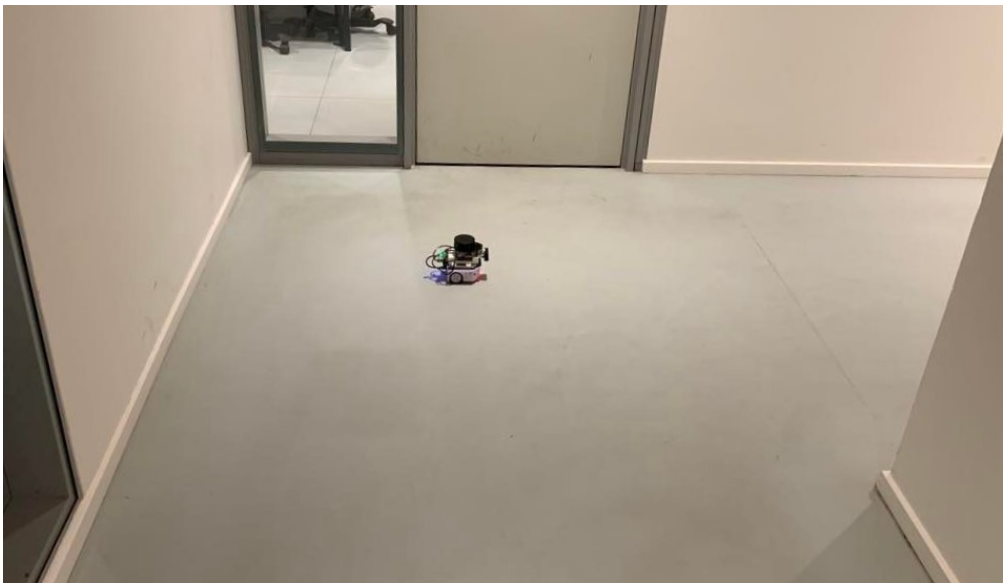


Figure 4: Static testing environment

The drift of the values is also visible in the visualisation of the robot's map in figure 5. The red ellipse

symbolises the robot's position and the blue dot points in the direction with the most amount of free space that the robot is inclined to explore next.

(a) (b)

Figure 5: Map on the beginning and after 5 minutes (left and right respectively).

5.1.2 Dynamic Environment and Static Robot

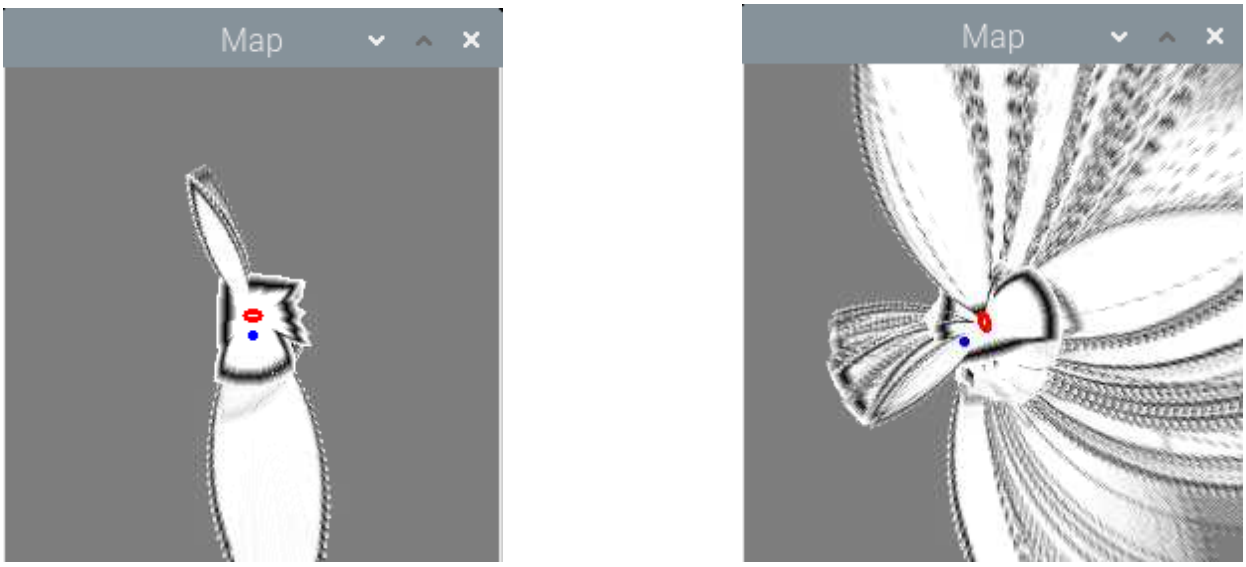
This test took place in the same location as the first test but we were changing the environment every 10 seconds by relocating a person within a radius of 1 meter around the robot as seen in figure 6. This environment led to a drastic increase in the drift of values as seen in the table 2

Elapsed Time (min)	0	1	2	3	4	5
Position (x)	125	121	121	119	119	121
Position (y)	125	135	132	130	131	129
Orientation (deg)	0	-65	-81	-87	-90	-107

Table 2: Drift in position and orientation over 5 minutes

The increased drift of the values is also visible in the visualisation of the robot's map in figure 7.

Figure 6: Dynamic testing environment



(a) (b)

Figure 7: Map on the beginning and after 5 minutes (left and right respectively).

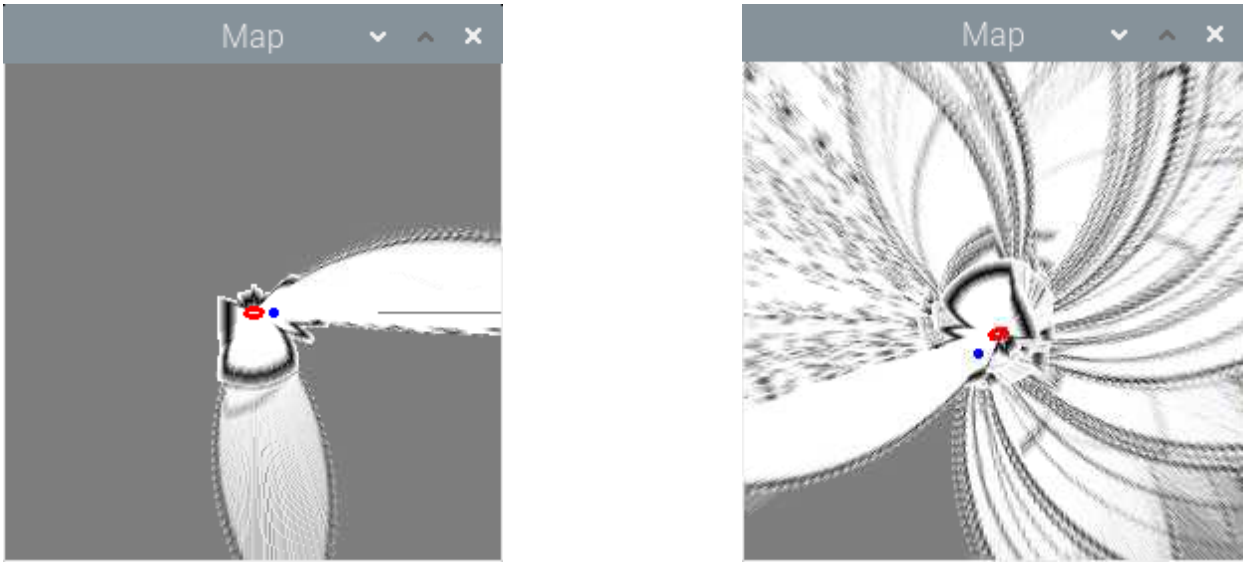
5.1.3 Static Environment and Rotating Robot

The third test took place in the same location as the previous tests, where the environment was static and the robot turned four times 180 degrees to one side and four times 180 degrees back. From the data in table 3 we see that at the end of the experiment the orientation was negative 200 degrees instead of being back to zero.

Real Orientation (deg)	0	180	360	540	720	540	360	180	0
Measured Orientation (deg)	0	154	303	456	612	403	210	-4	-200
Position (x)	125	121	117	117	116	117	120	123	127
Position (y)	125	125	125	130	131	135	135	135	136

Table 3: Drift in position and orientation during 4 half turns in both directions

The drift of the values is even bigger than in the test with dynamic environment and static robot and that is reflected by the images of the map in figure 8.



(a) (b)

Figure 8: Map before and after 4 rotations in both directions (left and right respectively).

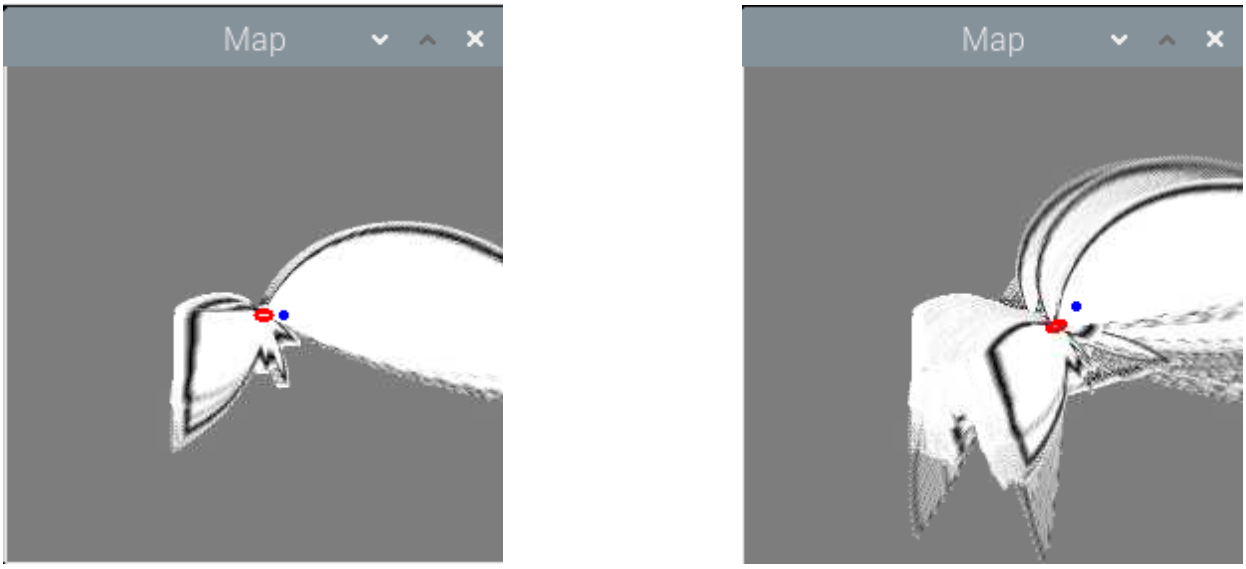
5.1.4 Static Environment and Translating Robot

The fourth test took place in the same location as the previous tests, where the environment was static and the robot was moved five times five tiles forward and five times five tiles back. The entire map is 250 tiles long along each axis and one tile corresponds to 6 cm in reality. In table 4 we can see that the robot's position remained precise ($\pm$ 1 tile) during the forward translations but on the way back its position barely changed or even drifted slightly forward. The strange results in the second part of the experiment could be due to a not completely static person pulling the robot on a piece of paper and a lack of landmarks in the environment.

X-axis Translation (tiles)	0	5	10	15	20	25	20	15	10	5	0
Position (x)	129	135	139	145	150	154	155	156	154	156	157
Position (y)	125	125	126	126	126	128	128	130	131	130	130
Orientation (deg)	0	-5	-12	-16	-16	-23	-19	-21	-22	-19	-21

Table 4: Drift in position and orientation during translation along the x-axis.

In the images of the map in figure 9 we can see that the map has rotated counter-clockwise during the experiment the same as in all of our other experiments, this might be due to a misalignment between the angular speed of the lidar and the sampling rate.



(a) (b)

Figure 9: Map before and after 5 translations forward and back (left and right respectively).

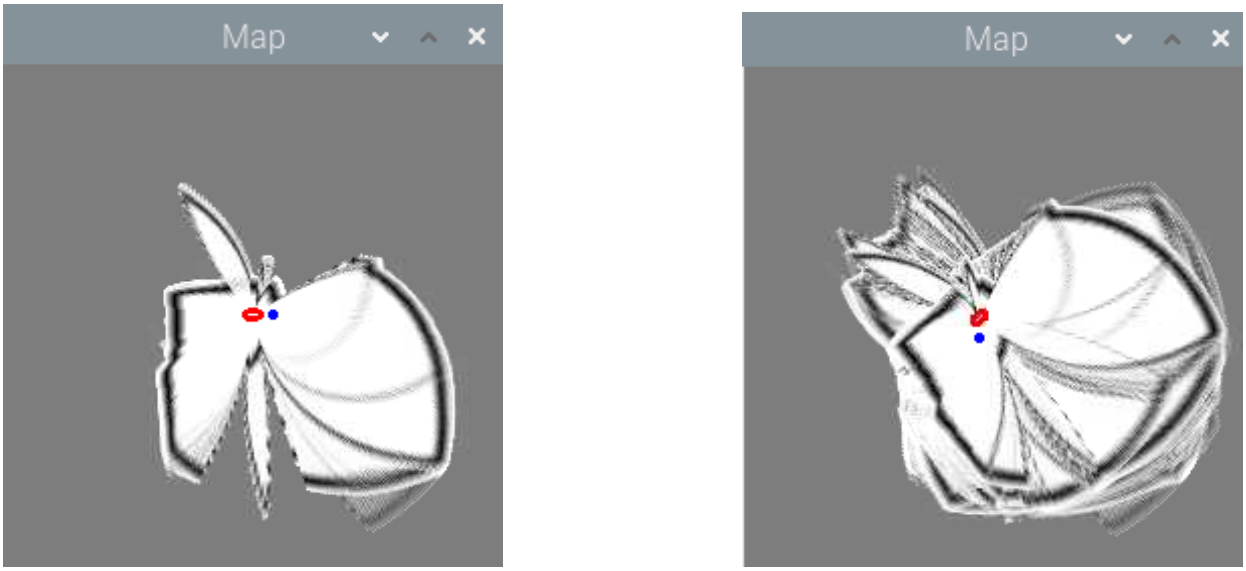
5.1.5 Static Environment With Landmarks and Translating Robot

The fifth test was exactly the same as the fourth test described in section 5.1.4 except that the robot was being translated in a classroom full of chairs, tables and other landmarks. From the data in table 5 we see that the robot was able to detect the forward motion as in the test without landmarks but it was also able to detect the backward translation along the x-axis.

X-axis Translation (tiles)	0	5	10	15	20	25	20	15	10	5	0
Position (x)	124	127	129	131	133	136	132	128	124	121	116
Position (y)	124	122	121	119	118	117	118	120	122	122	124
Orientation (deg)	-2	-9	-15	-19	-23	-25	-26	-30	-32	-34	-36

Table 5: Drift in position and orientation during translation along the x-axis.

The fact that the robot sensed both forward and backward motion is visible also from the map in figure 10. In the end of the experiment, despite the rotation of the map the robot appears to be very close to its initial position (as expected).



(a) (b)

Figure 10: Map before and after 5 translations forward and back in an environment with landmarks.

The testing environment for this experiment is shown in figure 11.

Figure 11: Static testing environment with landmarks.

5.2 Tennis Ball Detection

To test the reliability of our tennis ball detection we performed two tests. One in good lighting and a second one in bad lighting. Our detector is robust and detects the ball reliably in both conditions if the HSV target colour is calibrated for the given conditions.

5.2.1 Good Lighting

The tennis ball was detected up to a distance of 210 cm in good lighting as seen in figure 12.

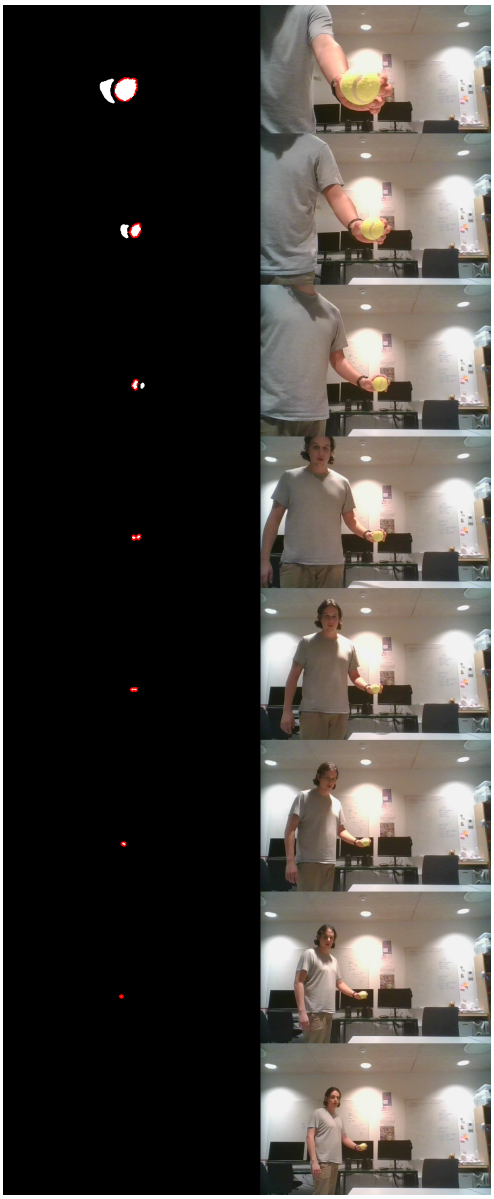


Figure 12: Detection of the ball at distances increasing by 30 cm in good lighting.

5.2.2 Bad Lighting

The tennis ball was detected up to a distance of 180 cm in bad lighting as seen in figure 13.

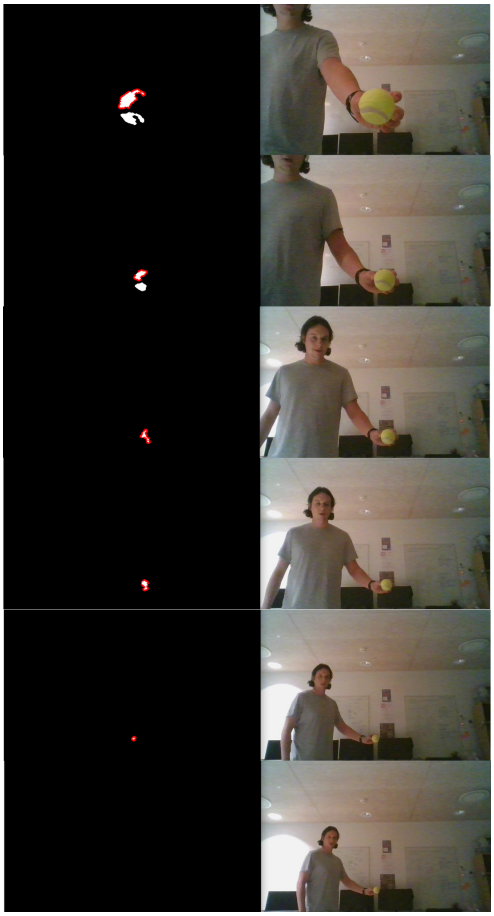


Figure 13: Detection of the ball at distances increasing by 30 cm in bad lighting.

6 Conclusion

We were very close to completing the task and we successfully tested the individual systems and got them working. Unfortunately, because our robot had a faulty camera that was not recognised by the Raspberry Pi and we did not have enough time to set up another group's robot to work with our code we did not take part in the final contest. Despite the circumstances, we managed to retrieve and visualise a map of the robot's surroundings and make the robot explore the environment. Thanks to our implementation of a hierarchical control system we also took advantage of the infrared sensors that acted as a safeguard to prevent collisions with walls. We were also able to reliably detect a tennis ball on our laptop's web camera. We documented the insights from our experiments and the limitations of the provided platform in the section 5.