FH Technikum Wien
Mobile Robotics + Service & Object-Oriented Algorithms in Robotics
Semester Project

Robotics Maze Escape

Alejo Flores Lucey (ID: if24x390)
Mauro Leandro Baez (ID: se24m502)

01

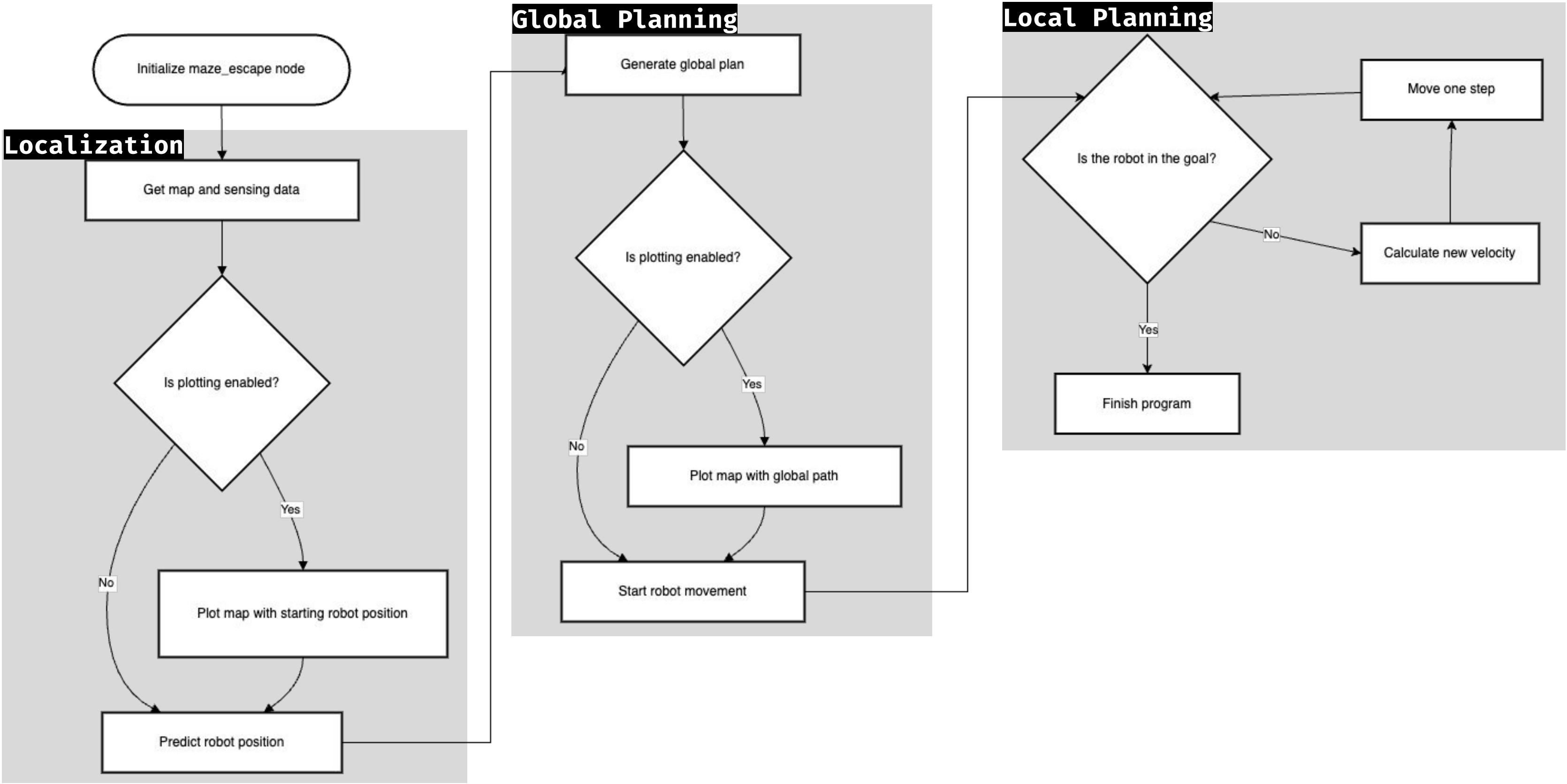
Introduction

Using RosPy, we developed the whole cycle of a robot positioning and movement around a world.

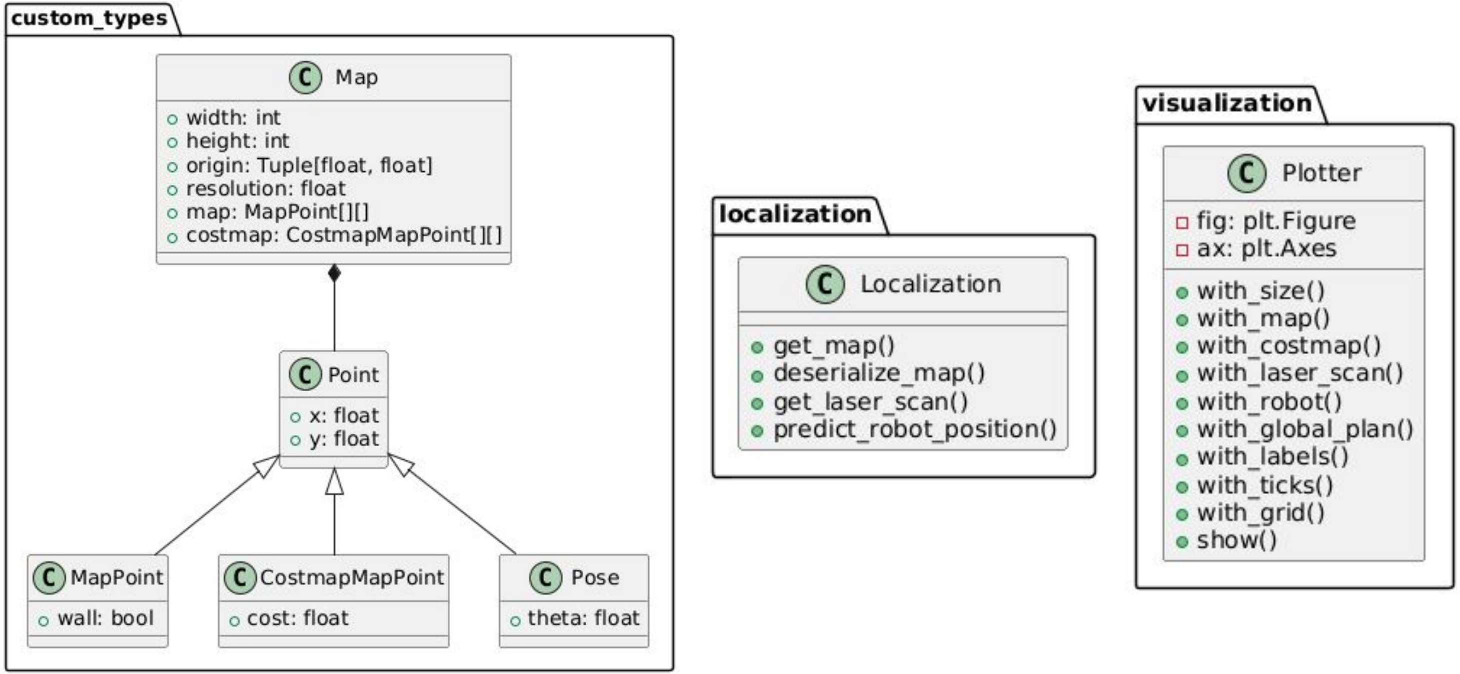
Implementation Details

- **Modular Python Codebase**: The project is organized into modules, each handling a specific aspect of the robot's behavior.
- **Custom publishers**: We publish messages to topics such as /maze_escape/global_plan and /maze_escape/goal_pose to better visualization in RViz.
- **Visualization**: The Plotter class, that follows the Builder design pattern, provides real-time visualization of the robot, map, and planned paths using Matplotlib.
- **Documentation**: The source documentation is auto-generated using Docstrings with the Google syntax, using the Python package pdoc.

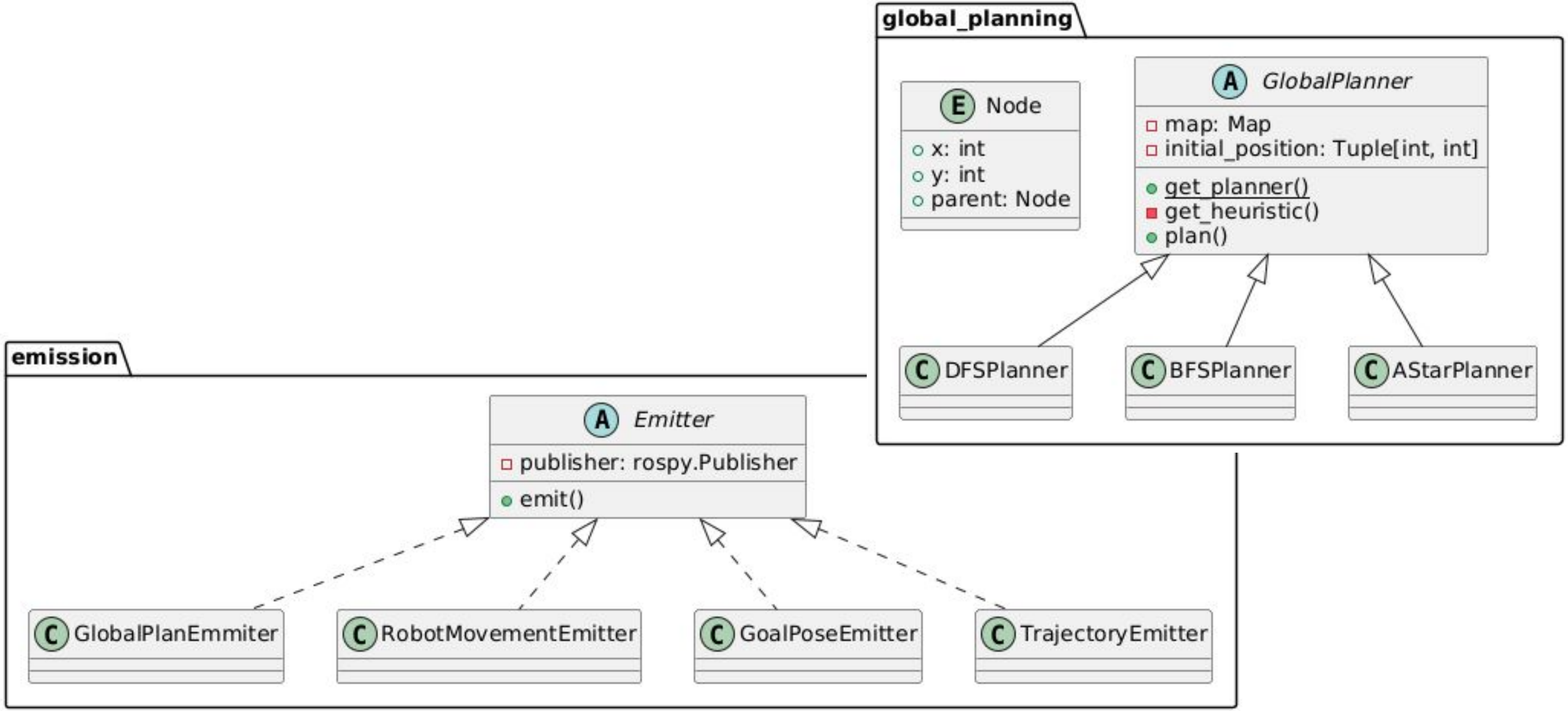
Basic Flowchart



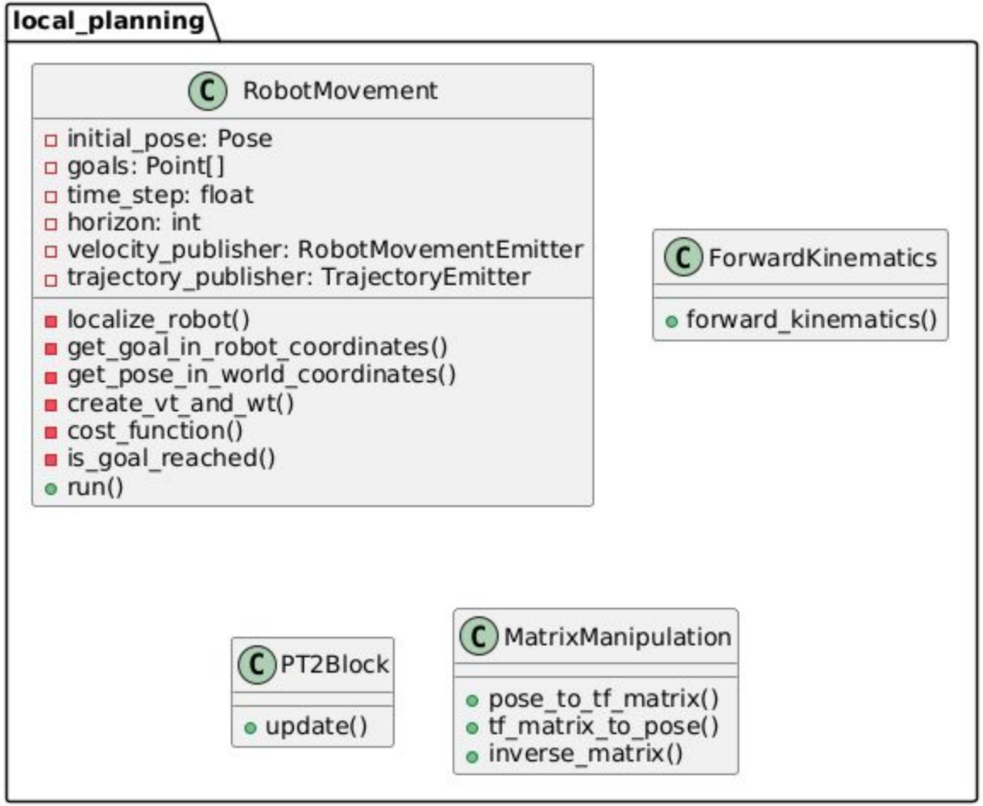
UML Diagram (1)



UML Diagram (2)



UML Diagram (3)

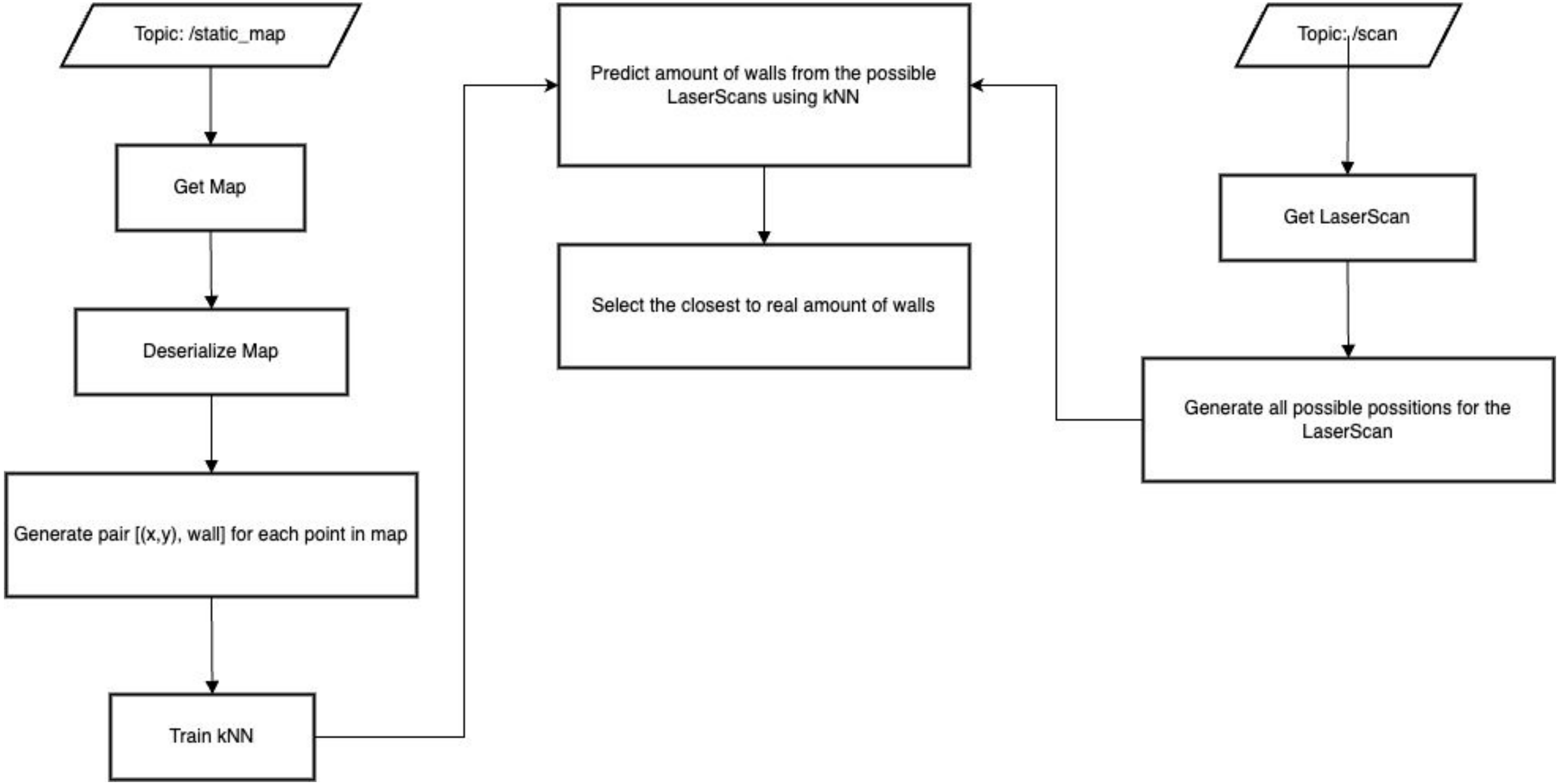


02

Localization

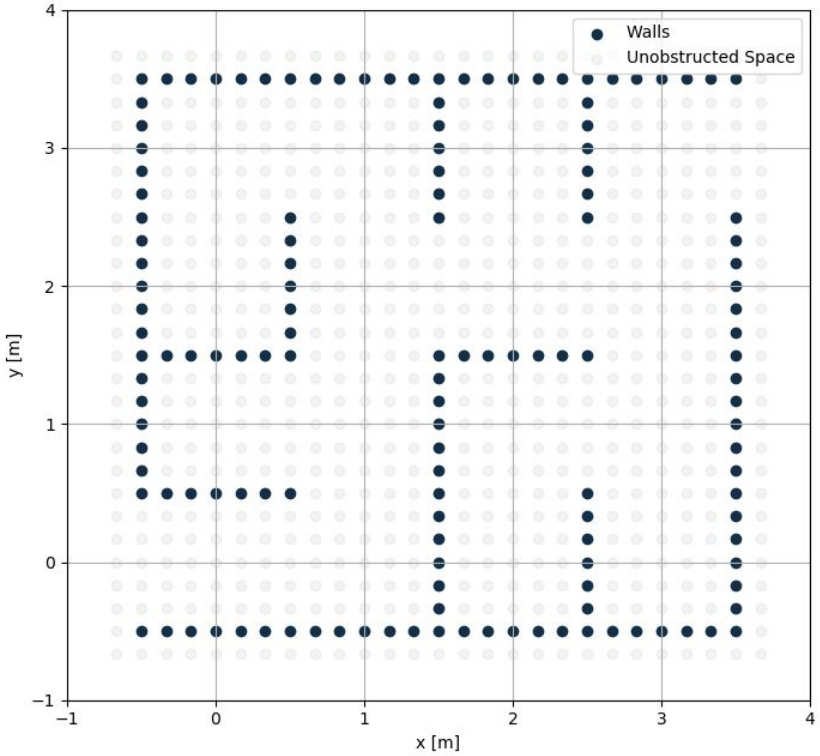
First, we get the map information. Then, using the laser scanner of the Turtlebot3, we use k-Nearest-Neighbours to predict the robot position in the world.

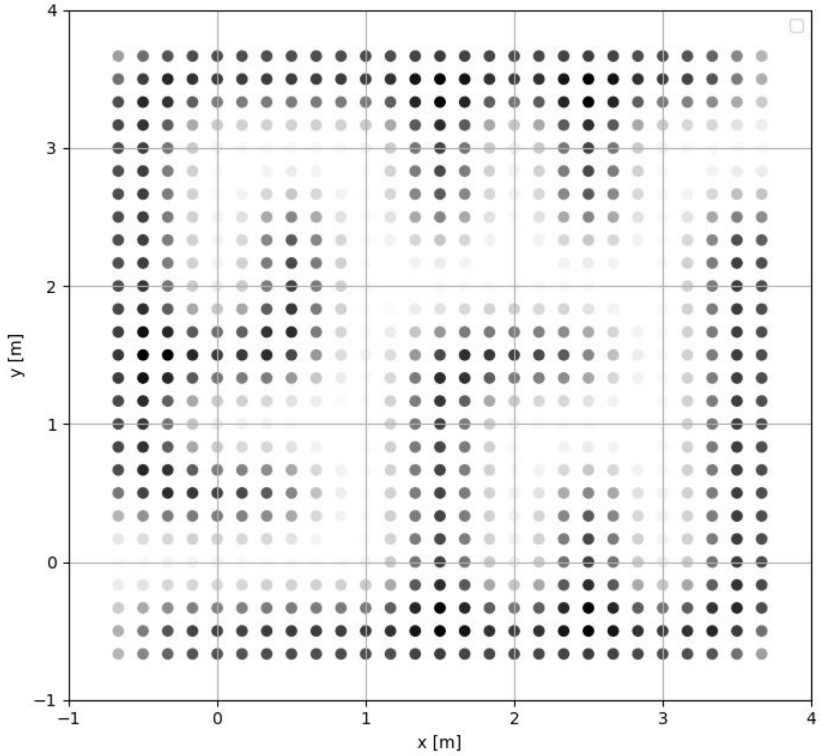
Flowchart


Topic: /static_map
Get Map
Deserialize Map
Generate pair [(x,y), wall] for each point in map
Train kNN
Predict amount of walls from the possible LaserScans using kNN
Select the closest to real amount of walls
Topic: /scan
Get LaserScan
Generate all possible possitions for the LaserScan

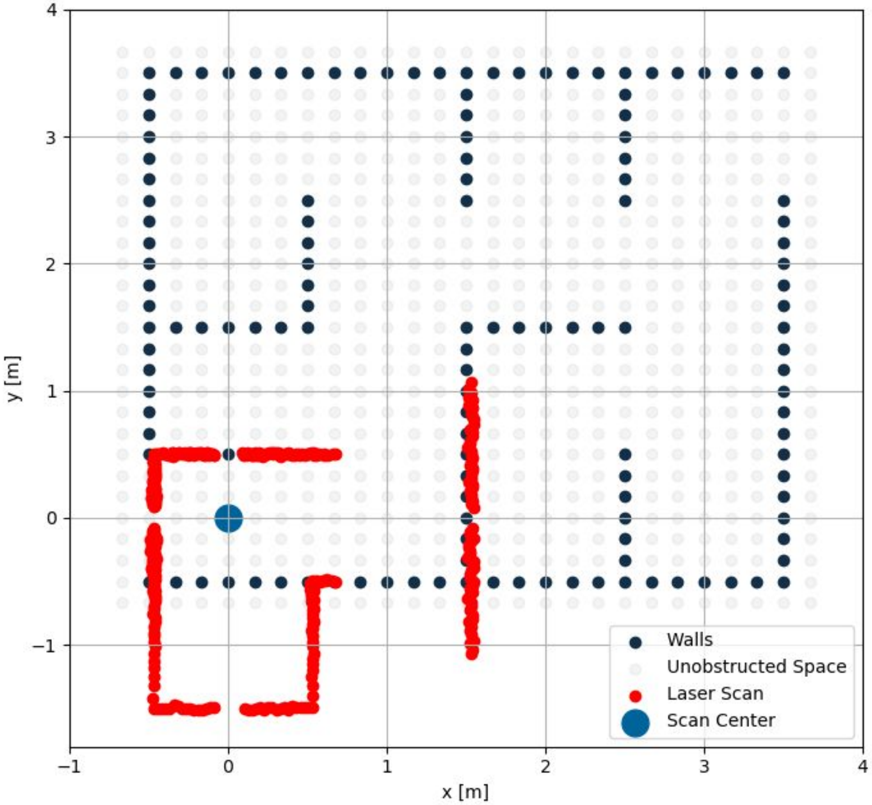
Generated Map

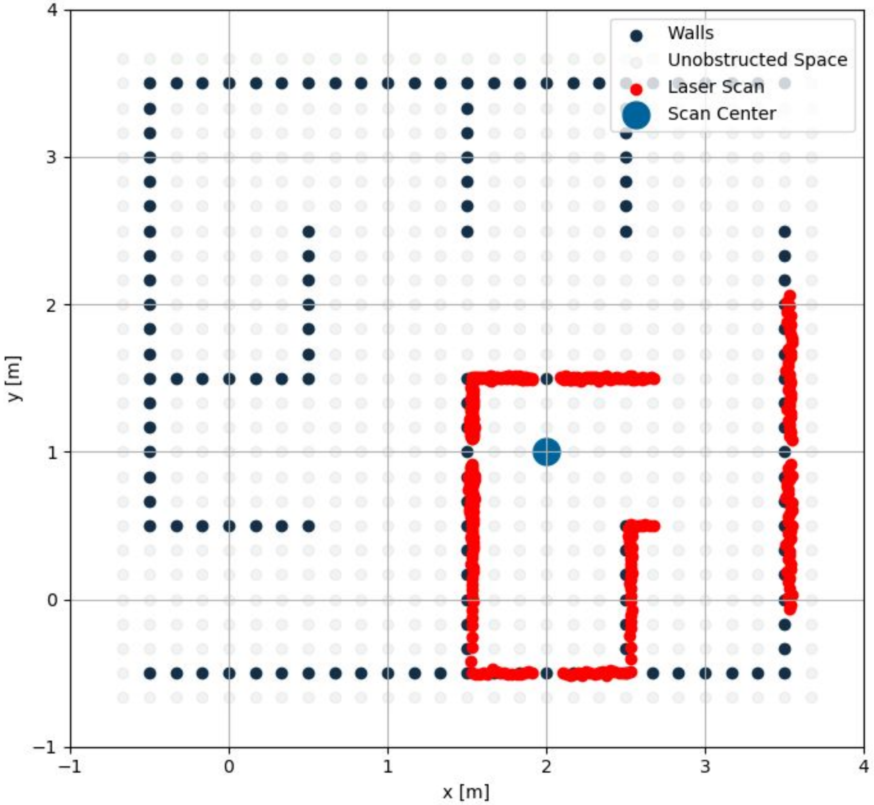
We have both a **basic map** and a **costmap**.





Predicting Robot Position



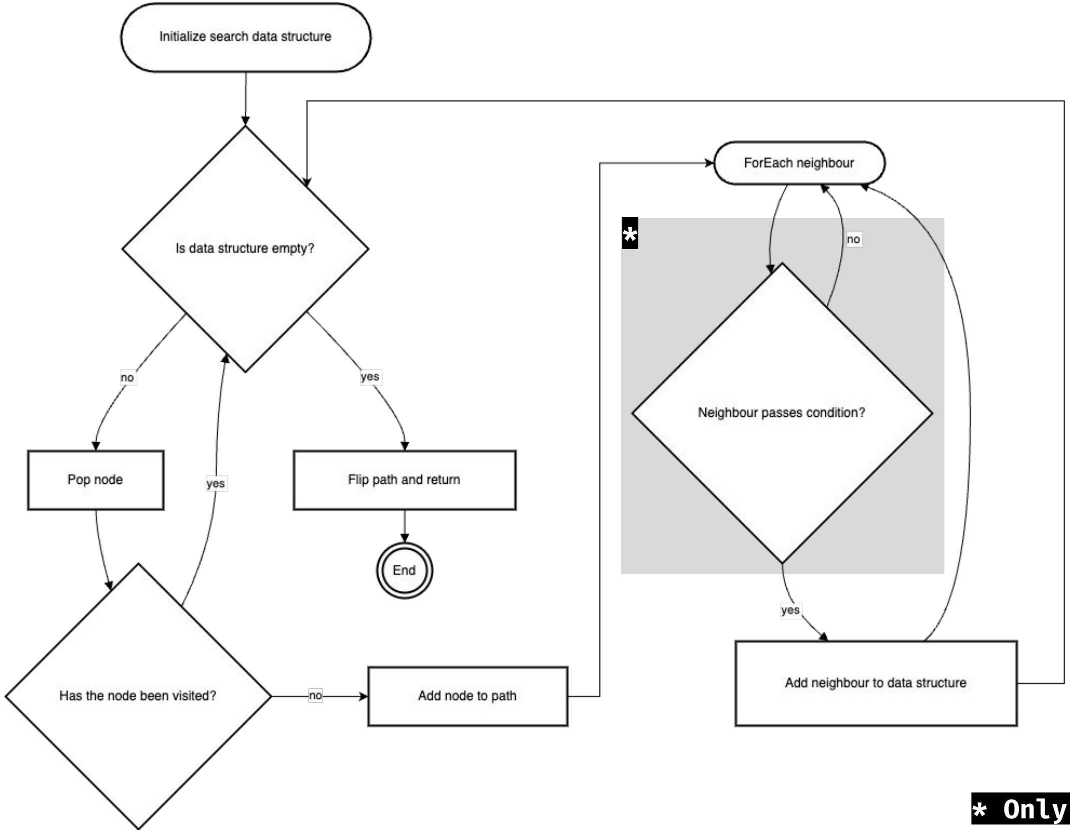


03

Global Planning

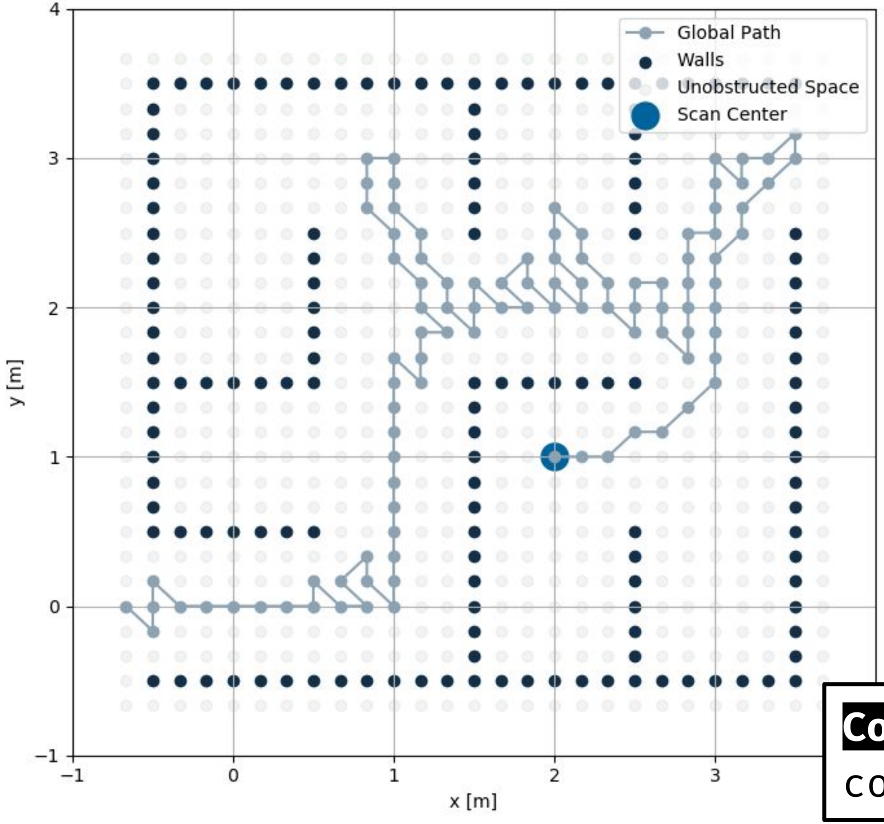
We implemented both uninformed and informed search algorithms. BFS, DFS, A* with Euclidean Distance, A* with Manhattan Distance & Dijkstra.

Flowchart



* Only in uninformed algorithms

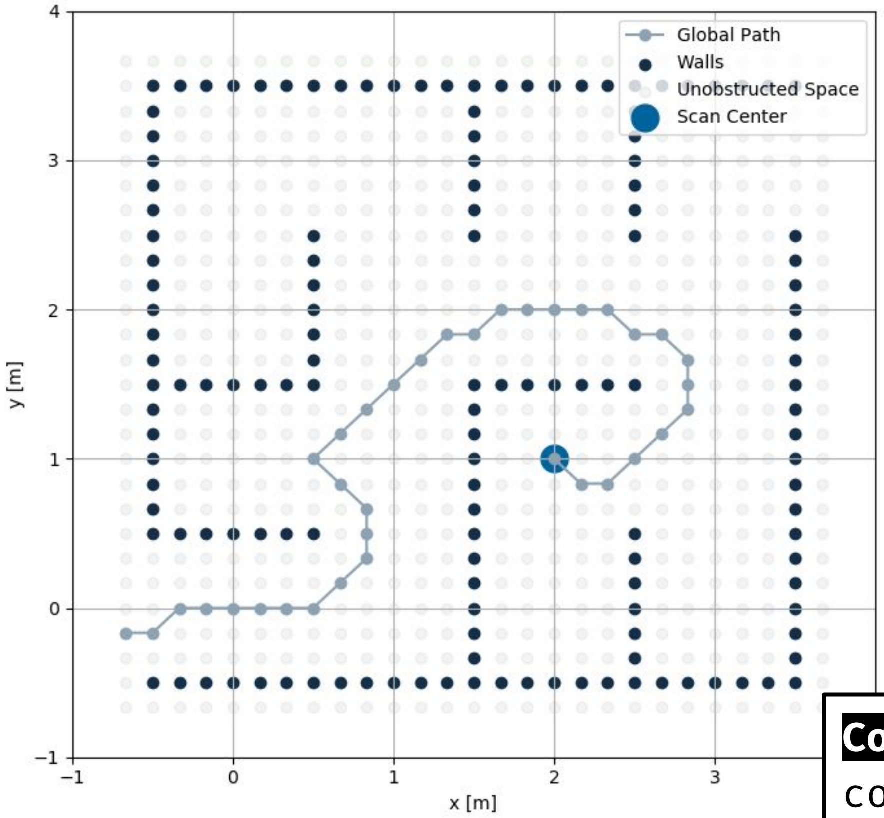
Depth-First Search (DFS)



Condition:

`costmap[x][y] < 0.15`

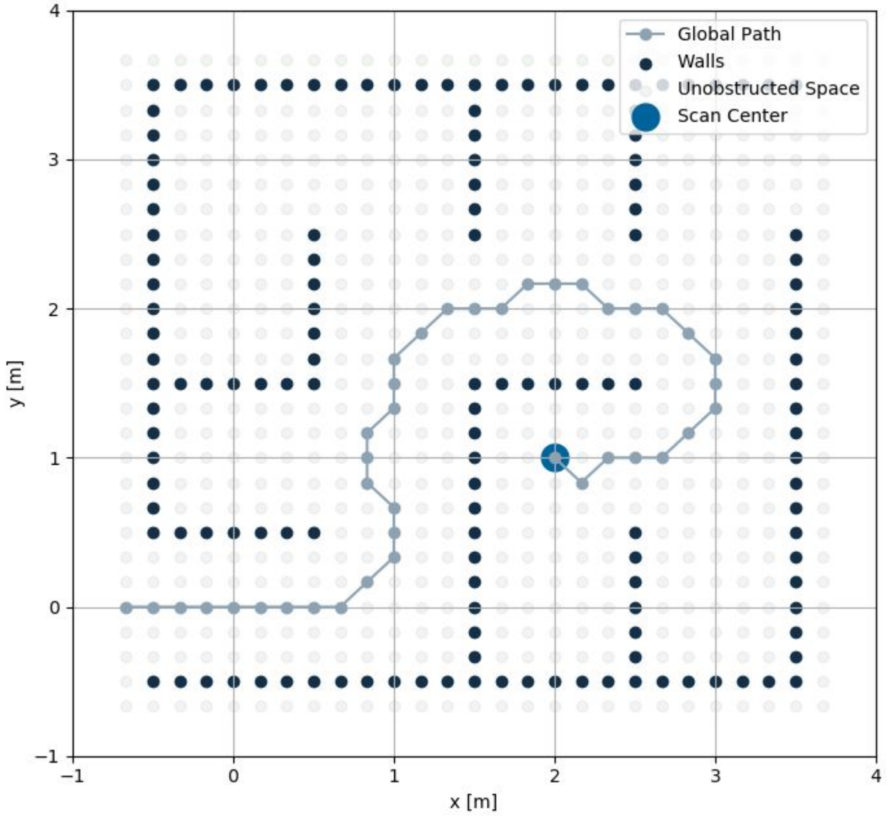
Breadth-First Search (BFS)

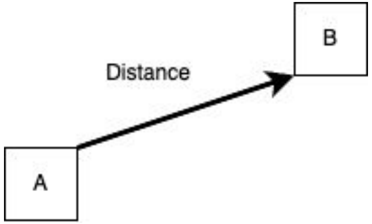


Condition:

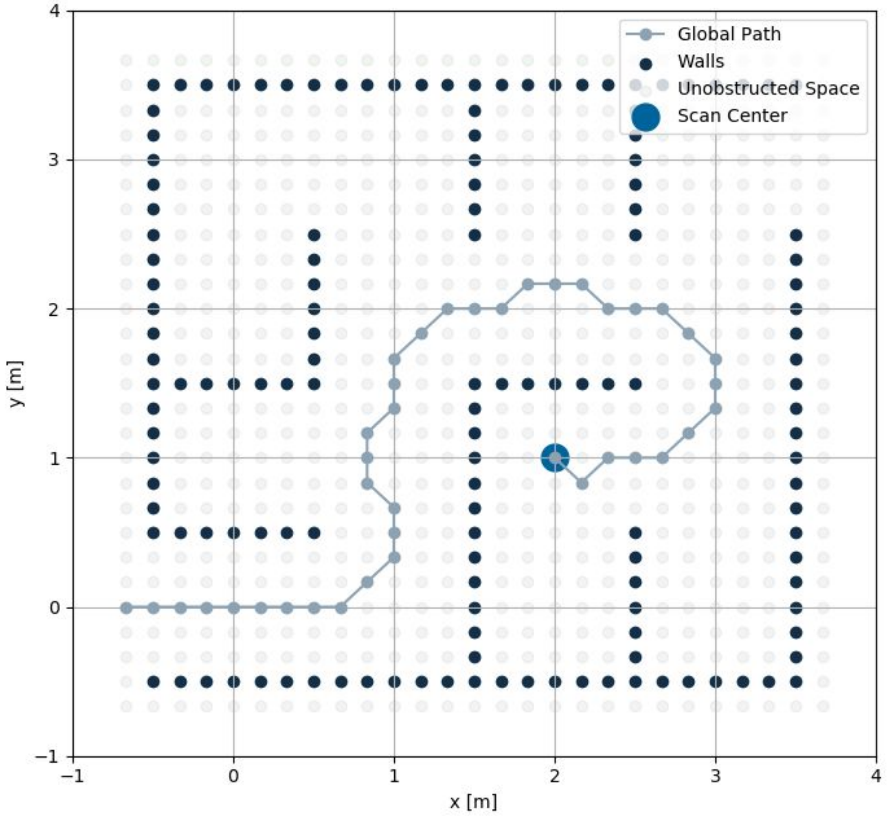
`costmap[x][y] < 0.15`

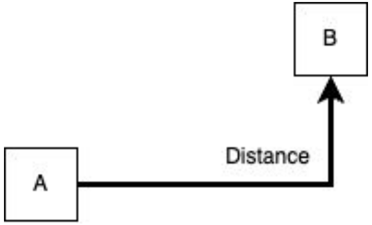
A* with Euclidean Distance



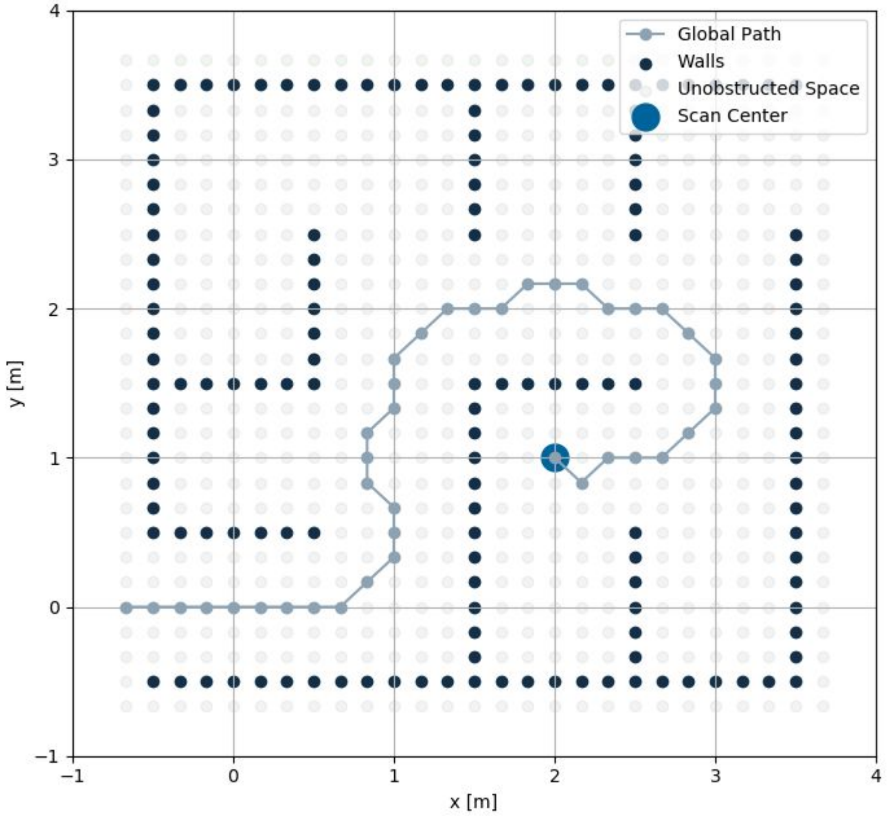


A* with Manhattan Distance





Dijkstra’s Algorithm

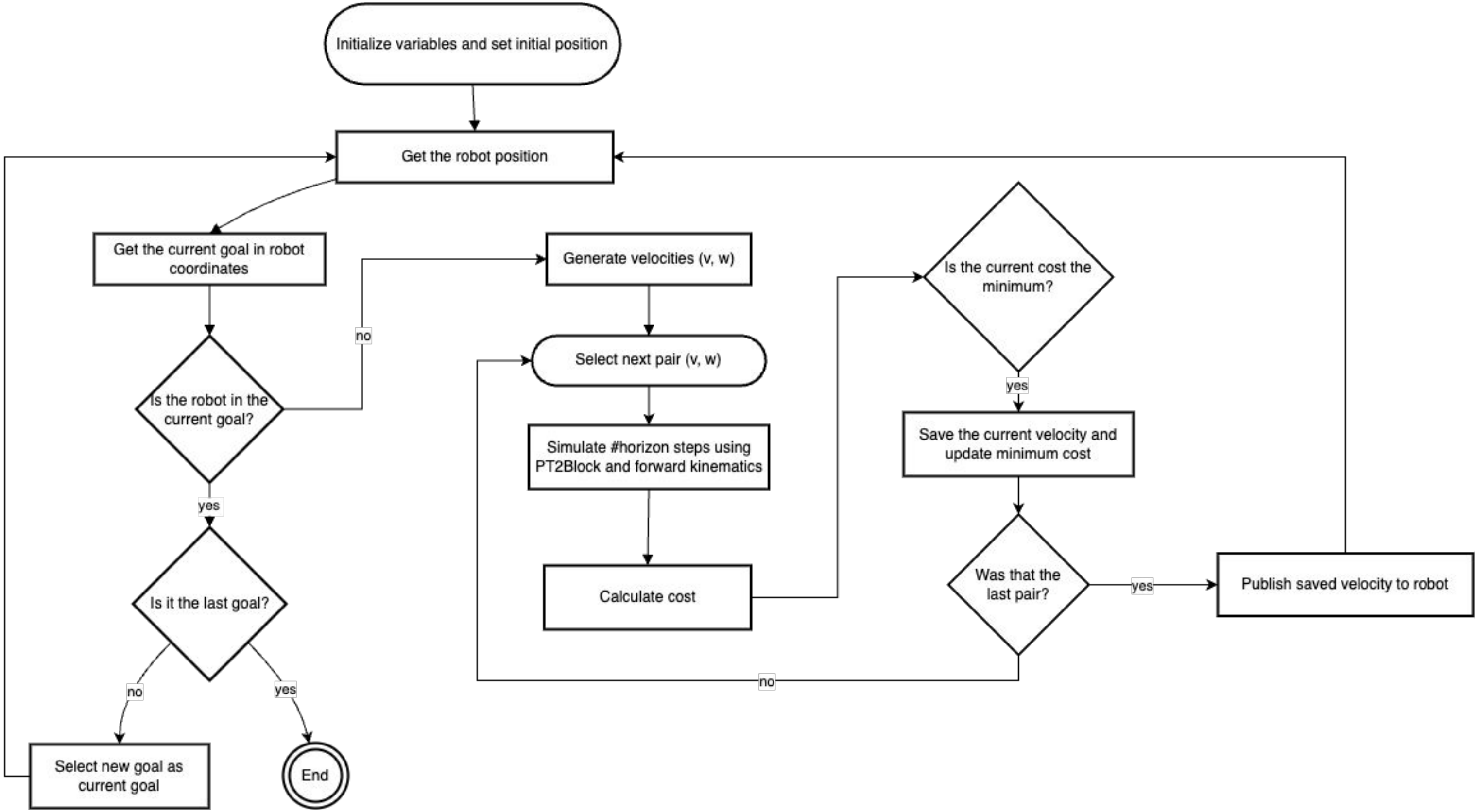


04

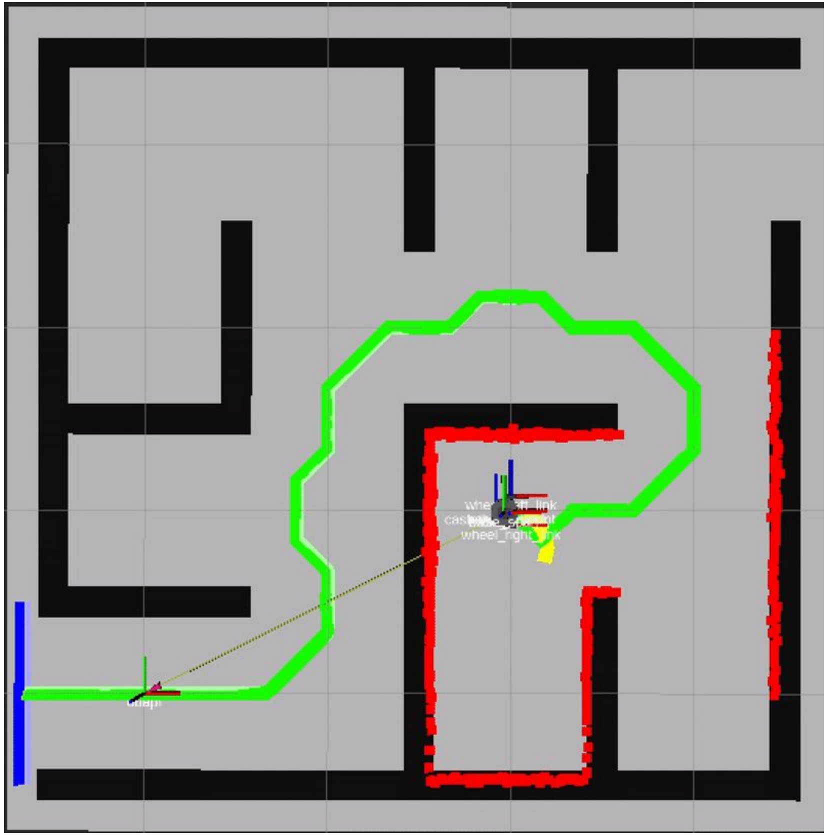
Local Planning

Using forward simulation, with variable look-ahead steps. We then send the velocity commands to the robot using topics.

Flowchart


Initialize variables and set initial position
Get the robot position
Get the current goal in robot coordinates
Is the robot in the current goal?
no
yes
Is it the last goal?
no
yes
Select new goal as current goal
End
Generate velocities (v, w)
Select next pair (v, w)
Simulate #horizon steps using PT2Block and forward kinematics
Calculate cost
Is the current cost the minimum?
yes
Save the current velocity and update minimum cost
Was that the last pair?
yes
no
Publish saved velocity to robot

Robot Movement



05
Demonstration

Let’s see how all of this works in RViz

How to run?

robotics-maze-escape

```
roslaunch robotics-maze-escape launch_simulation.launch \
    enable_plotting:=<true | false> \
    goal:=<1 | 2> \
    global_planner_algorithm:=<bfs | dfs | astar | dijkstra> \
    global_planner_heuristic:=<manhattan | euclidean>
```

Thank you!