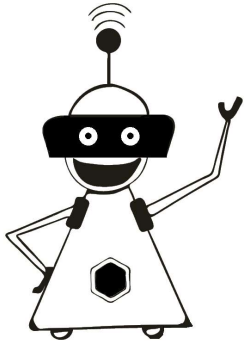
Tankerino

AI lawnmower

Department of Theoretical and Applied Science (DiSTA)
Master's degree in computer science
Thesis Advisor: Dott. Gallo Ignazio
Candidate: Francesco Belloni
Matricula: 725676



What is the most boring task you have to do in the garden?

Cutting the grass

Tankerino

This project intends to build an autonomous lawnmower prototype.

1) the robot should use its sensors to navigate the area with no user input

2) mowing as much of the field as possible

3) avoiding obstacles and staying in the right area

The idea for this project arises because current lawnmowers use:

- a perimeter wire (staying on the mowing area)
- random movements (are not efficient)

Autonomous systems need to travel the world on their own

- What does the world look like?
- Where am I?
- Where to go?

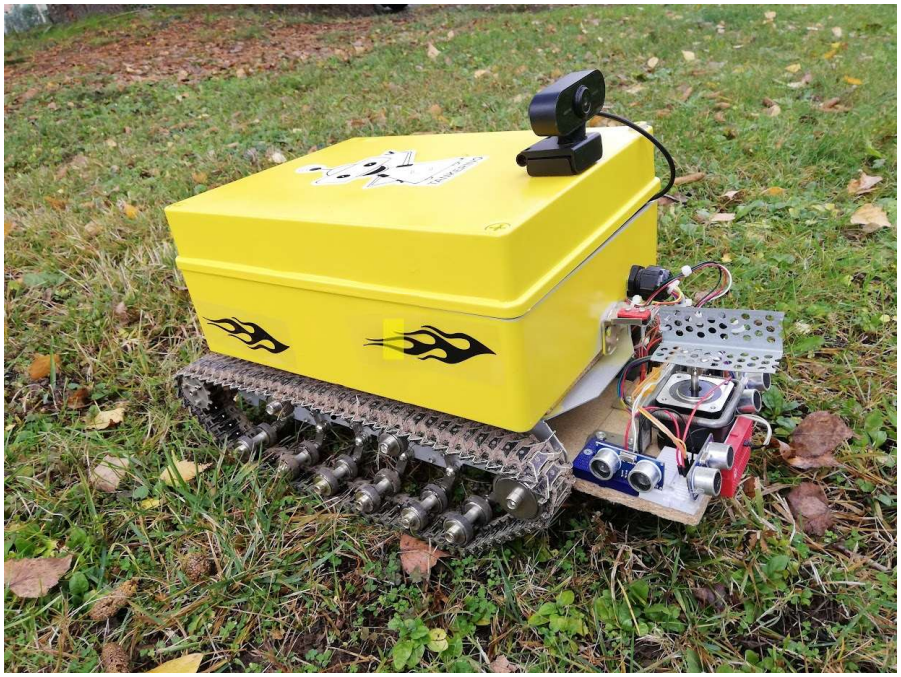
Sensors external

1 - What does the world look like?

Moving

Two DC motors with encoder

Camera



Lidar
Stepper motor
Relè

Avoid obstacles

5 ultrasonic distance sensors

Sensors internal

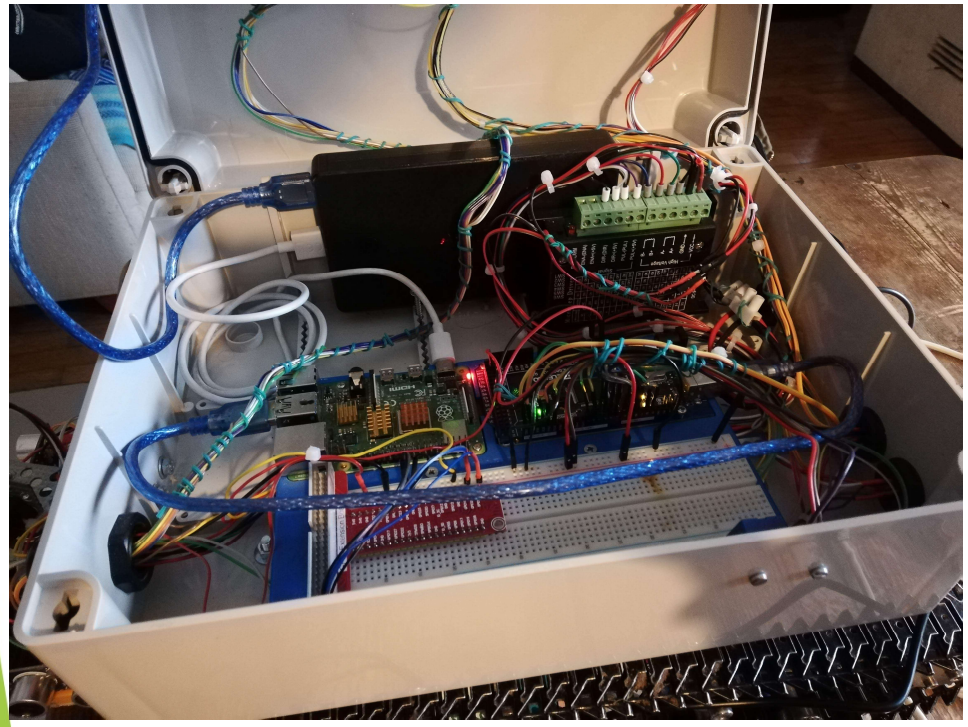
Bosh BNO055
- A compass
- An accelerometer

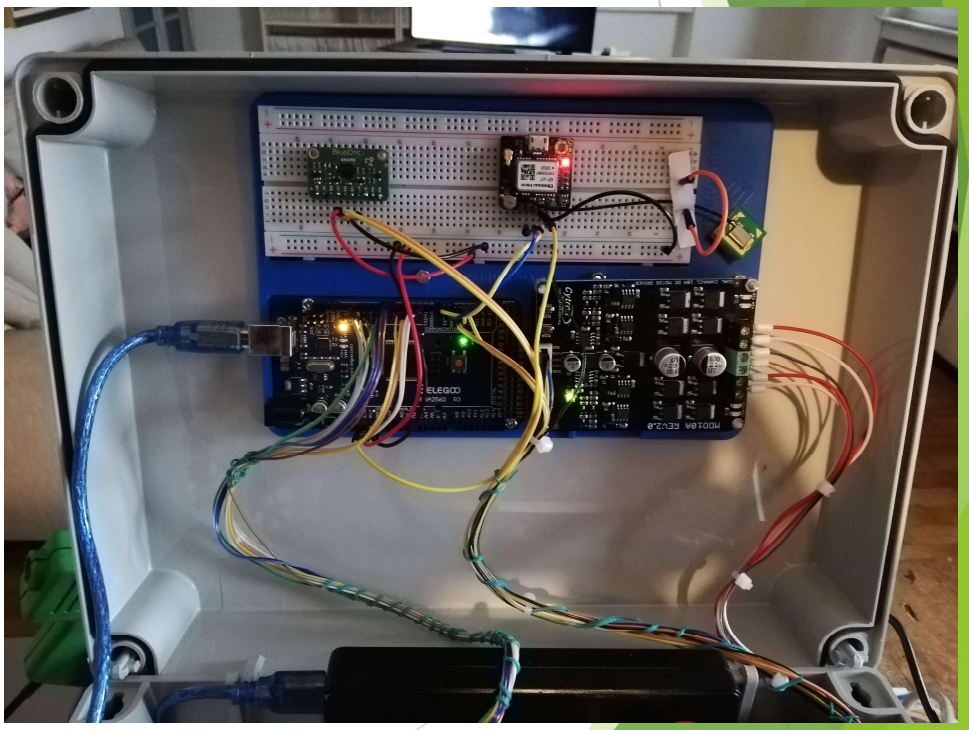
Raspberry Pi

Two Arduino

H-bridge

GPS





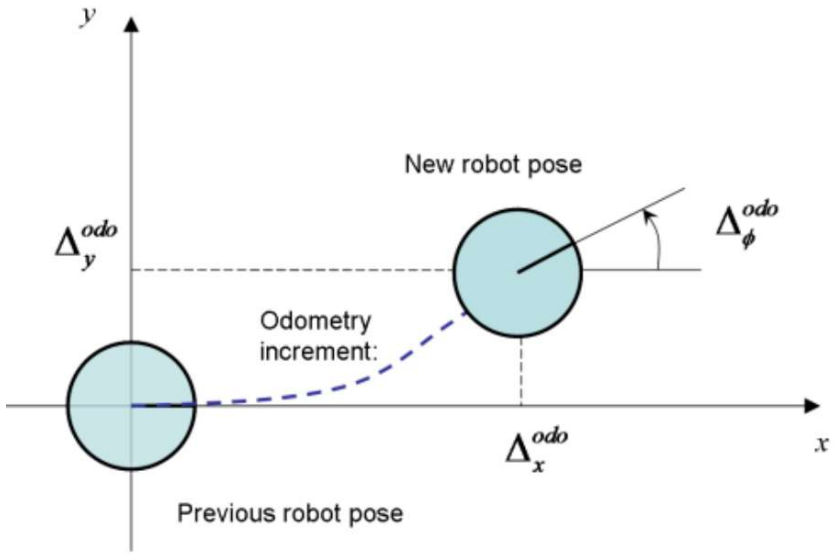
Odometry

It estimates the robot trajectory incrementally, step after step.

newstate = oldstate + last measurement

The drawback of odometry is the accumulation of errors.

The robot's position becomes less accurate as the distance travelled by the robot increases.



(F) follow a straight line = motor encoder

The robot can do two types of movements

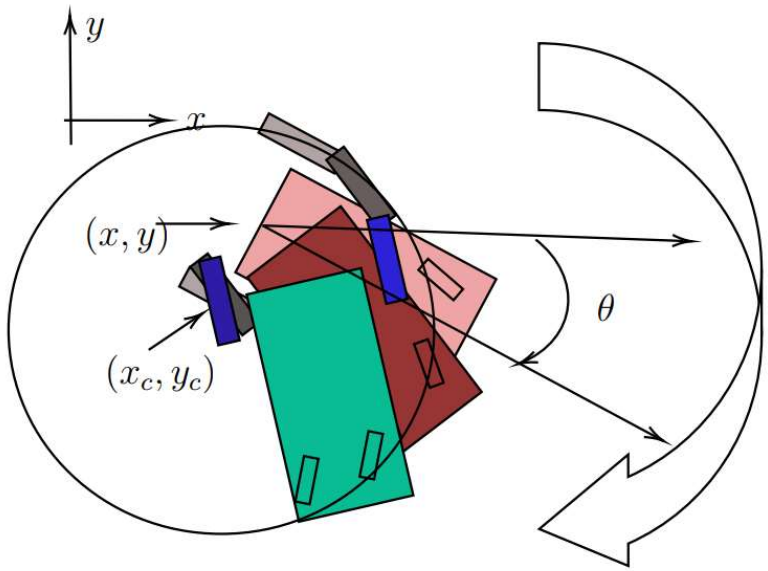
(F) follow a straight line

(R) rotate

Vehicle Description and Mathematical Model

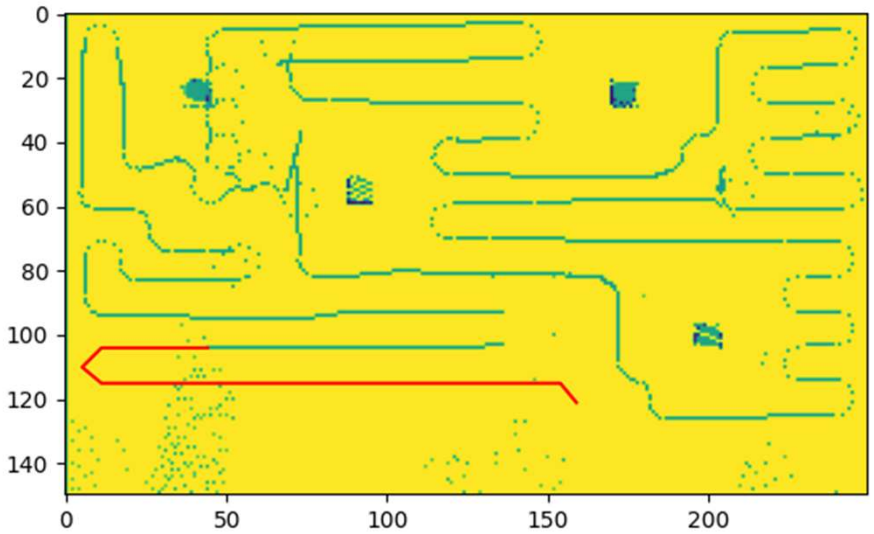
(R) rotate

```
def __turn(self, command_sent, theta_msg_grad) -> Position:
    # ======================================== #
    if command_sent == HttpCommand.RIGHT:
    # ======================================== #
        theta_msg_rad = math.fabs(math.radians(theta_msg_grad))
        center_of_rotation = self.get_right_center_of_rotation()
        c = self.get_current_position().clone() - center_of_rotation
        return c.clone()
                .rotate(theta_msg_rad)
                .translate(center_of_rotation)
```



Navigation Subsystem

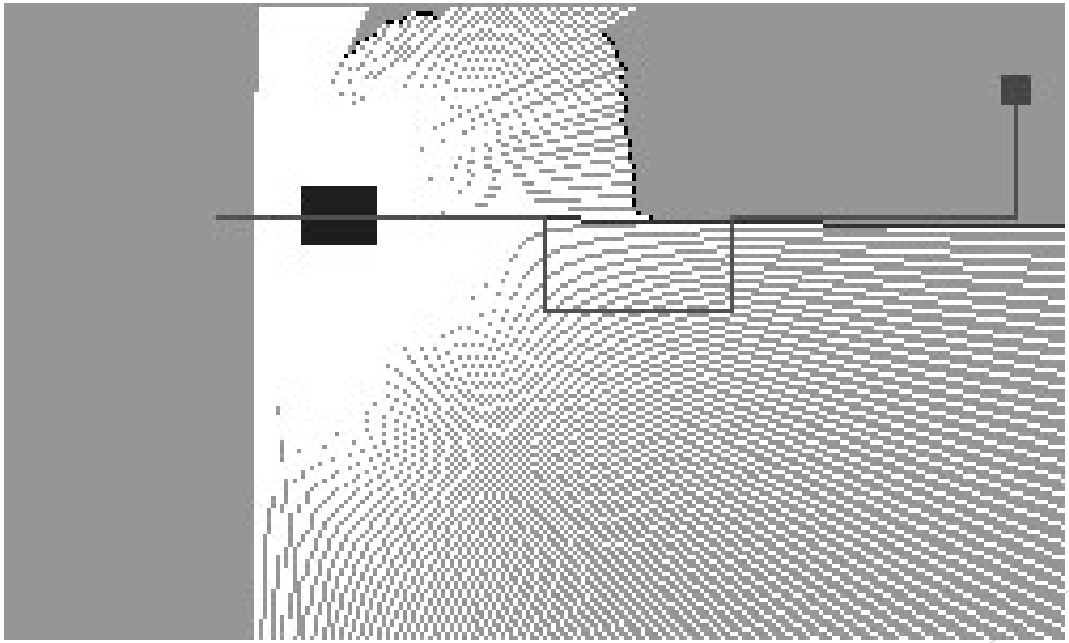
1. identifies the start point A and the end point B
2. describe and control the movement from A to B



Map and shortest path

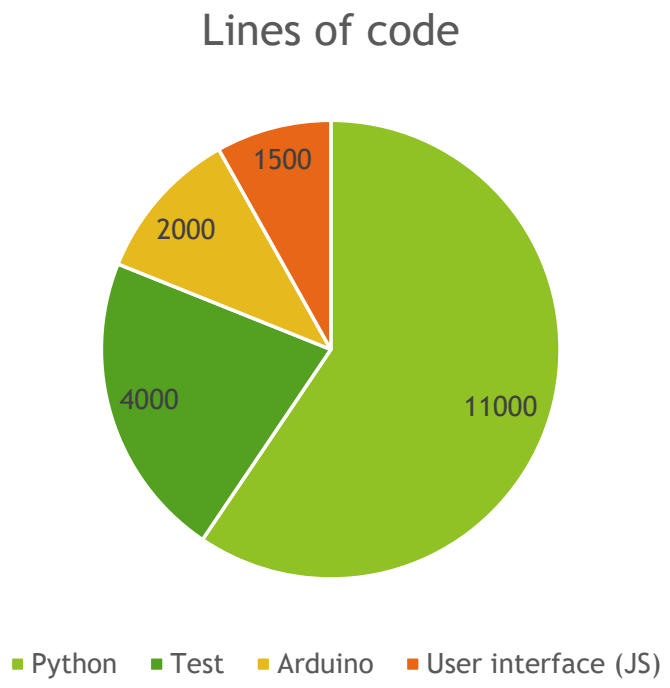
Grid-based map: This map is used to represent the environment where the robot acts. Each px of the map represents a square in the real world. The default resolution is 10 cm, so a px in the map represents a 10x10cm square.
While each cell: 255 different values the underlying structure that it uses can represent 3:

- Free
- Occupied
- Unknown



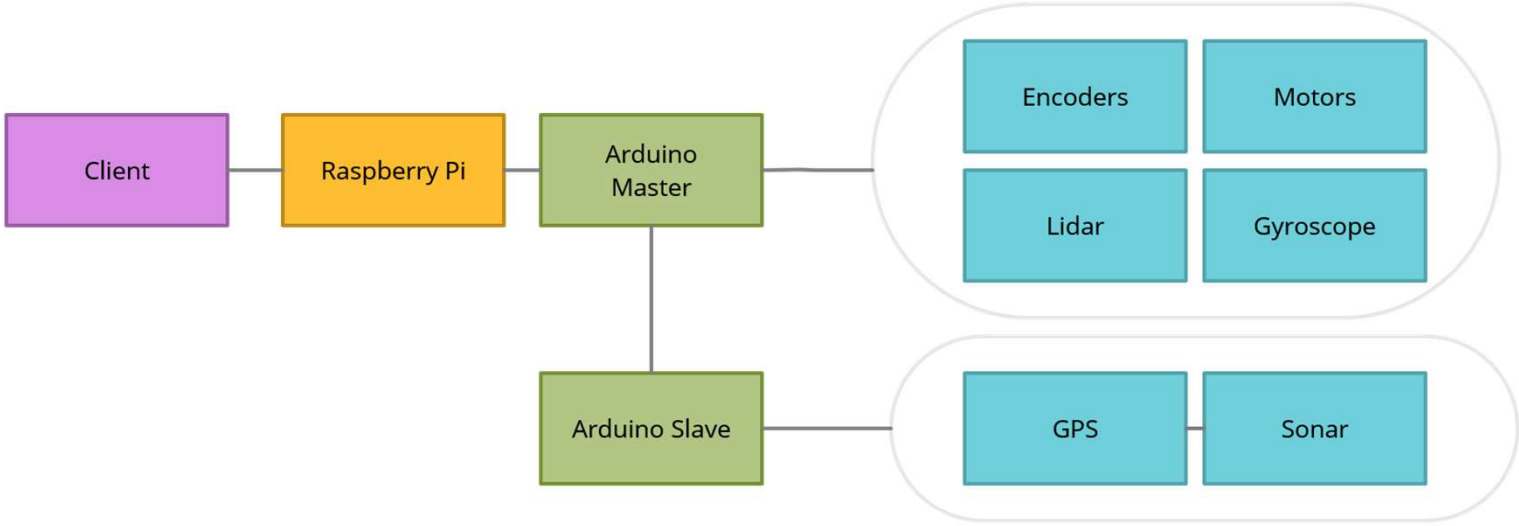
Software Implementation

Almost all the code was written from scratch. This choice allowed us to understand every aspect needed to build a robot. No specific frameworks or robotics libraries were used. This framework is around 18.000 lines of code.



System Overview

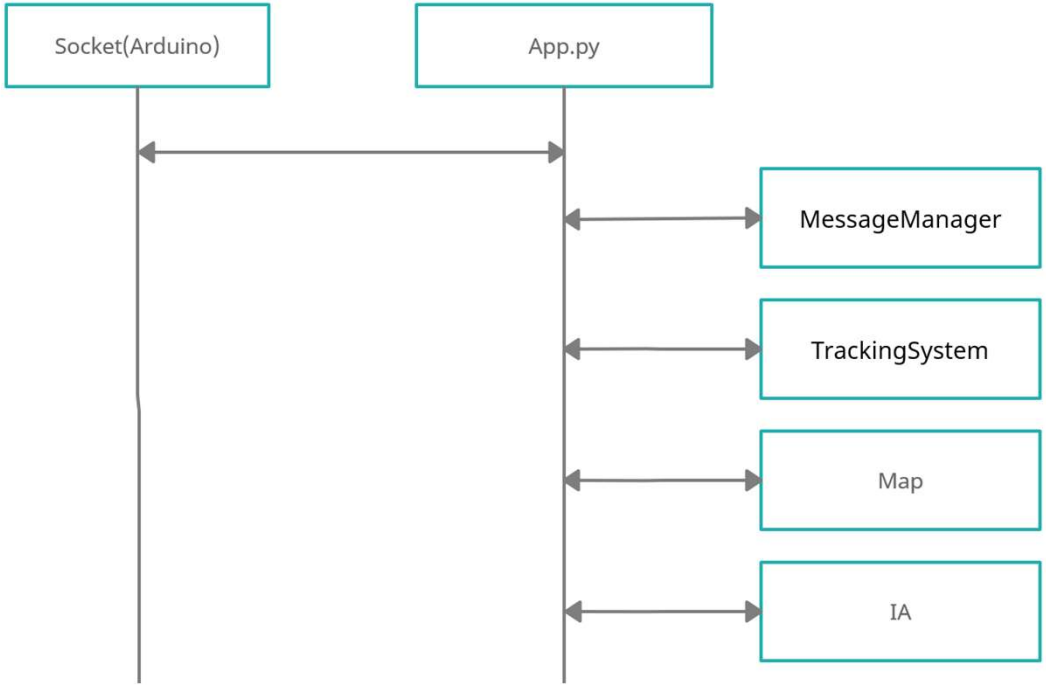
Two blocks constitute the software system.
One part is written for the Arduino microcontroller and mainly deals with sensors and motors.
The second part runs on Raspberry PI, which manages everything else. Tracking system, robot goals, maps ….



The user can easily communicate with the robot through a browser if they are both on the same LAN.

App.py - Program Overview

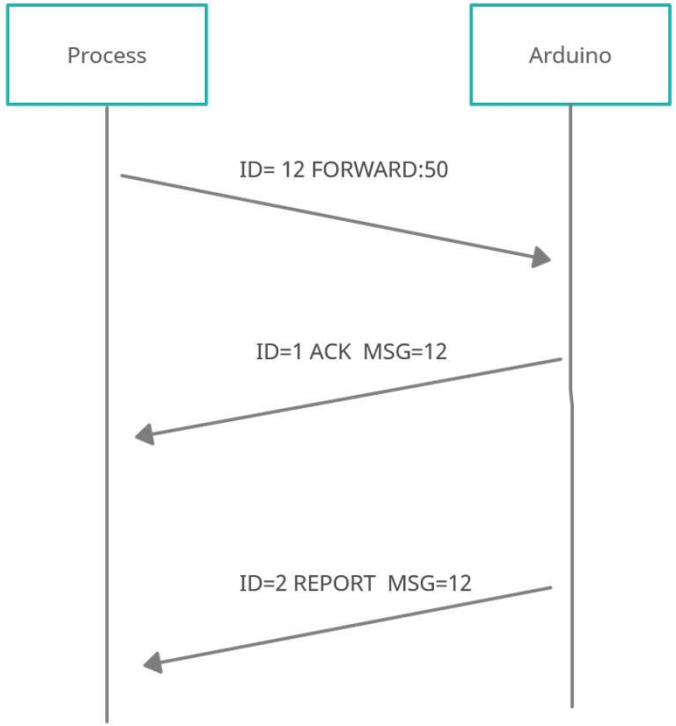
The architecture is based on the **Observer Pattern**. With this pattern, every class interested in an event, such as an Arduino message, will subscribe to it.



Messages

Managing messages between Raspberry Pi and Arduino is not an easy task.

Messages can:
- get lost from both Arduino and Raspberry Pi
- may also not be handled readily and accumulate in a queue
- error reading the message



The message in this second implementation has a parameter, for example, right-50.

Once the message has been received, Arduino will autonomously rotate 50 degrees to the right. For each message received, Arduino sends a NAK or an ACK message if it was able to decode it.

Each message has a unique ID. When Arduino responds to a command, this ID is reported. The program can understand if the command was executed.

AI

In this thesis two different strategies are tested. An interface was therefore created that allows you to easily change the main algorithm and thus test both solutions.

- Grid based coverage path planning
- e-star: An Online Coverage Path Planning Algorithm

```
1
2  class AIGeneral(ABC):
3  """
4  This class model the general inside the Robot,
5  which receives information and send out commands
6  """
7
8  def __init__(self,obstacle_map,tracking_system,mower_status):
9      self._obstacle_map: ObstacleMap = obstacle_map
10     self._tracking_system: TrackingSystem = tracking_system
11     self._mower_status: LawnMowerStatus = mower_status
12     self._is_new_plan_trajectory_available: bool = False
13     self._status = MowerStatus.IDLE
14     self._maps: List[LastMapAvailability] = []
15     # ....
```

Grid based coverage path planning

The algorithm that deals cover the area the map is divided into three parts.

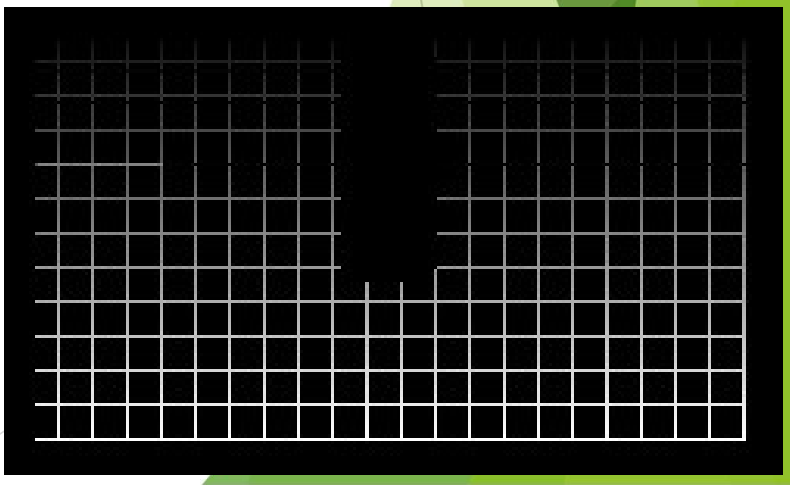
1- Potential Map

This map is for helping to plan the next goal. The idea behind this map is to divide the cells into:

- cells not visited: which have a lower potential.
- cells already visited: which have a higher potential than cells not visited
- obstacles: which have zero potential and are, therefore, not reachable

To avoid zig zags the map is divided as if it were a chessboard.

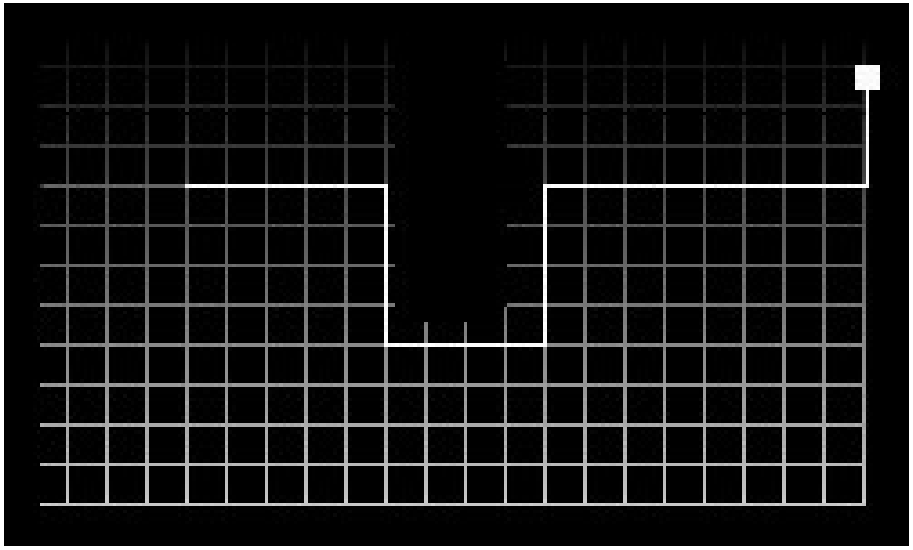
The potential correlates to the y-axis. The further away you get from the origin, the more the potential increases. The origin is in the upper left corner.



Grid based coverage path planning

2- Plan Trajectory

The robot starts on the left side then tries to reach the right side. Once the goal is achieved, a new one is calculated incrementing the "y" value. The robot will continue to go from right to left.



In these constant movements, the robot is encouraged to travel through unexposed cells, which have the lowest potential.

The a-star algorithm is used to calculate the path between the starting point and the goal.

Grid based coverage path planning

3- Calculation of Trajectory: CarrotCasing

To convert the path into robot movements, the CarrotCasing class was written

To avoid deviations due to uneven ground after many tests it was decided to advance a maximum of 40 cm at a time.

The robot needs about 11 cm to make a curve. We must therefore calculate this space when we carry out the conversion.

In the image, you can see the points that the robot must reach. The red dot after the curve indicates where the robot should be after turning.

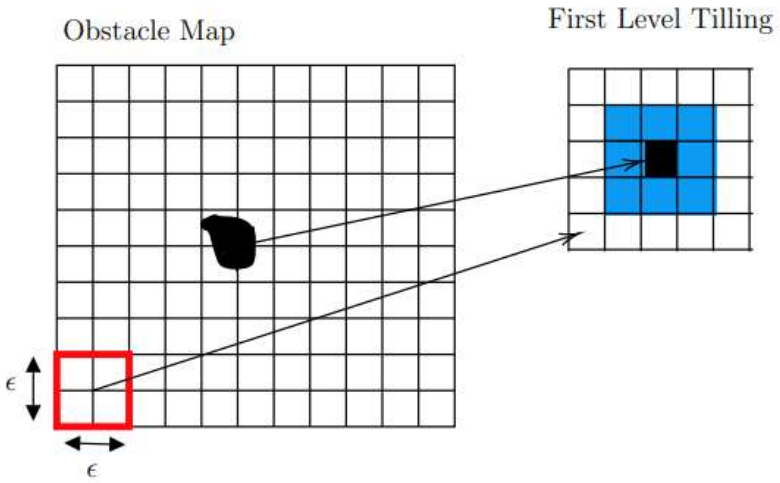
e-star: An Online Coverage Path Planning Algorithm

Briefly the algorithm is based on the Multiscale Adaptive Potential Surfaces (MAPS) model. It starts from the construction a tilling T on a map M.

This algorithm is also based on the concept of potential.

The information taken from ε cells are packed into a single cell. " ε" is the parameter needed for this algorithm.

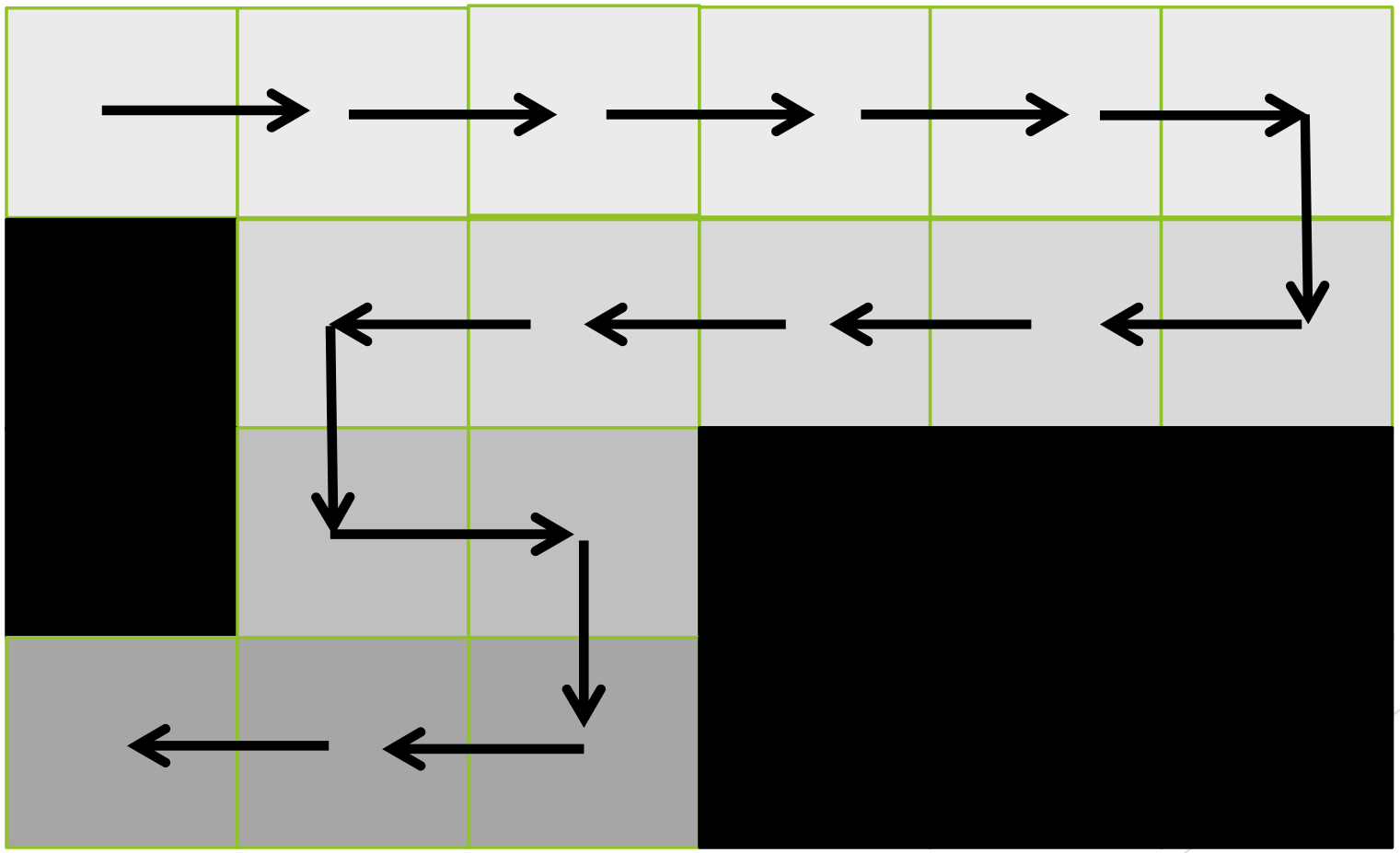
A cell can be visited, empty or contains an obstacle.



e-star

Connect the cells that have the lowest potential.

2) Find the sequence of waypoints

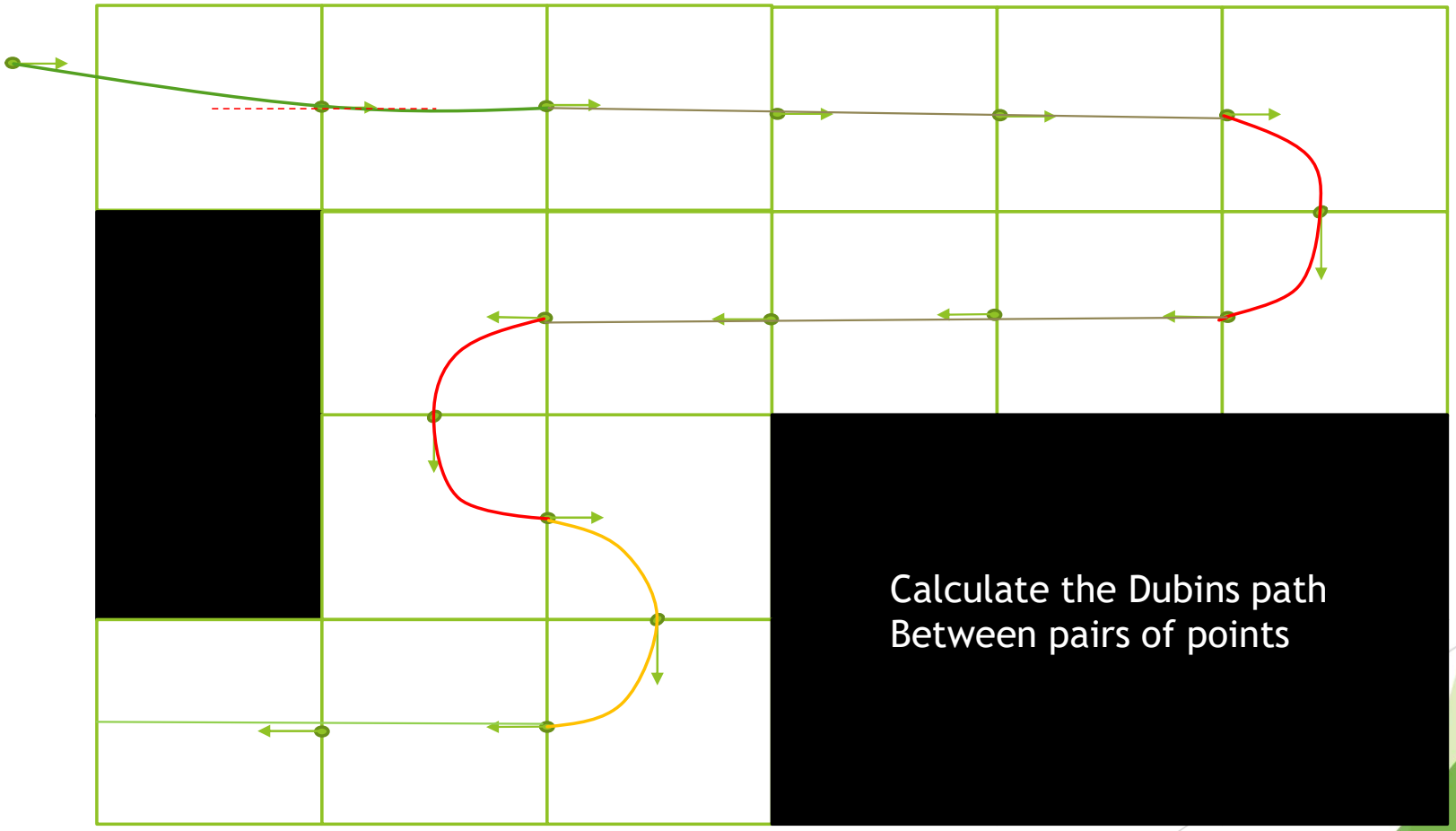


e-star

Calculation of Trajectory: Connect Middle Points

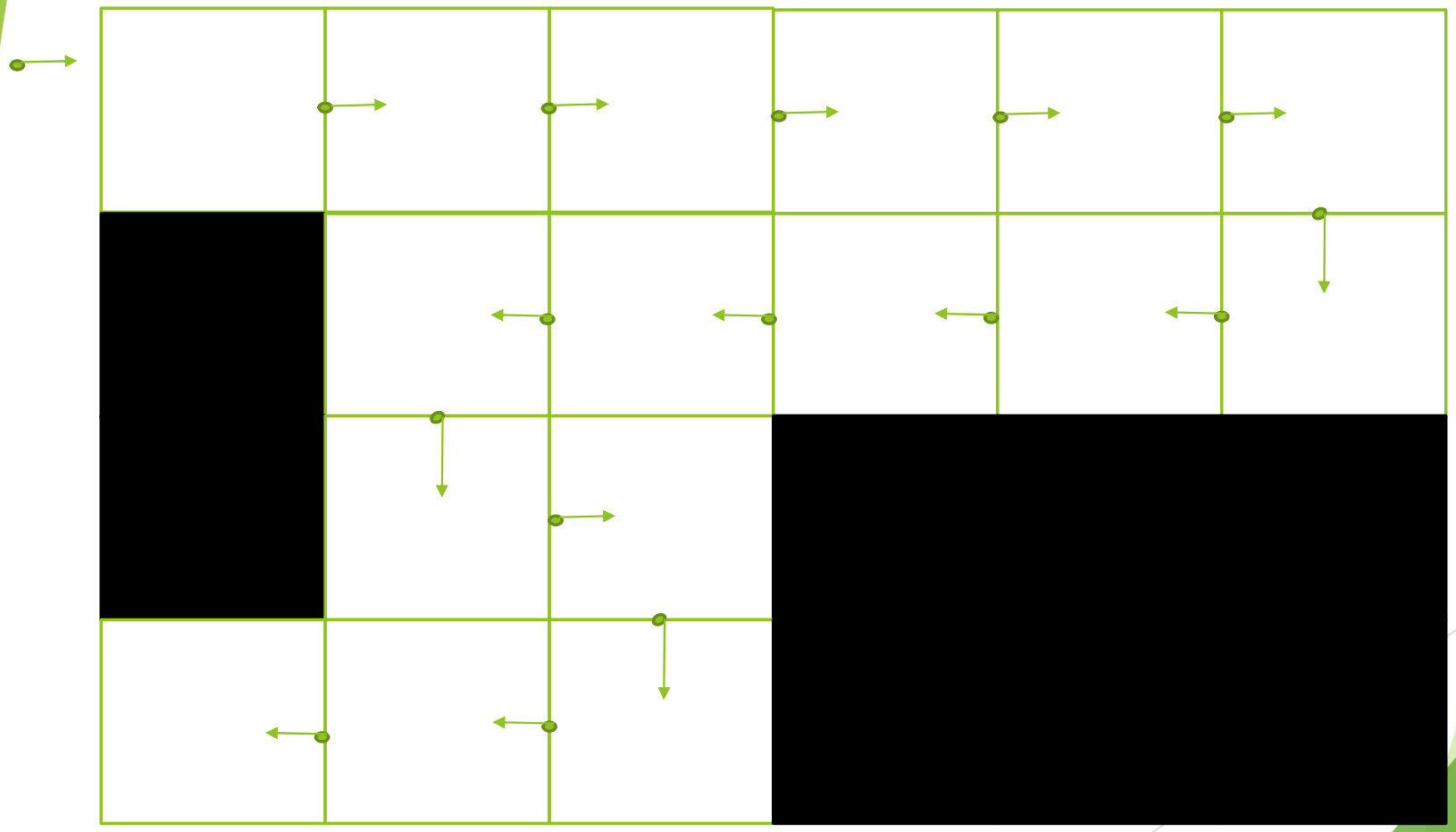
Find the intersection between the connecting line of the centers of 2 consecutive waypoints

Starting position

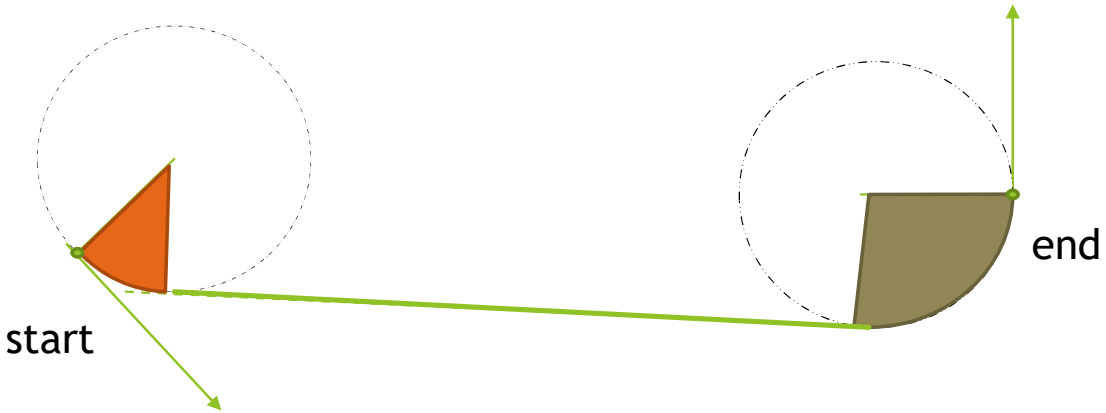


e-star

Finally, the Dubins path movements are converted into robot movements.



Trajectory: Dubins path



Algorithm

1. draw the two circumferences tangent to the starting and end point.
2. identify the tangent of the two circumferences
3. Identify the Dubins path

In geometry, the term Dubins path typically refers to the shortest curve that connects two points in the two-dimensional Euclidean plane with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path, and an assumption that the vehicle traveling the path can only travel forward.

Arduino

Two mega Arduino are used in this project.

Arduino master does practically all the work required to be able to command a rover. Among his tasks, we find:

- Read the commands that come from Raspberry Pi
- Read the sensor values
- Go forwards and backwards or turn according to the command
- Balance the speed to try to go straight
- Communicate with Raspberry Pi and send the collected data.

User Interface

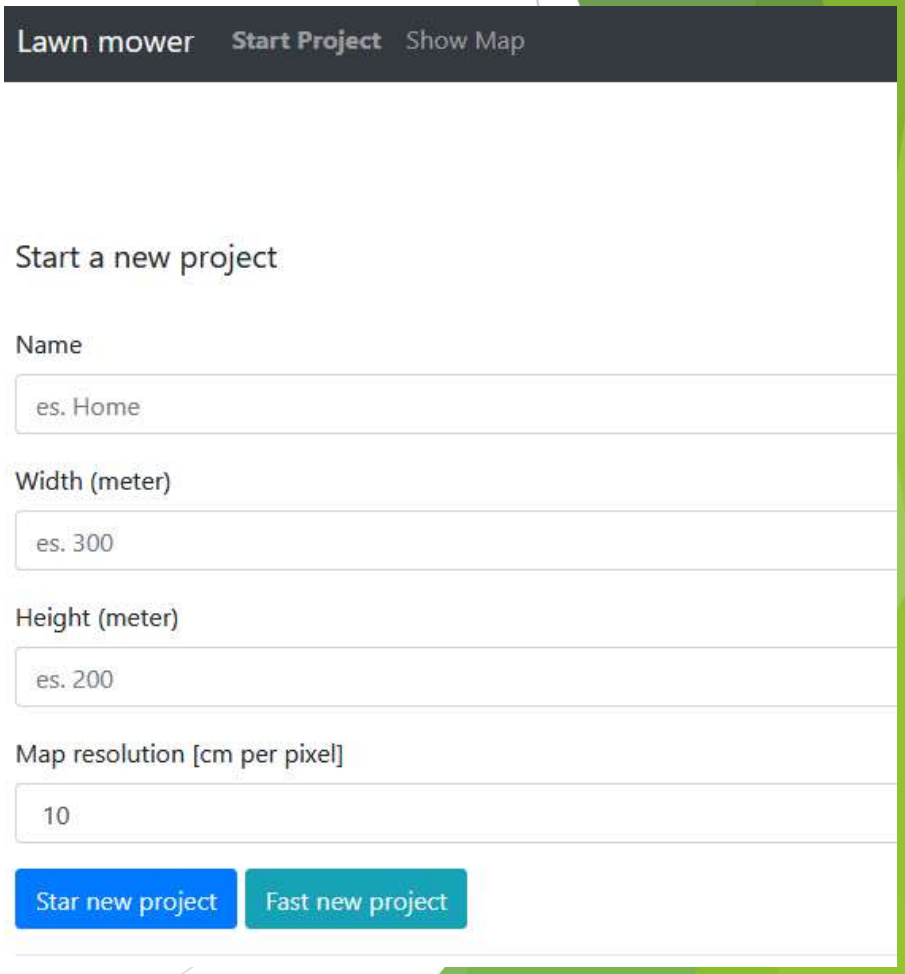
To allow the dialogue between the application running in the robot and the user, two simple web pages have been created. These allow you to start the mission and check its progress.

New Map

This page offers the user three choices:

- Create a new map
- Load an old map (from disk)
- Upload a custom map

If you want to create a new map, you can specify both the length and the height. Another customizable aspect is the resolution, by default set at 10cm = 1px

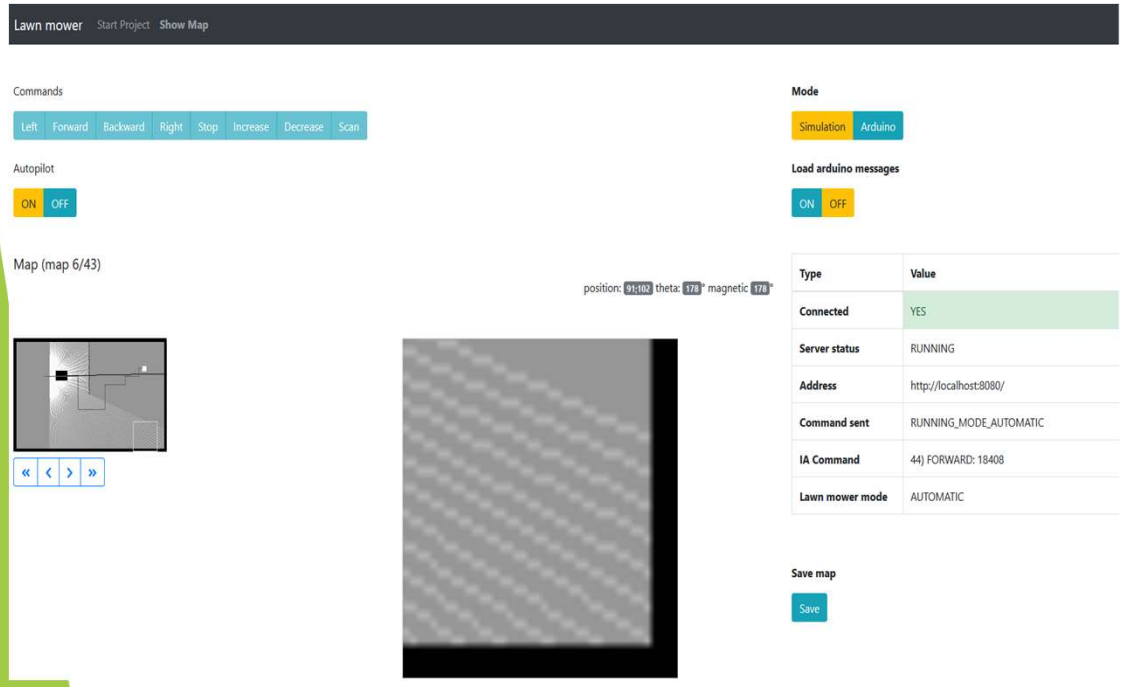


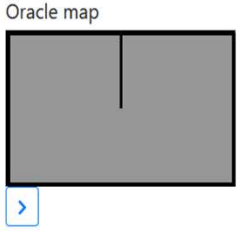
User Interface

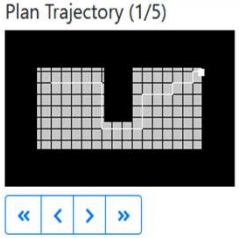
Dashboard

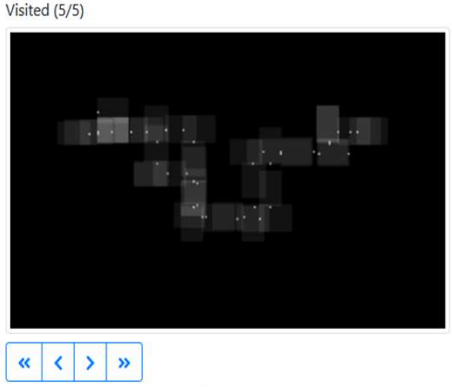
This page allows to control the robot and shows all the actions made by the robot.
There are several maps: obstacles map, trajectory map, visited map and potential map.
The robot can run manual and automatic.
In the UI, the buttons allow driving the robot manually.

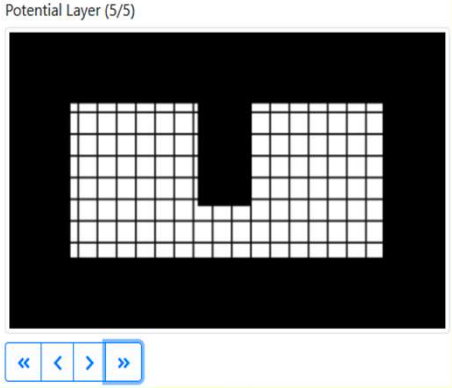
There is a polling timer set to 1s. The web server responds with all updated information in a JSON format. This information contains, for example: the last command sent, plan strategy, update map, ...





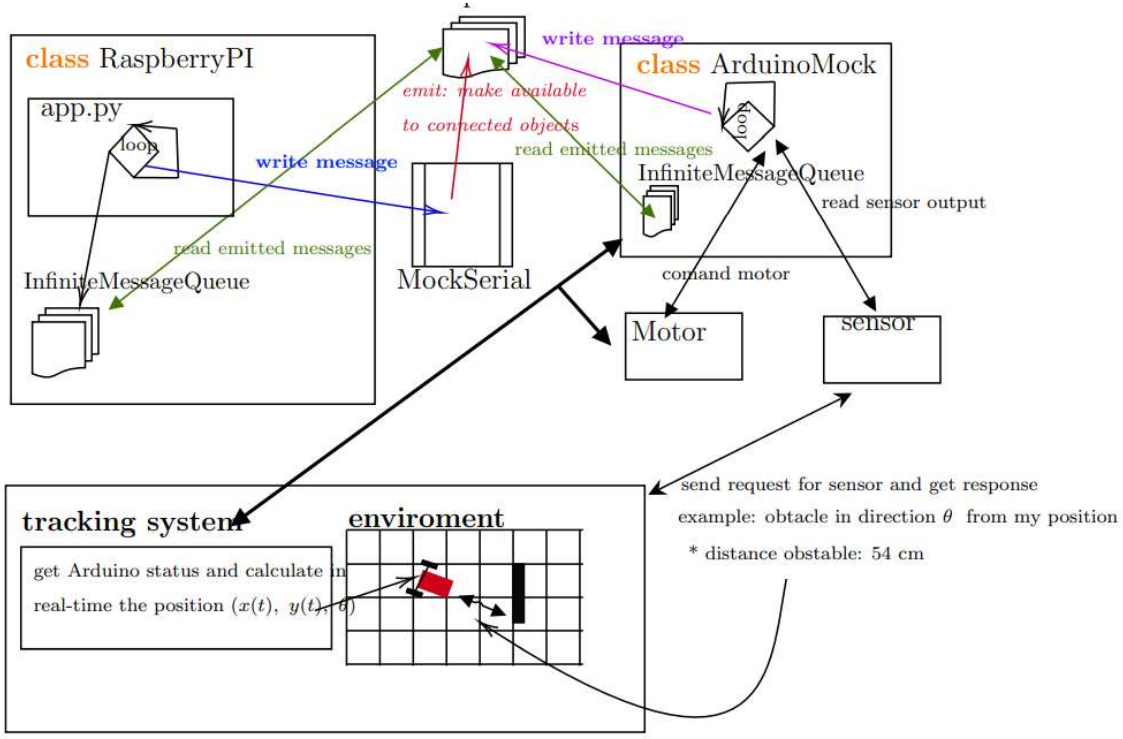






Mower Simulator

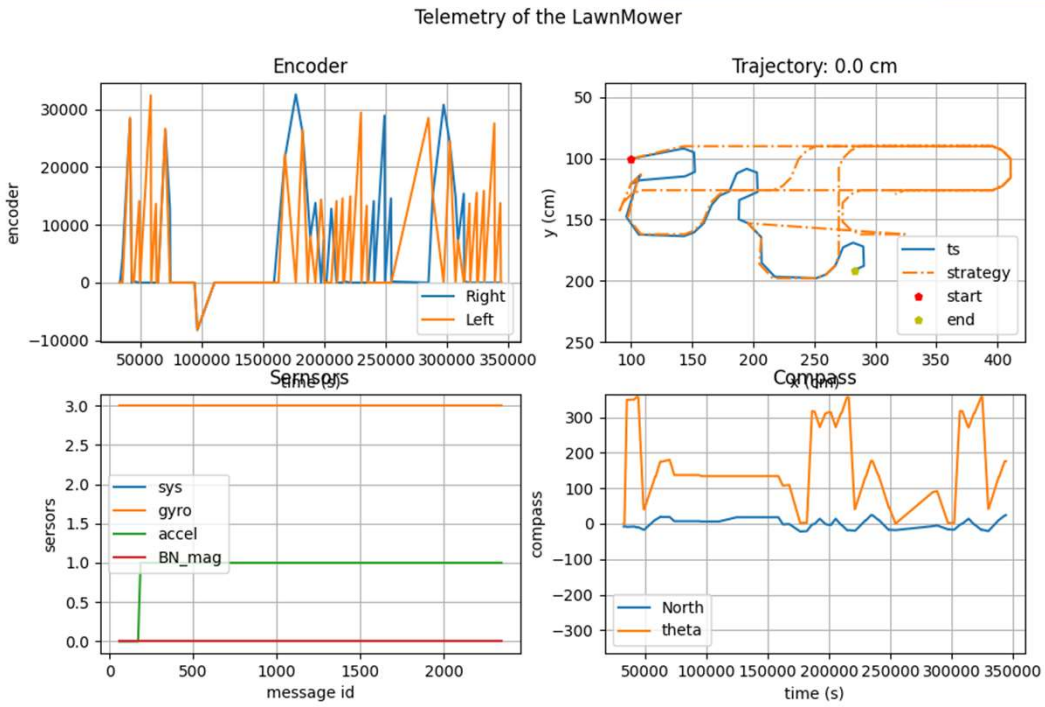
To speed up development times and to be able to debug the code, a robot simulator was created. The simulator receives the commands and moves the robot virtually on the map. All sensors are simulated to provide adequate response messages. A fundamental parameter is the amount of **noise** present in the data collected by the sensor to be able to obtain sufficiently robust algorithms.



Software Implementation

At the end of the mission, it is possible to analyse the robot's behaviour and choices made.

- Mission Analysis: Report
- Mission Analysis: Telemetry
- Logs



Tankerino report

Gridbase - empty map - simulation

Messages overview

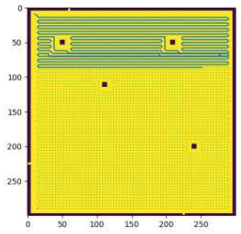
Duration:	00:04:44	Messages sent	374	Message received	1141
Report	362	Ack	374	Nak	0
Count turn	49	Left	8	Right	41

Simulation and Experiments

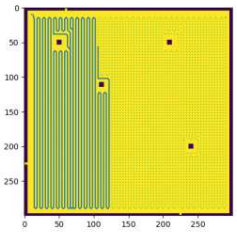
The simulations can be classified in two different categories:

- **Ideal Mower**: AI without any imperfection: useful to validate the implemented algorithms.

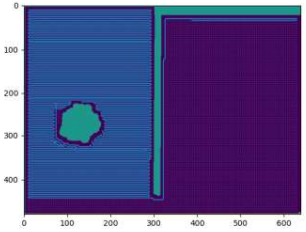
- **Realistic Mock**: the messages and responses are generated by the simulator



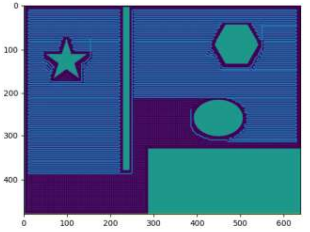
(a) random forest horizontal (75 x 75)



(b) random forest vertical (75 x 75)



(c) Bush and wall (120 x 160)



(d) geometric garden (120 x 160)

maps	(a)	(b)	(c)	(d)
size	75 x 75	75 x 75	120 x 160	120 x 160
visited	4820	4820	16484	12783 (-8)
obstacle	805	805	2716	6409
visitable	4820	4820	16484	12791
length path	4836	4838	16580	13524
direction changes	220	223	548	972

Table 5.1: Results of the experiments to evaluate and improve the algorithm $\epsilon^{\star}$

The results of these experiments are encouraging, the strategy cover almost the complete keeping the length of the path close to the lower limit.

Experiments on the Field

Initially, we started with a simple map 5 meters by 3 meters with no obstacles

Experiments on the Field

https://www.youtube.com/watch?v=Bv_9Sx9DnPg

Measures

- The first operation carried out was to check that the formulas used in the simulator are corrected.

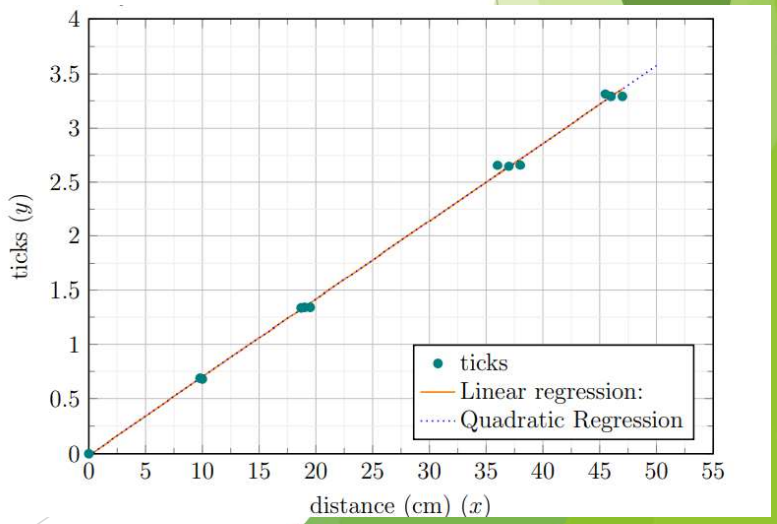
Wheel

Tick

Turn start

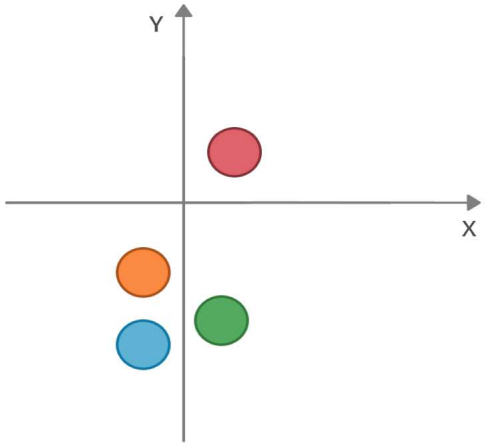
Turn end

ms id	planned cm	measured cm	json parameter	tick Left	tick Right	Mean
5	10	10	6550	6804	6793	6798,5
7	10	9,8	6550	6876	6878	6877
11	20	19,5	13100	13461	13443	13452
13	20	19	13100	13450	13450	13450
15	20	18,7	13100	13426	13381	13403,5
17	40	38	26200	26587	26588	26587,5
19	40	37	26200	26485	26477	26481
21	40	36	26200	26572	26571	26571,5
27	50	45,5	32750	33135	33142	33138,5
31	50	47	32750	32914	32906	32910
32	50	46	32750	32914	32906	32910



It was necessary to find the correlation between the motor tics and the cm travelled by the robot.

1 - Maintaining Straight Path and Executing 180° Turn



Mission	X	Y
1	7	-9
2	-5	-7
3	9	4
4	-8	-11

1. Run the vehicle for four meters, in ten steps of 40cm
2. Make a 180 degree turns on the spot
3. Return to the starting area
4. Compare the actual position with the calculated one.

- The experiment is running four times. The result is the following graph:

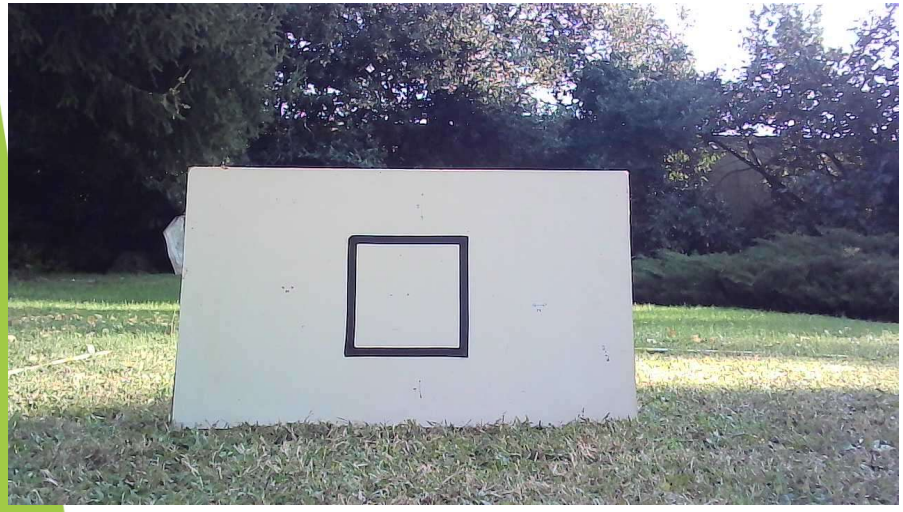
As result the encoders behave **quite accurately.**

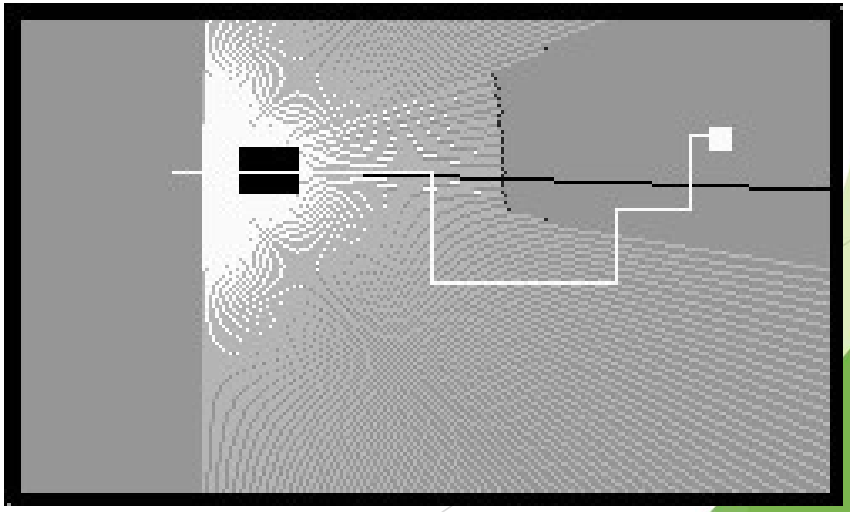
2 - Obstacle detection

Another early experiment was recognizing obstacles and then deflecting strategy accordingly. To recognize obstacles, the robot could use in this experiment the ToF (Time of Flight) LiDAR

A wooden panel is placed at 160 cm, which simulates the obstacle to be overcome. The following photo is taken by the camera mounted on the robot.

The panel can be clearly recognized in the image, the scanner works well. The lidar sensor is not perfect and introduced some noise. This problem will have to be solved in the future.





Comparison of implemented AI-Algorithms

For the final experiment, we compare the result of the two developed in the simulation and with the real robot. The experiments are carried out on an empty map and on a map with an obstacle.

Report name	Mode	Map type	Algorithm
1	simulator	blank map	grid based coverage path planning
2	simulator	blank map	e-star
3	simulator	map with obstacle	grid based coverage path planning
4	simulator	map with obstacle	e-star
5	mower	blank map	grid based coverage path planning
6	mower	blank map	e-star
7	mower	map with obstacle	grid based coverage path planning
8	mower	map with obstacle	e-star

Comparison of implemented AI-Algorithms

The two algorithms developed are very different from each other.

- E-star is much more innovative when it comes to handling complicated map situations. Its purpose is to visit the map as quickly as possible by optimizing the route.
- On the other hand, the ”Grid-based coverage path planning” algorithm is much simpler as it tries to go from left to right in the map until it is completely covered.

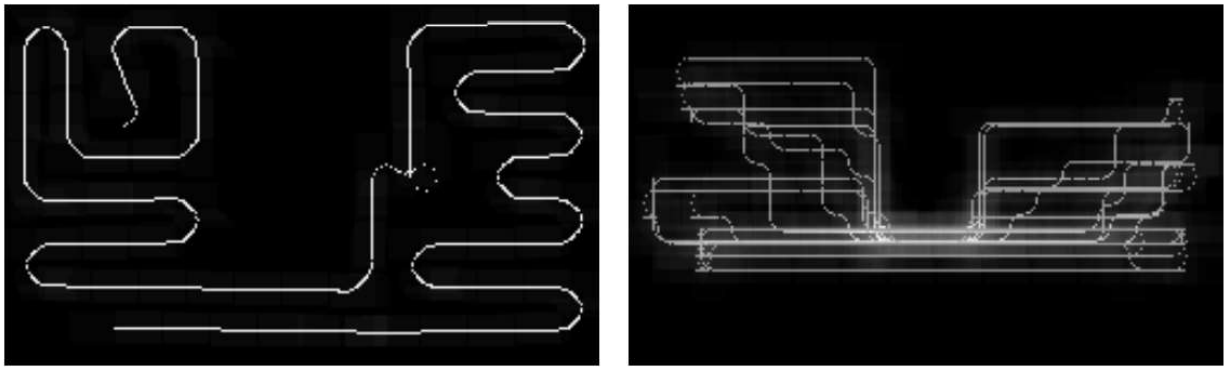


Figure 5.12: The two images compare the two algorithms in the obstacle map in the simulator.

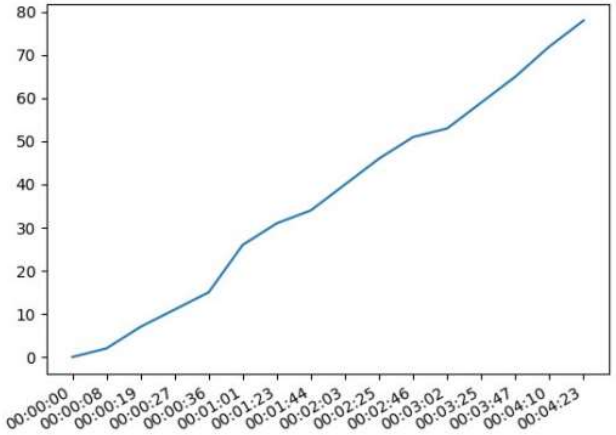
empty map - simulation

In the simulation, both algorithms do a great job visiting over 80% of the area.

Grid-based coverage path planning

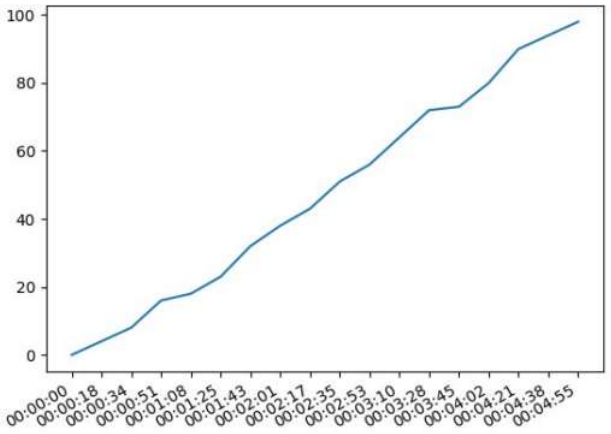
Messages overview

Duration:	00:04:44	Messages sent	374	Message received	1141
Report	362	Ack	374	Nak	0
Count turn	49	Left	8	Right	41
Count forward	254	meters:	98	Count alive	393
Count backwards	26	meters:	2	Count Stop	1
Count scan	12	Angular correction	29	Report Error	0



E-star

Duration:	00:04:59	Messages sent	348	Message received	1069
Report	311	Ack	347	Nak	0
Count turn	79	Left	36	Right	43
Count forward	163	meters:	57	Count alive	375
Count backwards	0	meters:	0	Count Stop	57
Count scan	36	Angular correction	10	Report Error	0



Gridbase - empty map – real mower

These are the data that can be read from the report.

Messages overview

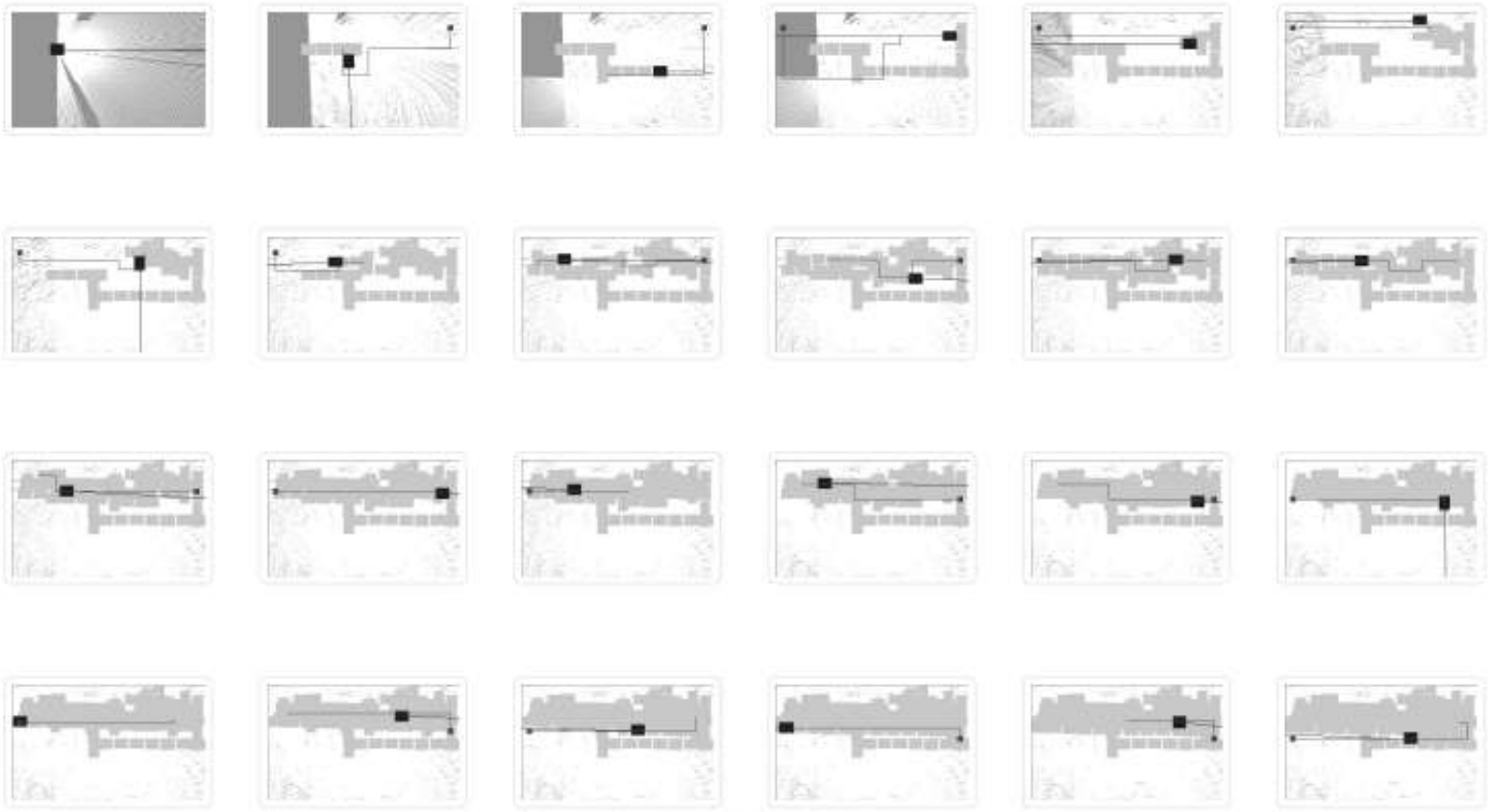
Duration:	00:36:12	Messages sent	379	Message received	1308
Report	304	Ack	329	Nak	49
Count turn	87	Left	49	Right	38
Count forward	178	meters:	63	Count alive	611
Count backwards	38	meters:	4	Count Stop	3
Count scan	19	Angular correction	51	Report Error	0

In the field the real robot worked for about 35 minutes, travelled about 60 meters.

Gridbase - empty map – real mower

Maps overview

In these maps, it’s possible to see how the noise added by the lidar makes the robot change its trajectory.

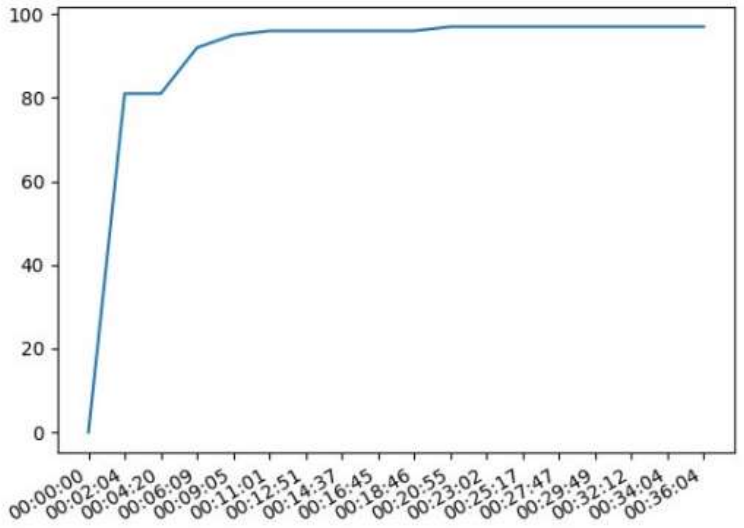


Gridbase - empty map – real mower

The map is all explored

id	time	unknown	explored	obstacle	not set	percentage
0	00:00:00	37500	0	0	0	39
2	00:00:18	16546	20926	22	6	55
4	00:00:40	13105	24356	18	21	64
6	00:01:02	11041	26395	26	38	70
12	00:02:02	7047	30377	28	48	81
36	00:05:21	3568	33818	11	103	90
73	00:10:02	1111	36264	2	123	96
316	00:36:02	704	36639	4	153	97

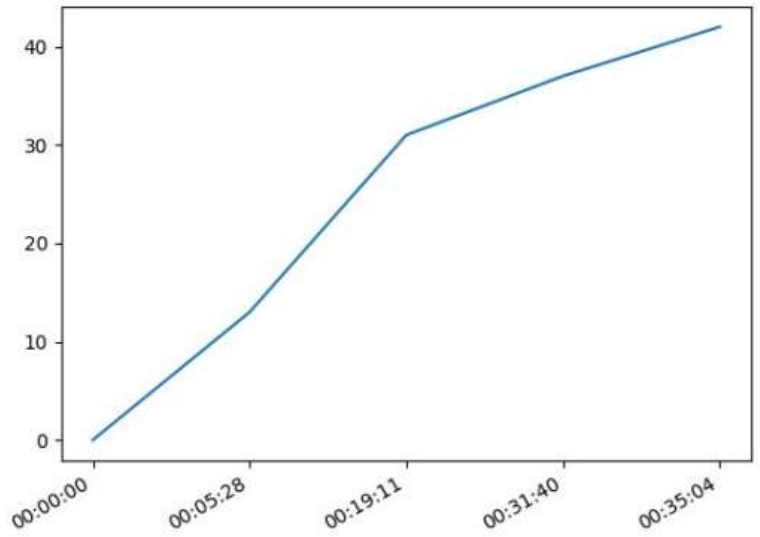
Explore map percentage



Gridbase - empty map – real mower

About 30 percent of the map is visited.

Visited map percentage



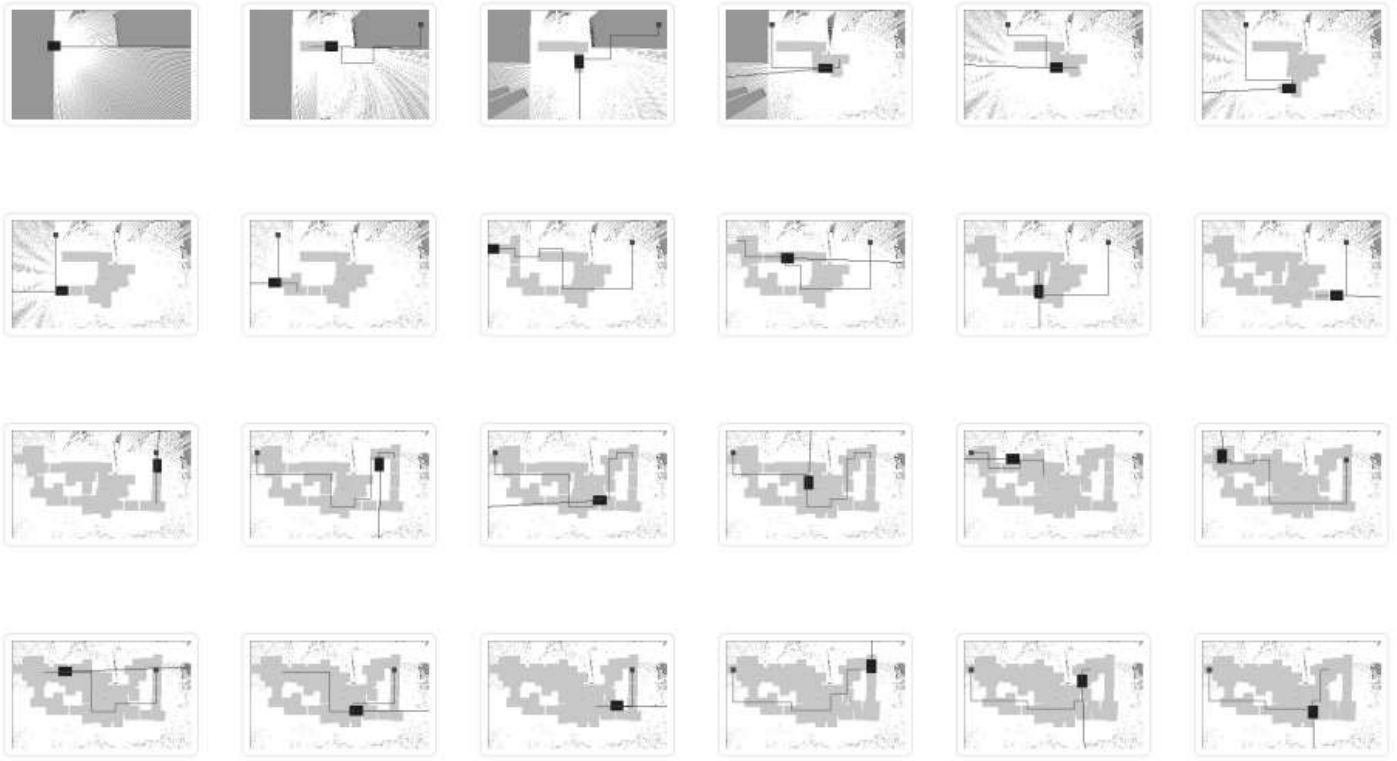
ID	Time	visited	not visited	obstacle	percentage
1	00:00:00	0	37496	4	0
10	00:03:46	3610	33886	4	9
20	00:06:30	5588	31908	4	14
30	00:13:33	10971	26525	4	29
40	00:23:28	12514	24982	4	33
50	00:31:40	14160	23336	4	37
56	00:35:04	16075	21421	4	42

Gridbase – map with an obstacle – real mower

Messages overview

Duration:	00:36:12	Messages sent	379	Message received	1308
Report	304	Ack	329	Nak	49
Count turn	87	Left	49	Right	38
Count forward	178	meters:	63	Coun	
Count backwards	38	meters:	4	Cour	
Count scan	19	Angular correction	51	Repc	

In the map with an obstacle, the result is practically comparable.



Conclusions

The project goals were not fully achieved. Nevertheless, the final result is remarkable.

The mower can perform up to 20 minutes an optimal way. During those 20 minutes, the robot visited 30% of the total area without going out of the edges or hitting any obstacle. **But then the sum of many minor errors affects the robot's location.**

After about 20 minutes and up to about 30 minutes, it is possible to see some localisation errors, which could still be considered acceptable. After approximately 30 minutes, the robot begins to exit the test area.

This problem, well known in the literature, can be avoided by implementing **SLAM techniques**. The cheap and not always accurate sensors increase the problem of localization.

Writing all the code forces us to deal with many obstacles. For example, there were both software problems and hardware difficulties. Building a robot framework was not easy; real-time applications are usually complicated.

The future works

- The future works to be implemented are using a SLAM algorithm and moving some part of the logic in Arduino to speed up the robot.
- Another way forward to help the robot locate it could be the implementation of DenseDepth algorithms.

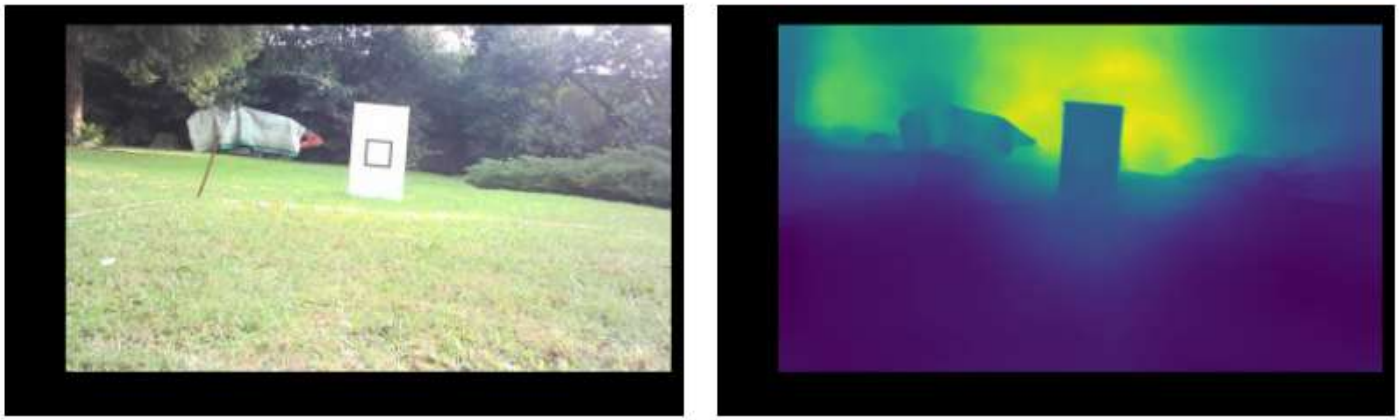


Figure 6.2: Result of the image with a Dense Depth algorithm taken by the robot camera. The program can detect the obstacle by its different colours.

Thank you for your attention