MOBILE ROBOTS
PROJECT REPORT – GROUP 47

João Vidal fc54467 **Henrique Barata fc54467**

MAZE NAVIGATION

INTRODUCTION

For this project, we decided to use Thymio for a maze navigation problem in which the robot would go to a set of specific locations in a maze, being able to navigate to and from them according to different commands – given by an rc5 remote controller.

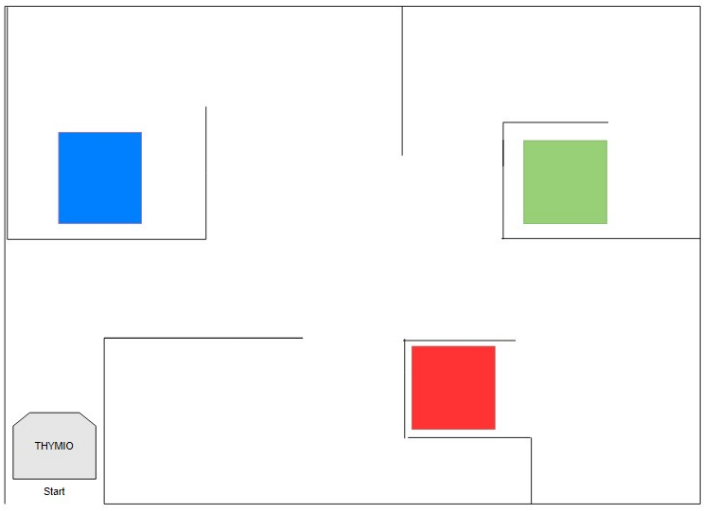

Figure 1 - Idealized Maze

The image on the left represents the initial sketch that was made of the maze, with the specific locations that thymio would navigate to marked by three colored baseplates spread throughout the maze.

The baseplates were marked with the colors Red, Green, and Blue, chosen for being the primary colors in Thymio's LED system. This would serve to alert us of its arrival at the designated location by using the prox.ground sensors to distinguish the baseplates Thymio was in, and turn on the LED's accordingly.

APPROACH

At first, we were planning on using a right-wall hug approach, but, given the cost of the maze and the different paths we wanted the robot to take, this would take unnecessary amounts of time and take high-cost, large routes around the maze to go to simple close paths.

Since sensor navigation is just a blind mode of avoiding obstacles, that would be too simplistic, so we decided to use space localisation to navigate the maze. Using our own calculations and taking advantage of some odomety, we make use of the x and y coordinates and calculate their changes in the 4 directions.

We ran into a few obstacles whilst coming up with an algorithm, due to our little experience with aseba and its memory conditions and variables. We decided to create a simple program that calculates the path the robot must follow after receiving an infrared command.

This made it possible for the robot to navigate 12 different paths, between the colored baseplates and the beginning of the labyrinth.

Additionally, the robot uses horizontal proximity to avoid crashing if it detects an obstacle, making it stop in its place.

IMPLEMENTATION

1. The maze

Using the instructions provided in the Aseba wikidot website, we built a virtual playground using a text editor in order to remotely run and test our navigation model.

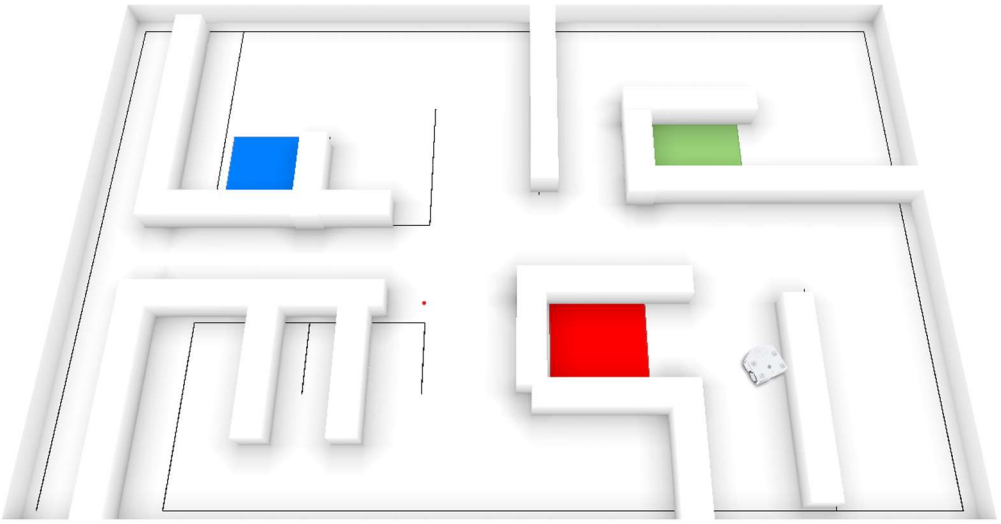

Figure 2 - Playground maze coded

This virtual model was used to adapt and re-run several tests of the algorithm and path mapping so we wouldn't have to constantly be shifting a real physical maze around.

The main issue regarding the playground maze was that the playground does not support infrared commands, and also correcting the algorithm to navigate on this path meant having to do additional calculations and adjustments when translating the movement to a real-world scenario.

The image on the right shows the physical version of the maze we built for the robot to navigate, which required several adjustements of turning speeds and node distances in order to work properly.

Even then, there are still some slight errors depending on the surface the robot sits on.

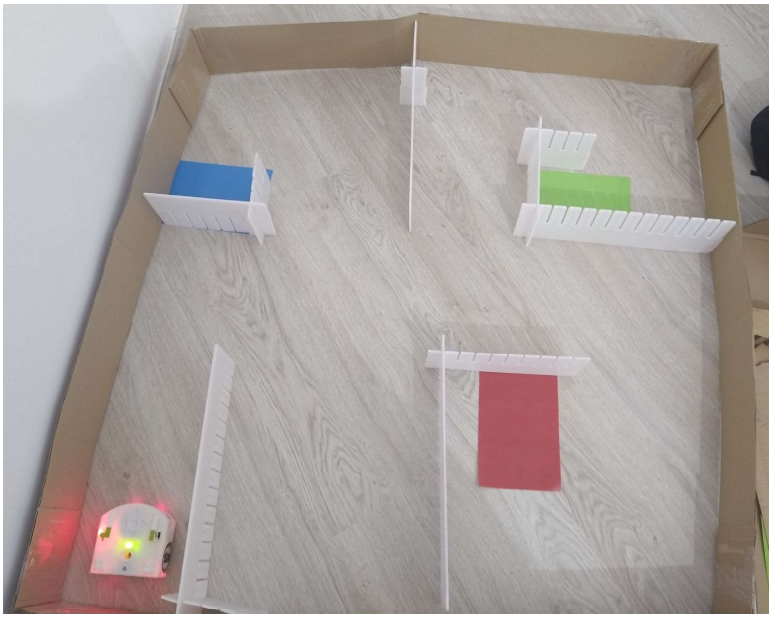

Figure 3 - Real Maze

2. The Navigation Model

We created a timer that would calculate the robot's moving state every 50ms, and calculate its travelled distance based on the **speed and direction** of the robot (updating x and y).

Since we store the nodes that the labyrinth has that are **navigable** by the robot, we can create a path for the robot to follow. Each node has x and y coordinates so the robot knows **which node to travel to next.**

To calculate direction, we compare the distance of the x and y coordinates and we can verify which axis the robot should be moving in. After so, another timer is created that is responsible for **the activation of the different movements** the robot can make. With the robot's current direction and the goal direction being verified, if they are equal, the robot can move **forward**, if not, he turns to the side that **grants him the optimal path**.

The robot also has a **stop** function that tells him it has reached its destination, which will be marked by a colored baseplate. If the color of the baseplate matches the given command, it lights up the color of the baseplate and that means success.

In order to make sure the robot does not crash, we implement a stop system when the front horizontal sensor detects proximity. The robot stops in the middle of the labyrinth, and it readjusts its path accordingly. Since we kept encountering an Error that stated: *'Script too big for target bytecode size'*, we left this function in, but it would only readjust the path to the bluebase, working independently from the position of the labyrinth the robot was in.

Like we previously studied, these types of localizations based on travelled distance can be affected by several different factors, such as the adherence of the wheels on the target surface (it was necessary to correct for several different surfaces), wheel misalignment, that may require additional calibration sometimes and some other calculation errors.

3. The Sensors

Another problem we encountered was regarding the prox.ground.sensors. Since their intended use is for measuring the distance to the ground, this makes them helpful in scenarios like following a black line on the ground, but not very good in color detection.

We had to use different materials for the baseplates and adjust their values each time we changed places for different types of ground the robot would travel (a black desk presents different values of reflexivity from, for instance, a white floor).

```
onevent prox
   if prox.ground.delta[0] >= 670 and prox.ground.delta[0] <= 720 then
      call leds.top(0,0,0)
   end
   if prox.ground.delta[0] >= 740 and prox.ground.delta[0]  <= 800 and rc5.command == 2 then
      call leds.top(0,0,prox.ground.delta[0])
   end
   if prox.ground.delta[0]  >= 500 and prox.ground.delta[0] <= 600 and rc5.command == 1 then
      call leds.top(prox.ground.delta[0],0,0)
   end
   if prox.ground.delta[0]  >= 850 and prox.ground.delta[0] <= 920 and rc5.command == 0 then
      call leds.top(0,prox.ground.delta[0],0)
   end
```

Figure 4 - Codes to turn on the LEDs according to the baseplates

Note: these values are working accordingly to the floor shown in Figure 3, and need to be adjusted for different surfaces and their reflexivity.

4. The InfraRed Remote

We used an infrared remote to give thymio the commands assigned to each baseplate, so he could determine which path he would have to take.

The controls implemented were "**1" – Red; "2" – Blue; "0" – Green;** as well as an additional command to return to the initial location in the maze (the command "3").

We also coded the sequence of commands required to control the robot using the remote control arrows, for demonstration purposes, allowing us to navigate the maze the way we want or adapt it into some sort of game.

RESULTS

The results that we achieved were promising, both in the playground maze and in the real one. Since the distances are adjusted to the real world, the playground can no longer navigate accurately through the coded walls, but it can still be used to test the algorithm and verify if the paths are being built and navigated correctly.

The infrared remote was being used correctly wether to input commands or to control the robot, and the ground sensors are correctly displaying the LEDs based on the baseplate they encounter.

The robot can successfully navigate the maze given a command to go to a specific baseplate and stop when it reaches its destination. The paths are correctly stored and interpreted by the robot to follow the different nodes.

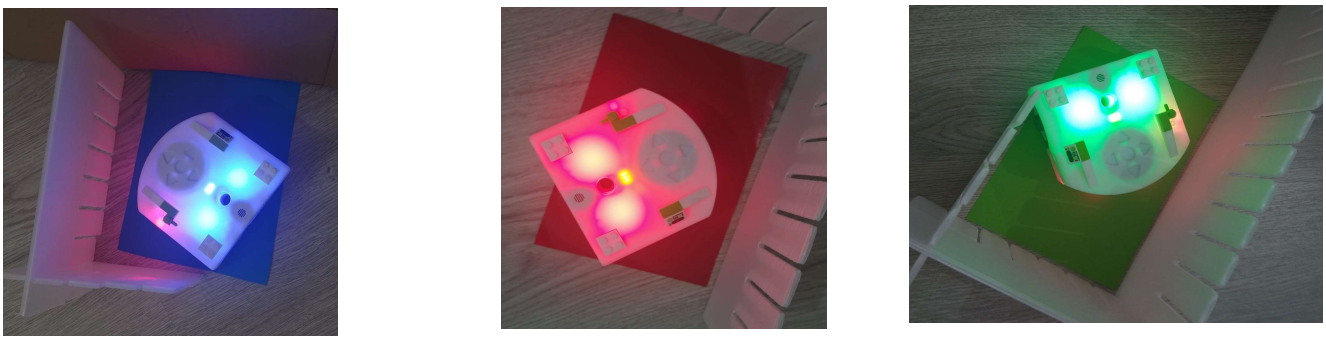

Figura 5 - The robot after successfully navigating to the blue, red and green bases, and lighting the respective LED's

SOME OF THE ISSUES FOUND

Problems with mapping in AESL, made us manually create the paths using nodes;

Problems with the robot's memory limitations: the script's memory was limiting and we couldn't implement everything we planned. We had to cut parts of the code to make it run, for example, when the robot encounters an obstacle, it only turns to the left, which was the only part we could implement due to reaching the script's code limit.

Problems with the floor sensors, as they gave different values for the same floor color in different locations – it depends not only on the distance but on the light, material and texture of the surface.

Problems with the robot's turning and direction-changing, as they are not always accurate, causing the robot to move in crooked directions and not navigate correctly in the maze. Since these deviations cannot be calculated using odometry, we have to manually adjust the robot's position for it to continue the path successfully.

FINAL COMMENTS

We enjoyed making this project for the novelty and creative experience that we hadn't had contact with before. Learning how to make algorithms in aseba without the options to build matrixes was tough, but it was a very interesting project overall. We would like, for future iterations, to be able to dynamically choose which nodes the robot navigates, allowing us to alter the maze and the robot could navigate it after tracking all possible paths.

The main thing we would like to work on are the errors related with the adherence of the wheels on different floors. The robot, when moving from a position to another from one of the baseplates, will sometimes knock over walls. – even after we verify that the code is correctly implemented (by running the playground simulation or removeing the walls) - since the navigation will take the robot to the desired position. If we could ensure the errors with turning and spinning were reduced as much as possible, we could perhaps navigate more accurately between nodes.

BIBLIOGRAPHY

HTTP://ASEBA.WIKIDOT.COM/EN:ASEBAPLAYGROUND