Technische Hochschule Deggendorf- Campus Cham

MSS-M-2: Case Study Autonomous Systems (WS22/23)

Groups Name:

* La tortuga
* Team Rocket

Turtlebot3 Automatic Parking

Real-world and ROS simulation

# Teams Members

Contents

[Project overview 2](#_Toc125905112)

[SLAM 2](#_Toc125905113)

[Navigation 2](#_Toc125905114)

[Project Implementation 4](#_Toc125905115)

[Testing environments 5](#_Toc125905116)

[The code structures 6](#_Toc125905117)

[Wall follower 7](#_Toc125905118)

[Mapping and goal finding 11](#_Toc125905119)

[Trajectory planning 13](#_Toc125905120)

[Path following and Park 14](#_Toc125905121)

[Code Integration 15](#_Toc125905122)

[Results 16](#_Toc125905123)

# Project overview

The objective of this project is to implement the automatic parking for turtlebot3 using self-written python functions. The main steps needed to achieve this goal is illustrated as below:

## SLAM

The SLAM (Simultaneous Localization and Mapping) is a technique used in robotics to create a map of an unknown environment while simultaneously keeping track of the robot's location within that environment. The goal of SLAM is to create a map of the environment with the goals which can be used by the Navigation to generate the trajectory.

The SLAM step achieved by using:

* LIDAR sensor: uses the laser beams to measure the distance to surrounding objects.
* Reflecting tape: used to define the goal since the reflective tape has a higher intensity which will help to differentiate the goal from the other object in the environment.
* Matplotlib: to draw the maps using the points detected by the LIDAR sensor

## Navigation

The goal of the step is to find the optimal trajectory (the shortest trajectory) then go to the parking slot. For implement the trajectory planning, three different methods were studied:

* Move-base packages in ROS
* RRT and RRT\*: Rapidly-exploring random tree
* I-RRT\*: Informed Rapidly-exploring random tree

The table below summarize the pros and cons of each method:

| The method | Pros | Cons |
| --- | --- | --- |
| Move-base packages | Large scale environment  Dynamin environment  Global and local planning | Predefined packages by ROS- out of the project objectives  Complex to set up, configure, and debug using python |
| RRT and RRT\* | -Memory efficient  -Flexible path planning  -Controllable parameters | -Take more time to converge to optimal path.  -Trajectory path is not optimized. |
| I-RRT\* | -Improvement over RRT\*  -Take less time to converge  -Probabilistic node sampling.  -Ellipsoidal heuristic planning  -Rewire existing nodes.  -Algorithm parameters are adjustable | -Sometime take more time to find first path due to random nodes. |

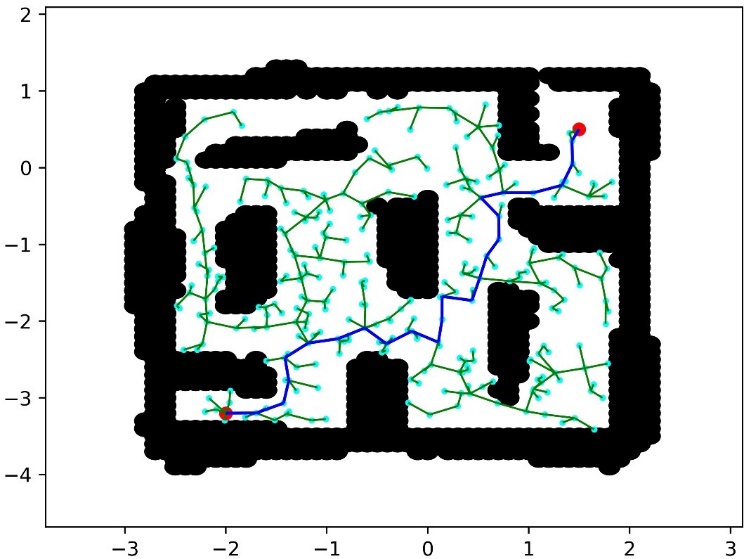
The pictures below show the performance of RRT\* and I-RRT\* for a pre-generated map of Plaza environment in gazebo.

Figure 1 RRT\*

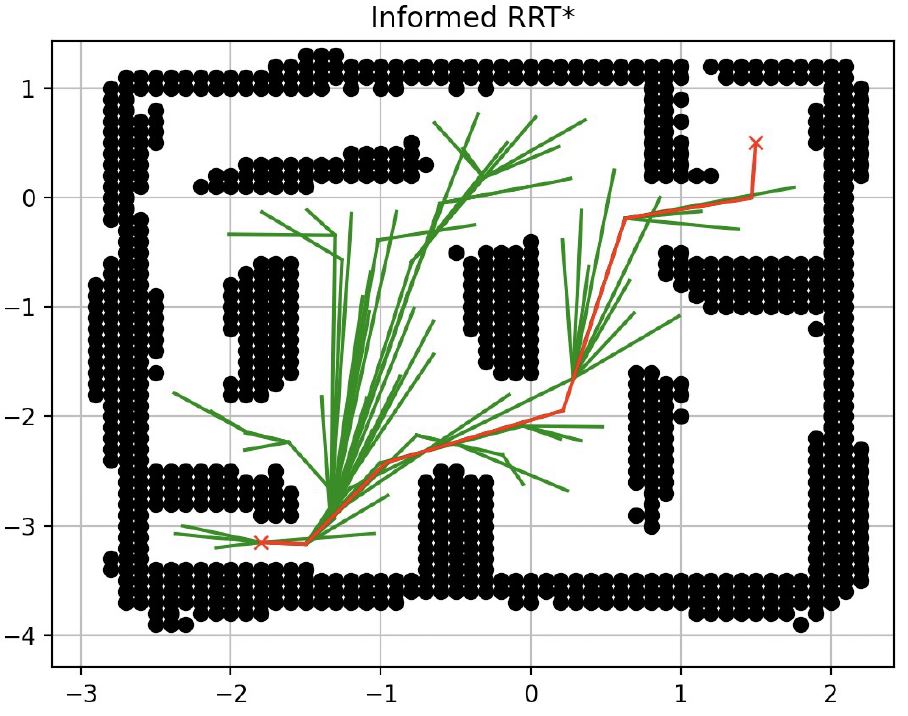
For finding the optimal trajectory, RRT\* took approximately 2 minutes while I-RRT\* took less than 1 minutes, subsequently I-RRT\* is used for trajectory planning.

Figure 2 Informed RRT\*

After that, the generated trajectory will be used to navigate to the goal.

Now, before moving to the project implementation, the robot specification should be reviewed.

Robot model: Turtlebot3 burger

|  |  |  |
| --- | --- | --- |
| Items | Values | Notes |
| Maximum translational velocity | 0.22 m/s | For velocity publish |
| Maximum rotational velocity | 2.84 rad/s (162.72 deg/s) |  |
| Size (L x W x H) | 138mm x 178mm x 192mm |  |
| LiDAR range | 360 degrees | The range is not always 360 |

# Project Implementation

The project was divided to 5 main functions, as shown in the illustration below:

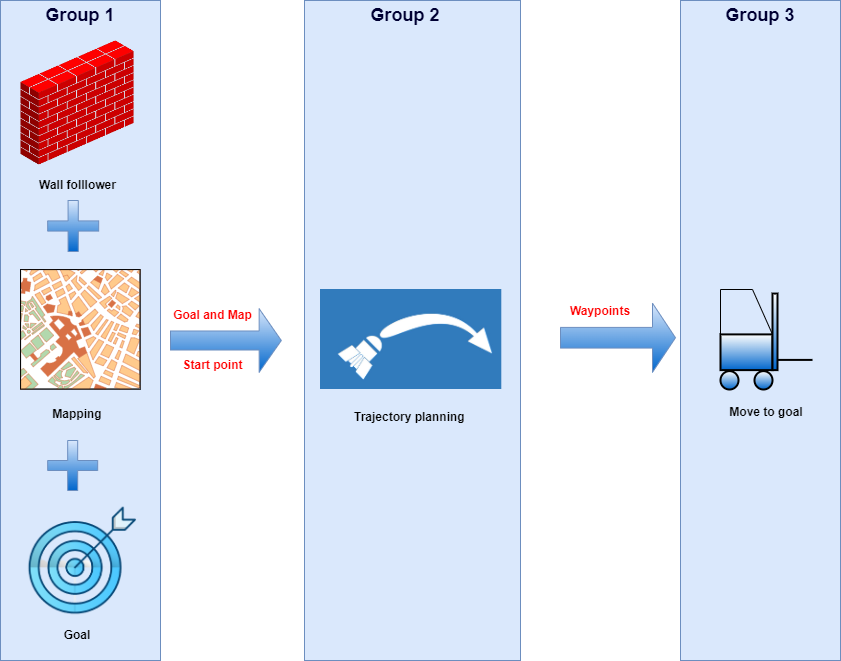


Figure The project main functions

At the beginning the robot will start to follow the wall which is away to do the scanning, in parallel the LiDAR values are read and saved to create the map each point in the map has 3 attributes X,Y coordinates and intensity. Based on the intensity values the goals will be defined.

Starting point, goals and the map for the environment are the output of the first group which will be used to create the trajectory by the trajectory planning function.

Eventually, the waypoints generated by the trajectory planning will feed to the point-to-point function to move the robot towards the goal.

However, one of the biggest challenges for project implementation was reflective tape intensity. This is due to fact that both wall and reflective tape have the same intensity range. It was also difficult to implement the intensity testing in the simulation environment.

To tackle this issue, black box is used instead of the reflective tape. Now instead of finding the goals based on high intensity values, lower intensity values are considered as a goal.

The picture below shows an example for intensity plotting using black box and reflective tape.

## Testing environments

As previously mentioned before, the project code was tested for debugging using gazebo simulation and real test environment.

The pictures below show these environments.

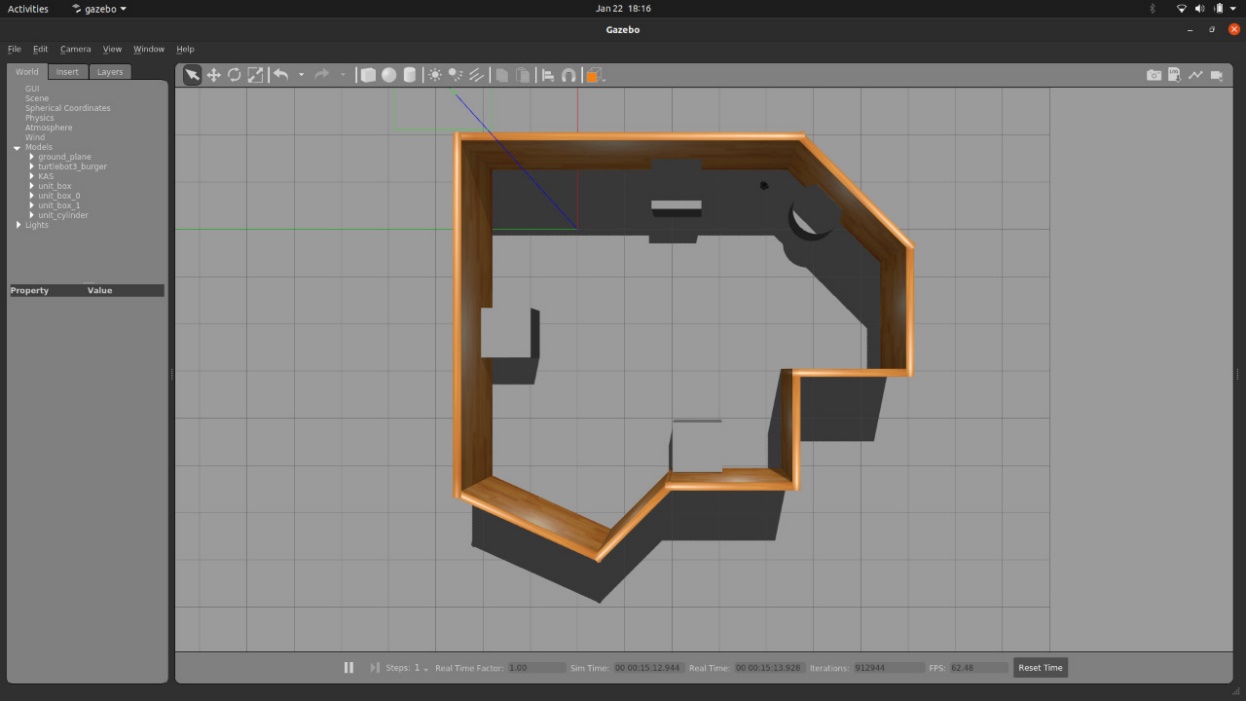


Figure Gazebo environment



Figure real testing environment- the goal is marked

# The code structures

Before starting with the codes, the picture below shows the UML diagram for the main class (Parking). Please refer to appendix 1 for full scaled picture.

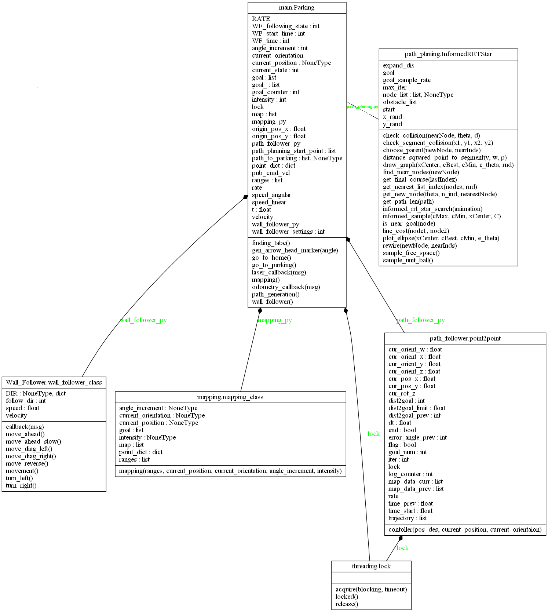


Figure UML diagram for code classes

Through this report, the main functions will be highlighted while the complete code will be submitted via GitHub.

Starting first with the main states of robot journey, the variable self.current\_state indicates to the current state.

* Mapping and wall\_follower: self.current\_state equals to 0
* Path planning: self.current\_state equals to 1
* Go to goal: self.current\_state equals to 2

Based on the self.current\_state value the actual state will change.

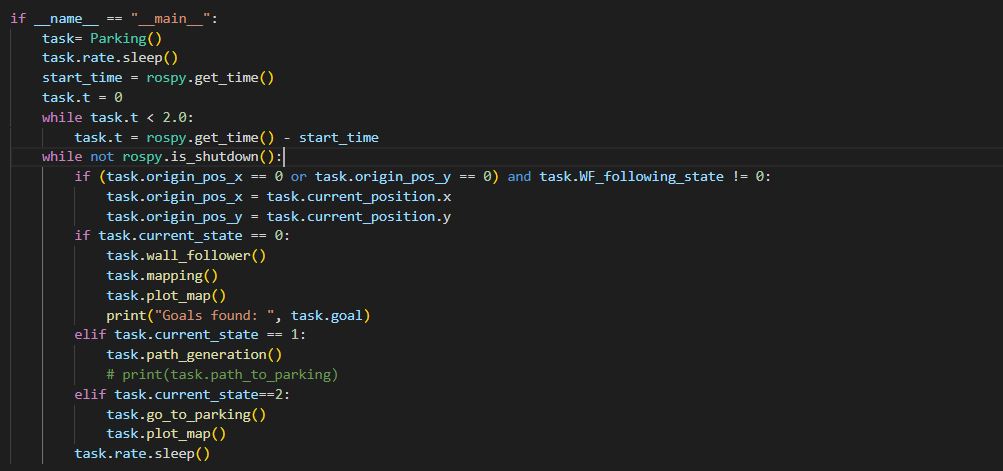


Figure changing between function statement

## Wall follower

The wall follower class consists mainly these functions:

Init function: for initialize the main variables.

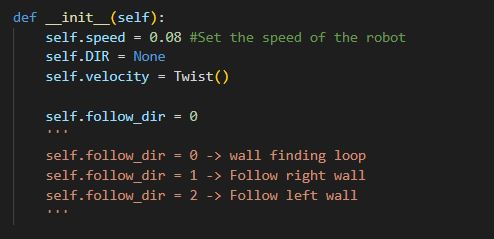


Figure initialization

self.follow\_dir variable represent the status of robot which will affect the robot movement.

Callback function: for adjusting the LiDAR ranges and sorting the ranges for Font\_list, left\_list and right\_list then find the minimum range then assign to directory.

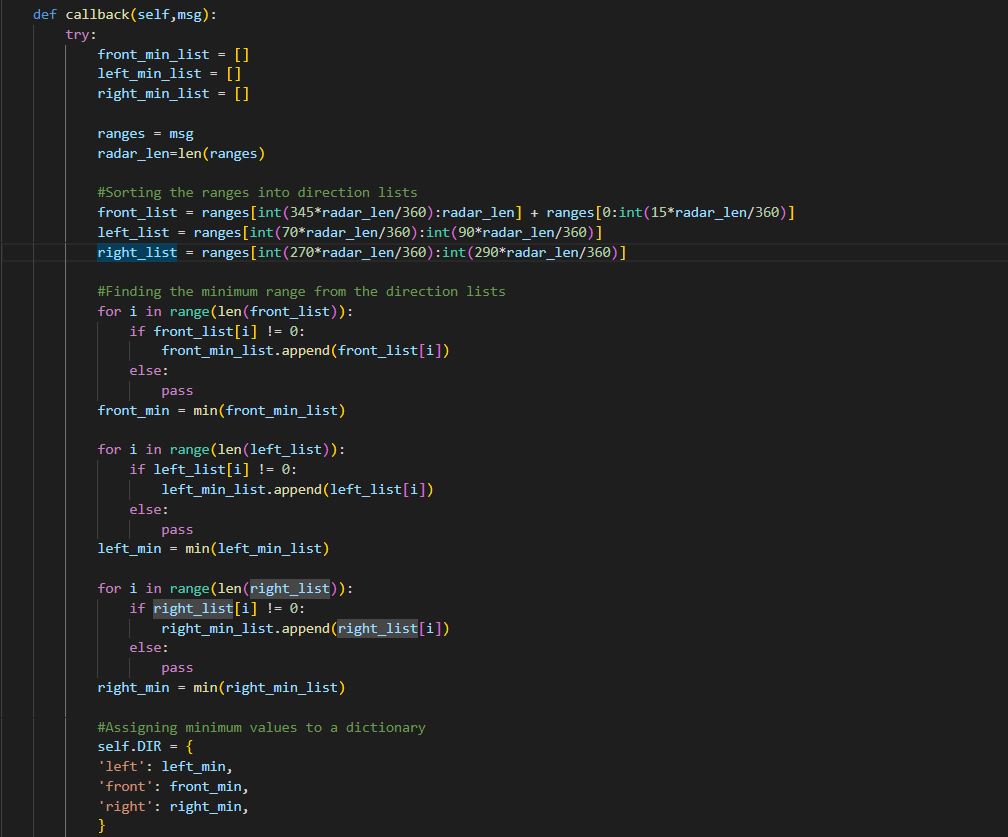


Figure callback function

The second function is movement: in this function the minimum and maximum thresholds are defined which will be used to move the robot ahead, turn right, turn left, diagonal left, and diagonal right based on certain conditions.

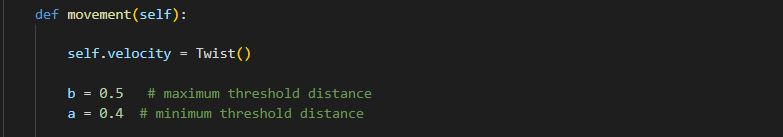


Figure setting the threshold

The conditions are shown as below:

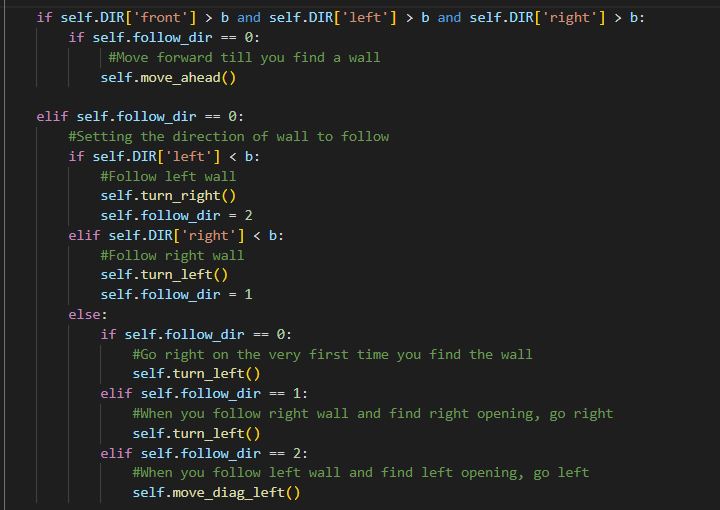


Figure wall follower conditions



Figure wall follower conditions



Figure wall follower conditions

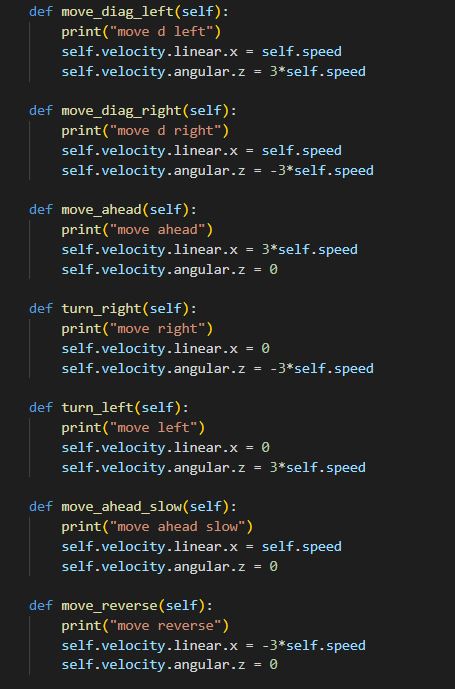


Figure the robot movement functions

Based on the conditions which is basically a comparison between thresholds and Lidar reading, the robot will move and change the status. The movement done by publishing the velocities (linear and angular).

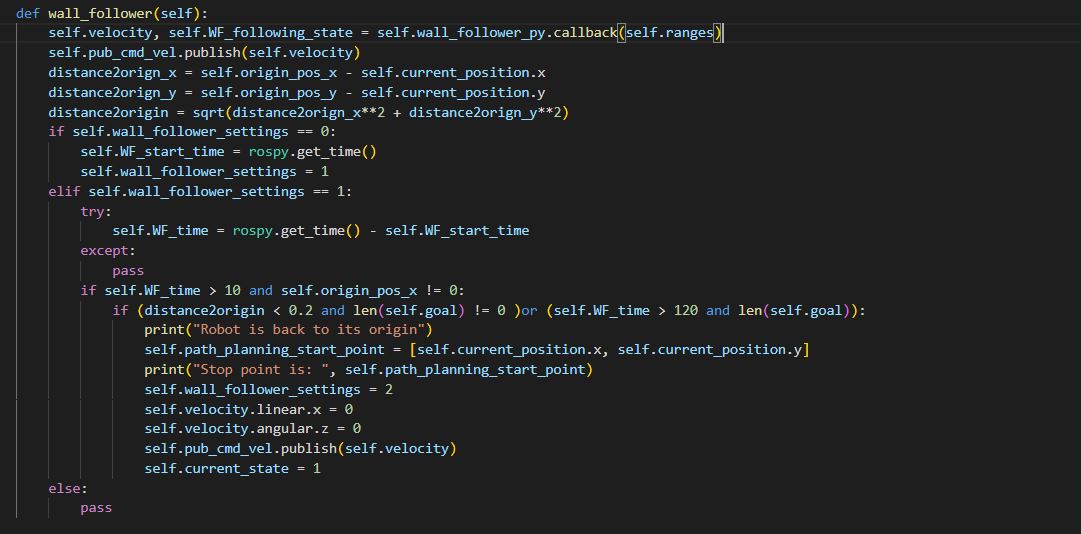


Figure Wall follower function in the main code

The wall follower function will stop if the robot is close to the starting point and the robot find a goal or if the robot search for 120 second and find a goal.

The self.current\_state will change to 1 which will lead to move to path planning stage.

## Mapping and goal finding

For mapping the matplotlib.pyplot is used to plot x,y coordinated for obstacles, goal and wall and to implement this:

* Looping through the LiDar reading to convert them to x,y coordinates using the Lidar values and angle\_increment
* Both x,y points were converted from body frame to global frame using the rotation matrix.
* To plot a clear map without overlapping (also for the trajectory planning), before appending the points from the previous step, the points were rounded to 1 digit then if the points are not in the list (new point), they will append.
* For detected the goal, the intensity values are used, the values should be less that 80 (the reflective tape issue was discussed before) also the goad should be detected 3 times (12 low intensity points in sequence) to be added to the goals list (to avoid errors in intensity values).
* The goal list with the map will be returned out from the mapping function to be used in trajectory planning.



Figure Mapping function-intensity values pattern and points coordinates

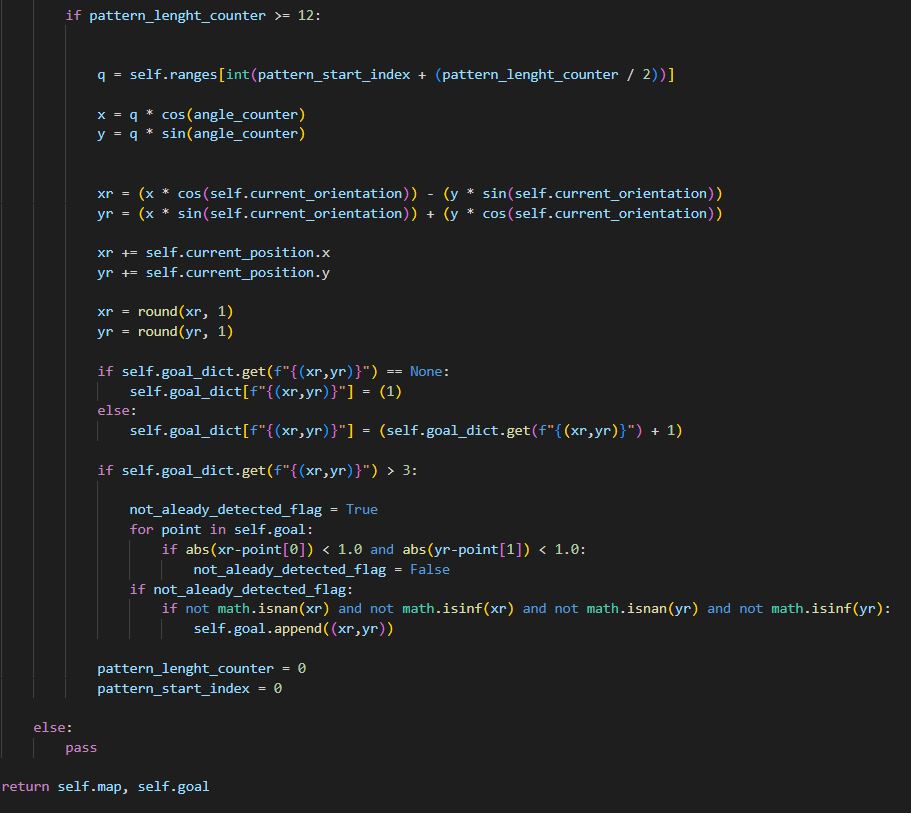


Figure goal list

Finally, it is worth mentioning that the map consists x, y and intensity values for each point.

## Trajectory planning

Informed I-RRT\* is used for trajectory planning, the algorithm will go through all the goal points inside the goal list and using the mapping to create the trajectory. For each goal, two paths are created to choose the optimal path (the shortest one).

Text

Description automatically generated

Figure path\_generator function

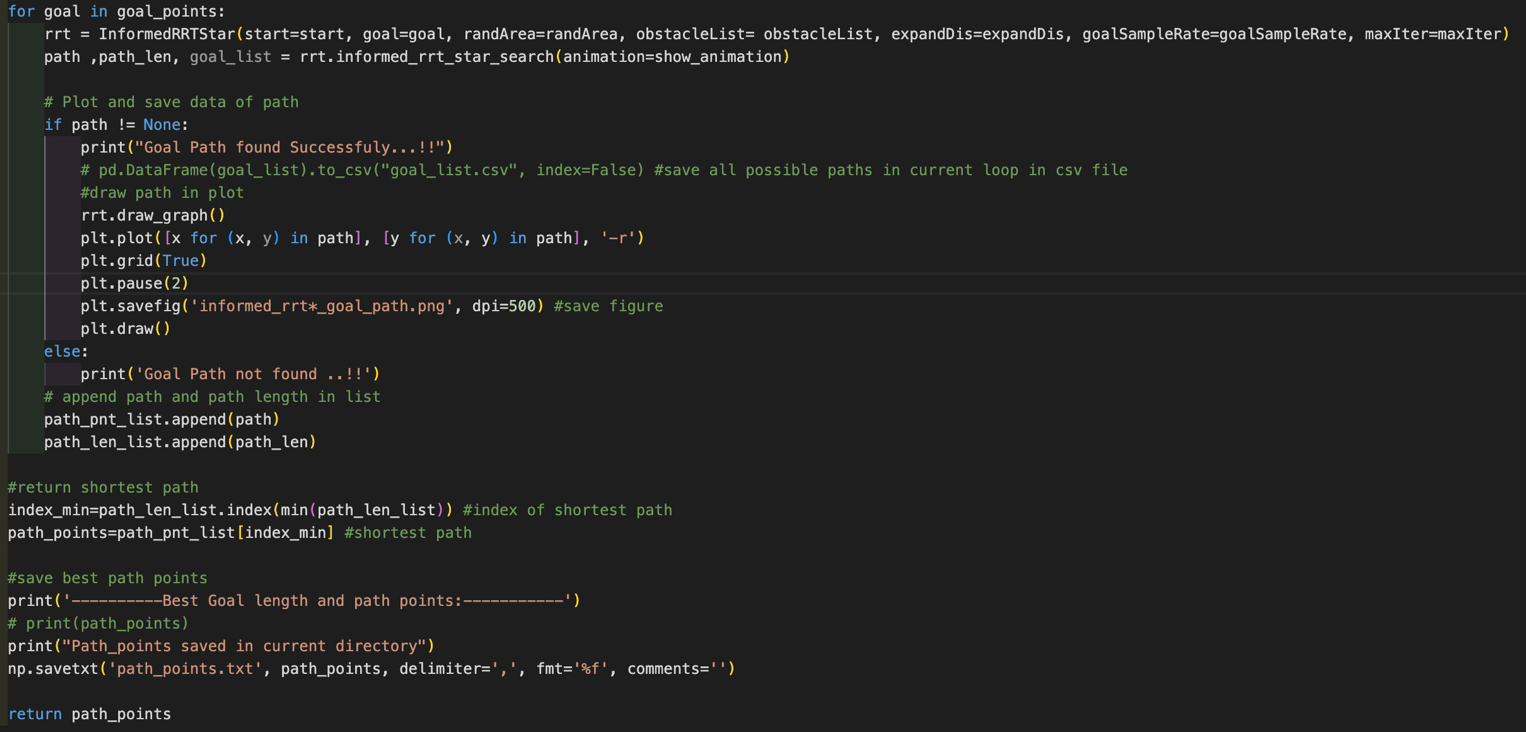
Here the minimum length of steps can be adjected as well as the goal sample rate. These values were optimized for our case study. While the number of iterations is coded to be dynamic based on the algorithm needs.

Figure Looping through the goals, finding the shortest path for each goal and find the shortest goal waypoints

## Path following and Park

The waypoints created by the trajectory planning used as input for the path\_follower. PID controller is used with point-to-point method.

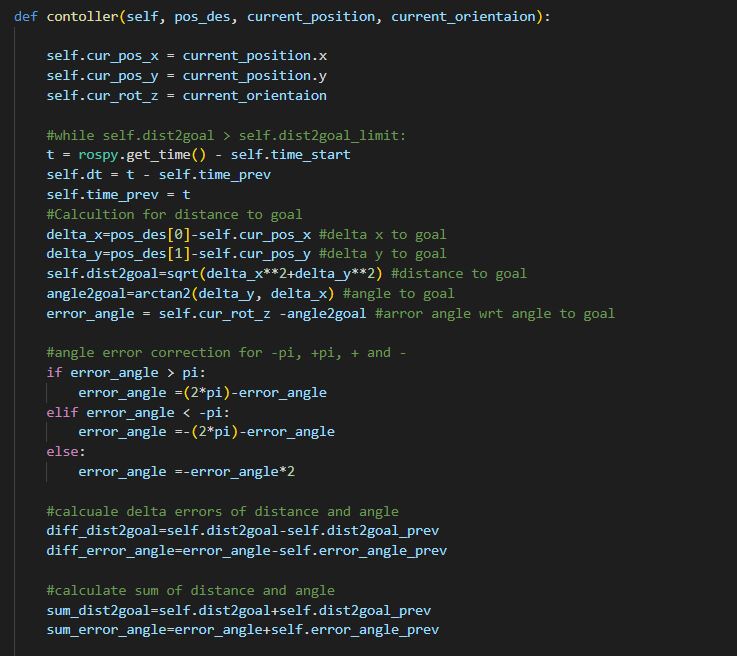


Figure measuring the distance to goal and error

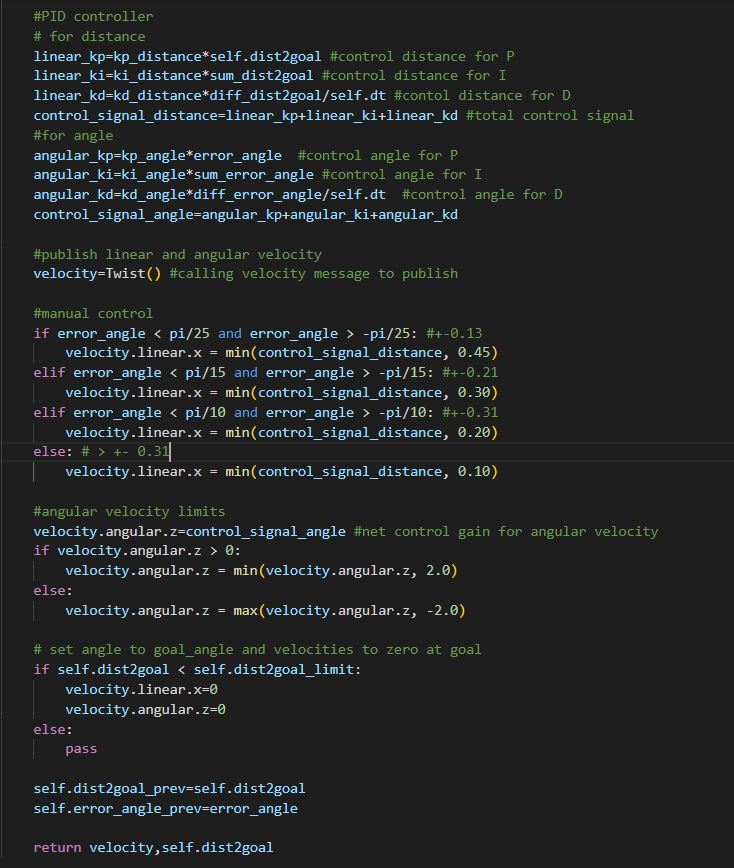


Figure Controller part

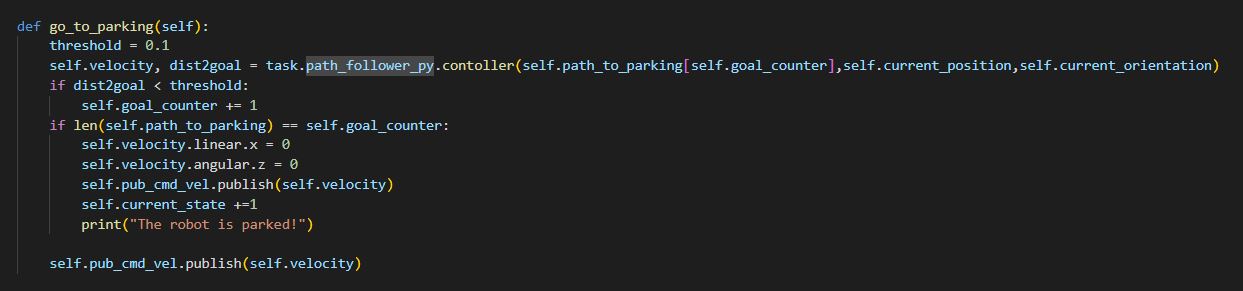


Figure go\_to\_parking function

## Code Integration

All the previous function are integrated in one main file. In this file the ROS call back functions were defined, the needed inputs are assigned for each stage and function, and the plot function is defined.

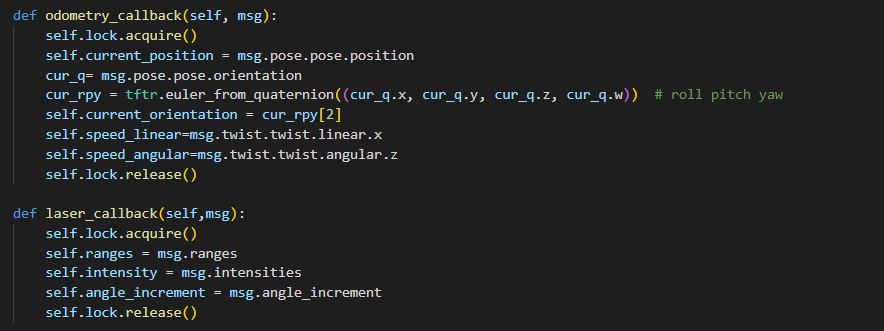


Figure call back functions



Figure plotting function

# Results

The video (<https://www.youtube.com/watch?v=Z-PZd_QhgtA> ) shows the final results of Automatic parking project. However, the pictures below show captions of the results.

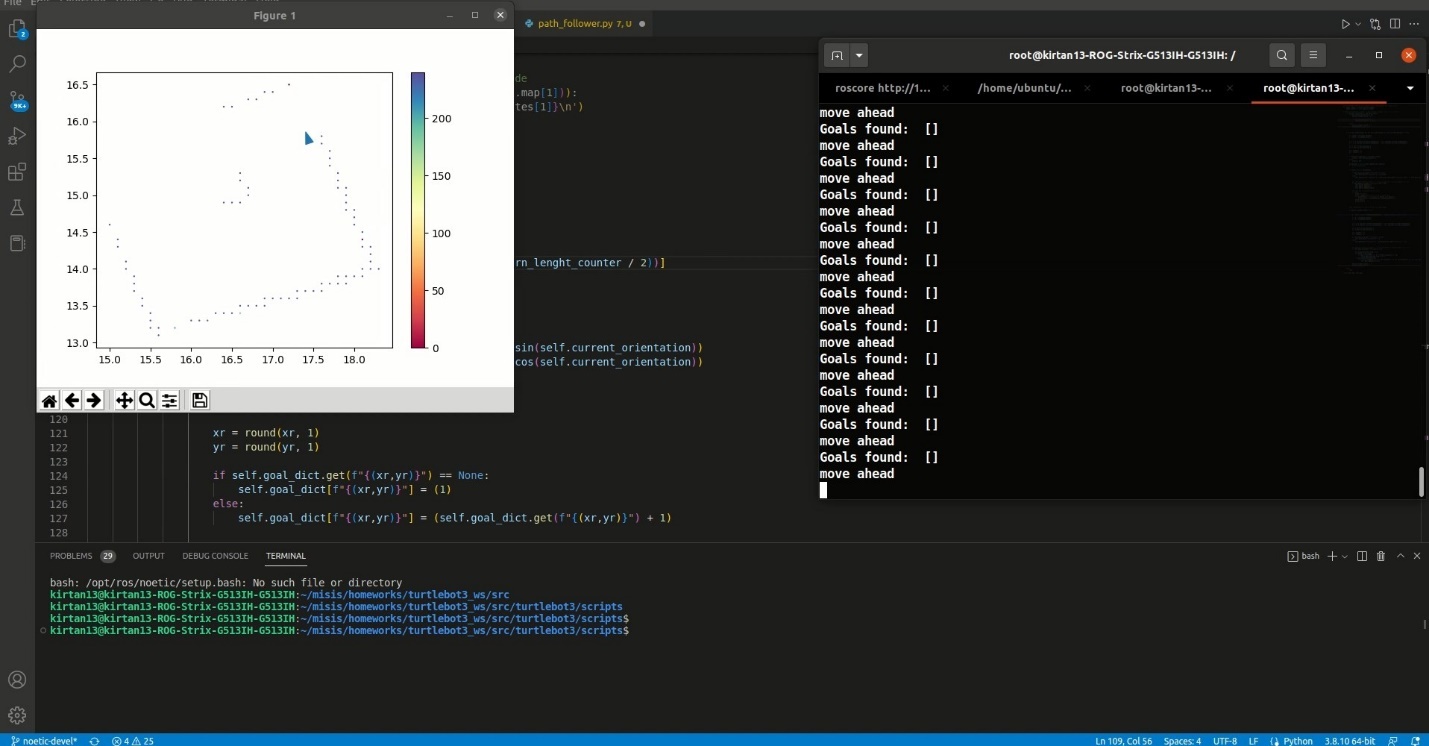


Figure wall follower and mapping

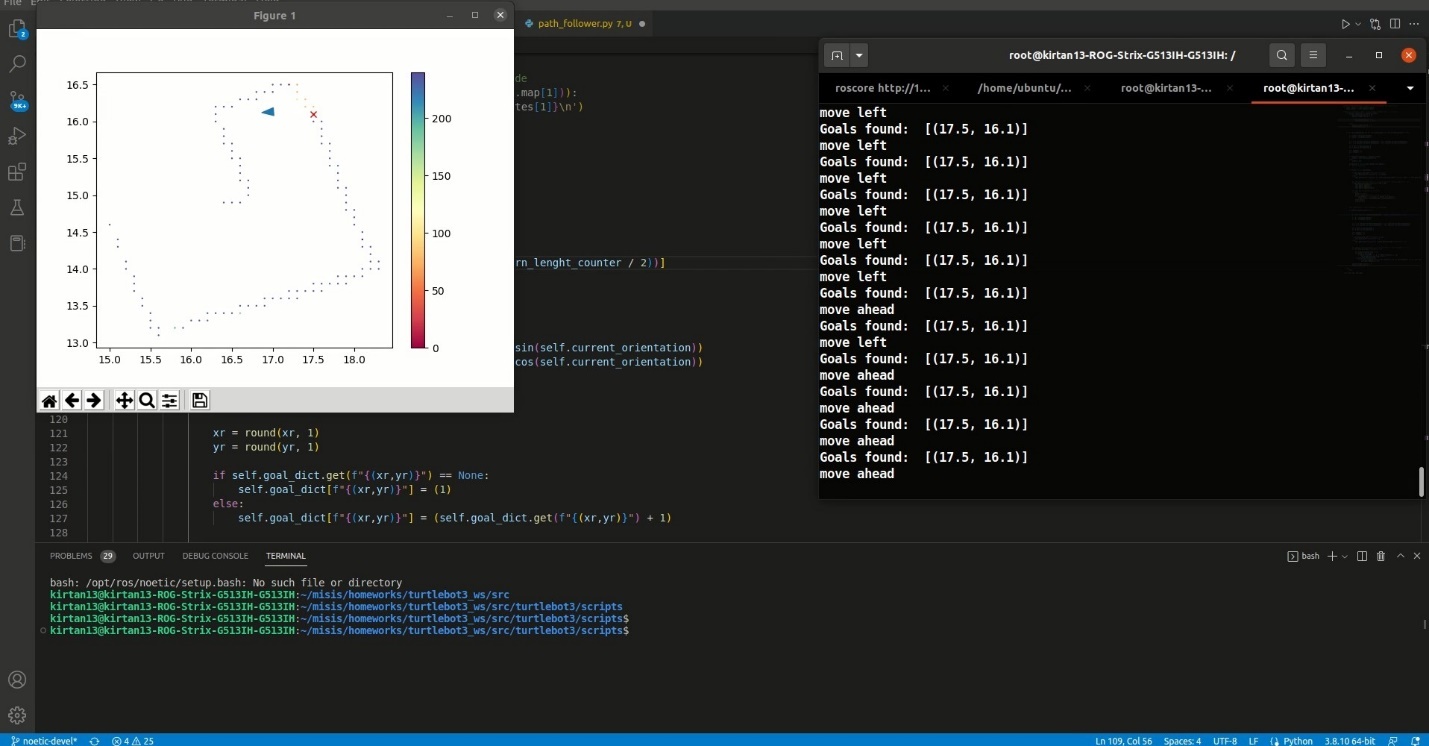


Figure find a goal (x marked) the goal coordinates are printed in the terminal

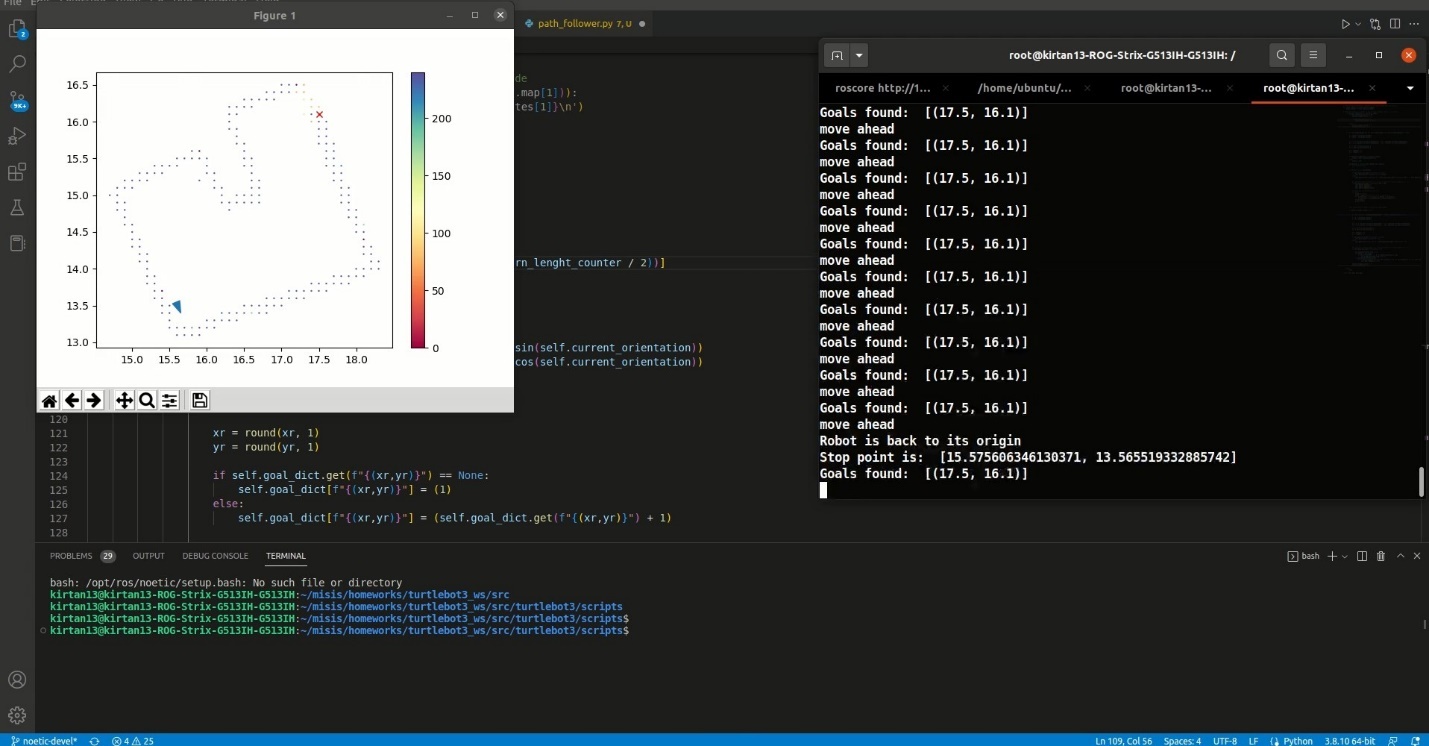


Figure Back to origin (home)- see the terminal

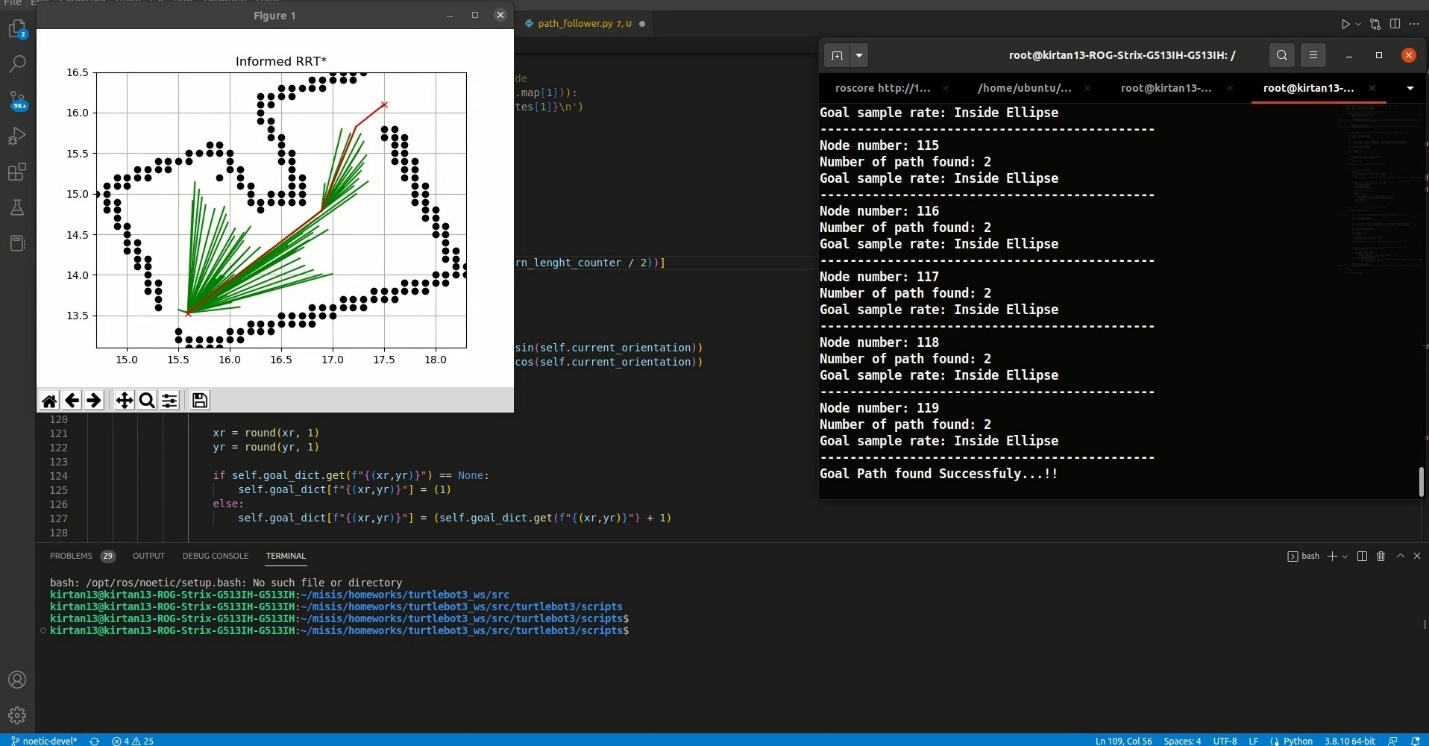


Figure Trajectory planning results the shortest path found- see the terminal

The shortest path was found in 41 seconds

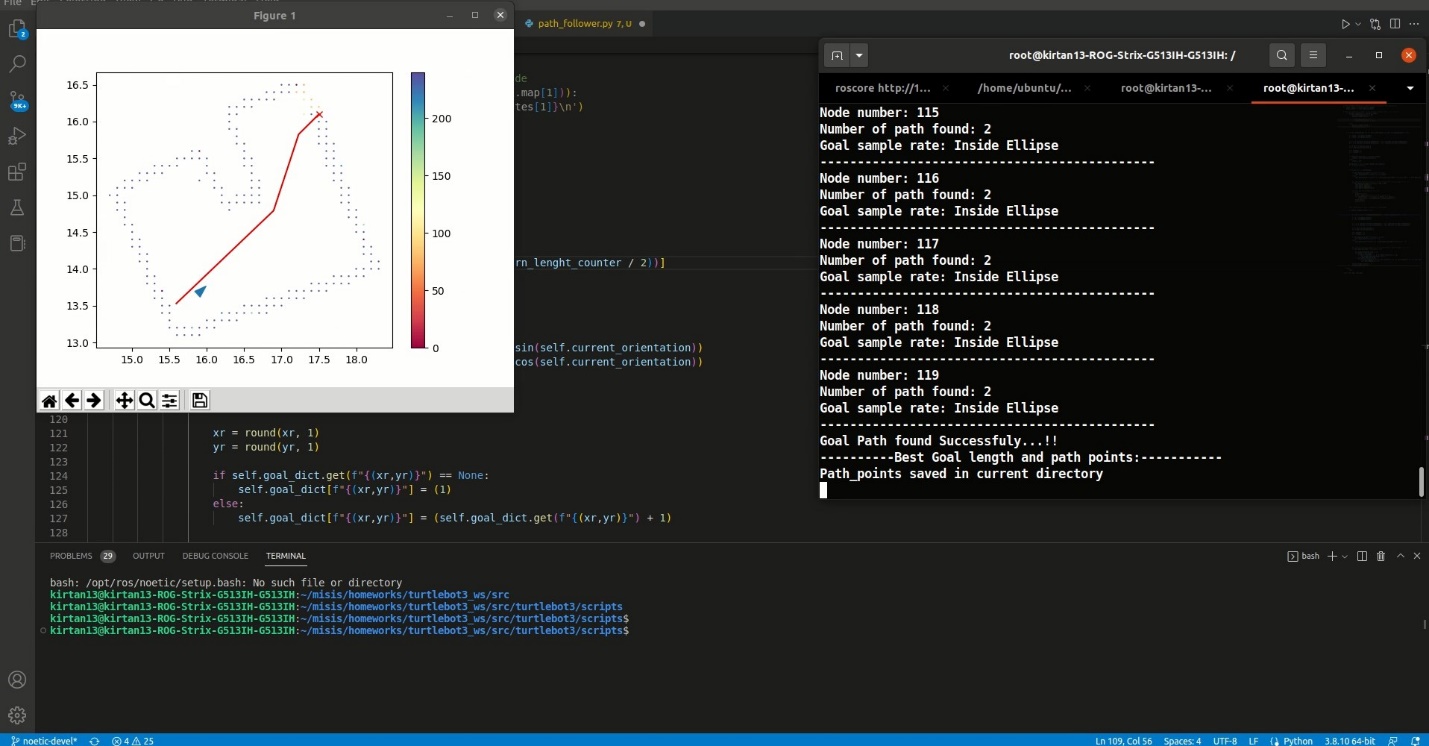


Figure Following the trajectory

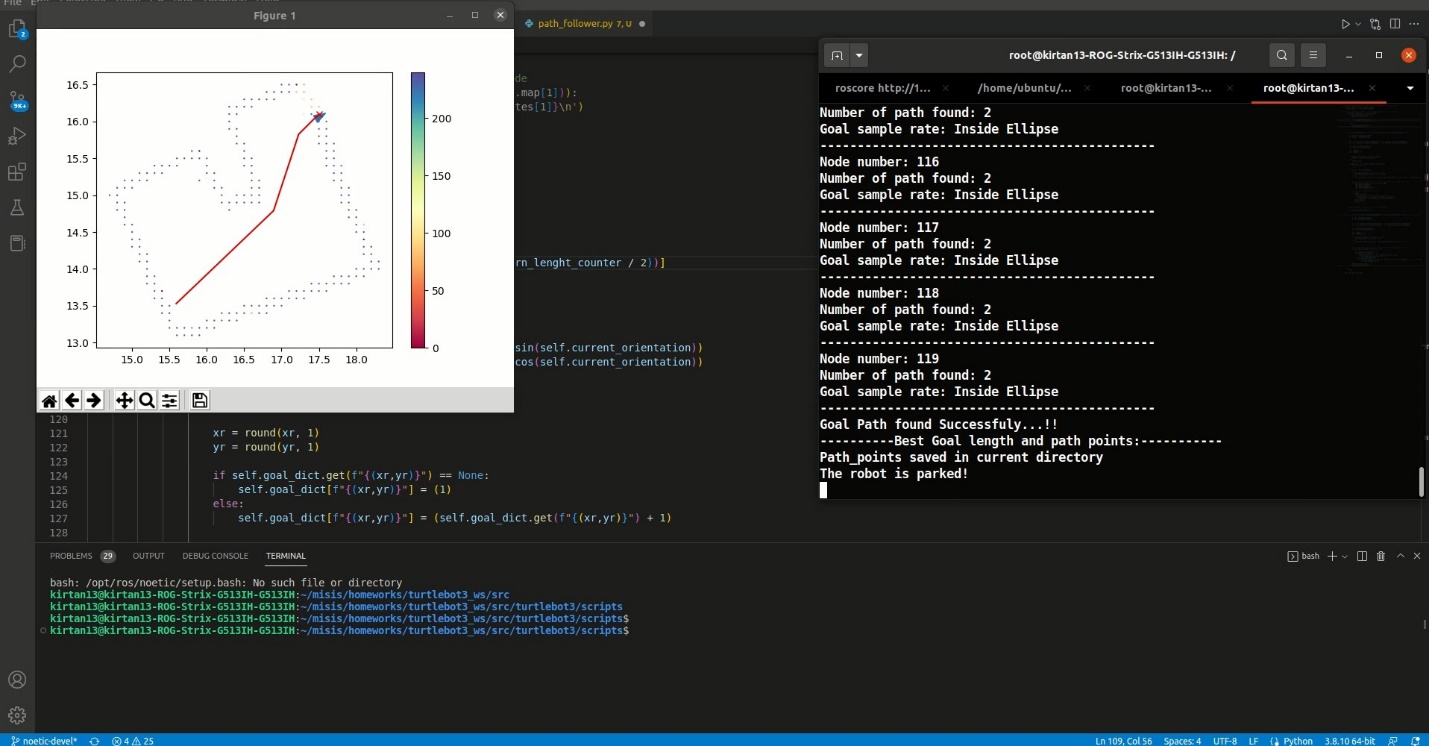


Figure The robot parked!!