Robot's Localization, mapping and path planning

Francisco Gil Mata (99221) **Gonçalo Aroeira Gonçalves (99226)**

1. Introduction

a. Motivation

The ability to self-localize, to map environments, and plan paths are essential capabilities for robots. These kinds of technologies have revolutionized numerous industries and applications, making robots more versatile and autonomous than ever before. From autonomous vehicles navigating complex urban environments to robots assisting in disaster response, search, rescue missions and even healthcare settings. These methods play a pivotal role. They enable robots to accurately determine their positions in the environment, create detailed maps of their surroundings, and plan efficient paths to reach their goals, thereby enhancing their capabilities in industries such as logistics, agriculture, manufacturing, and healthcare. These tools empower robots to undertake monotonous and occasionally hazardous tasks instead of humans, frequently yielding superior outcomes.

b. Problem Statement

In this paper we will analyze these kinds of algorithms using a TurtleBot3 Waffle Pi robot. [1]This robot is equipped with a raspberry Pi processor, 360º LiDAR, two DYNAMIXEL actuators for the wheels and an IMU sensor. This allows us to subscribe to topics, such as the laser scan data ('/scan'), the Odometry data ('/odom') and the robot's position ('/pose').

The first mini project focuses on two fundamental aspects of mobile robotics, self-localization and environmental mapping. In terms of self-localization, we have employed two methodologies - the Extended Kalman Filter (EKF) and Monte Carlo Localization (MCL). Concurrently, our project also focuses on constructing a comprehensive environmental map using Gmapping algorithm.

The second mini project focuses on robot path planning. This is the ability for a robot to plan a path and move to a specific goal. For this purpose, we used the Rapidly Exploring Random Trees (RRT) algorithm, which is a powerful and widely used path planning technique in robotics. The main focuses were on understanding and adjusting the different parameters of RRT, and adapting the algorithm to improve the plans it produces.

2. Mini project 1

a. EKF

The Extended Kalman Filter (EKF) is a recursive estimation algorithm that extends the traditional Kalman Filter to handle nonlinear systems. It is a widely used method for state estimation in various fields, including robotics, control systems, and sensor fusion. The EKF works by estimating the state of a dynamic system based on noisy measurements and a dynamic model.

For our robot the essential subscribed topics are:

- /odom- Odometry is a way to estimate a robot's position and velocity by tracking how its wheels or other motion components have moved over time.
- /imu- The /imu topic is used to publish data from an Inertial Measurement Unit (IMU), which is a sensor that measures the robot's motion and orientation.

From the dataset, using EKF we were able to graph the **filtered odom**, which is a process in which odometry data, which estimates a robot's position and movement based on wheel encoder measurements, is enhanced and refined using the EKF. On contrary, the **ground truth**, refers to the absolute and accurate reference data or information that serves as the definitive benchmark for comparison and validation in various fields (see **Figure 1**).

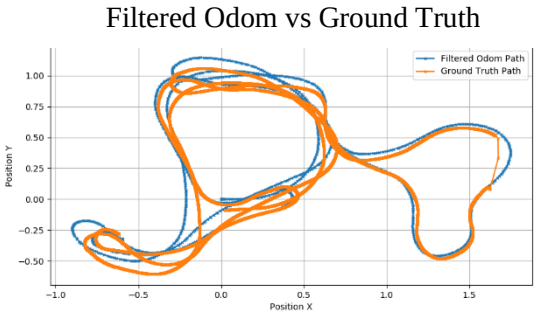

Figure 1

Upon analyzing the graph, it's evident that the ground truth and filtered odometry paths share a common overall trajectory. However, there is a notable and small difference between the two paths. This variance can be attributed to the uncertainty introduced by the EKF algorithm.

Specifically, the separation between the ground truth and filtered odometry paths is more pronounced in the beginning. This phenomenon suggests that the EKF algorithm had a greater level of uncertainty in the robot's localization, particularly in the initial stages of data collection.

In essence, the graph illustrates that while the two paths have a basic resemblance, the distinct gap between them is primarily due to the uncertainty inherent in the EKF algorithm. This uncertainty is most pronounced during the early phases of data collection when the EKF algorithm had a less precise estimate of the robot's position.

b. Gmapping

GMapping [2] is a popular algorithm used in robotics for Simultaneous Localization and Mapping (SLAM). SLAM is the process of a robot or vehicle autonomously creating a map of an unknown environment while simultaneously determining its own position within that environment. GMapping, short for "Grid-based Mapping," is a specific implementation of this algorithm.

In this case the gmapping uses the laser from the LiDAR sensor to map his environment. That means that the relevant subscribed topics from Gmapping are:

- /tf - Transforms necessary to relate frames for laser, base, and odometry

- /scan - Laser scans to create the map from

We recorded a rosbag (see **Figure 2**), with those topics in mind, from a teleoped route of the lab. This bag allowed us to test different Gmapping parameters.

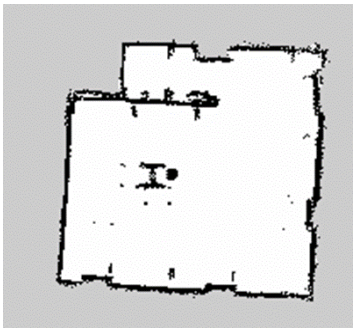

Figure 2

Kernel size

We tested the kernel size parameter. Increasing the kernel size to 2 results in greater smoothing of the map and the removal of noise (see **Figure 4**). However, an excessive increase leads to a loss of detail. Decreasing the kernel_size to 0.5 (see **Figure 3**) we can see more detail, much of which is noise from people walking, and not fixed obstacles.

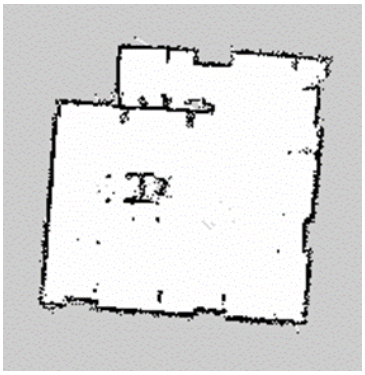

Figure 3

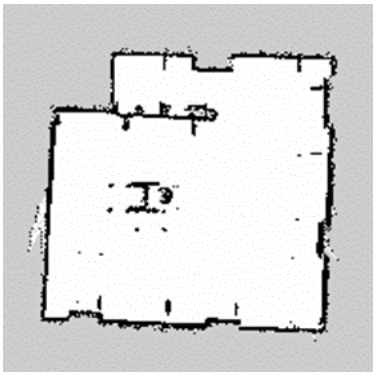

Figure 4

Particules

We conducted experiments to evaluate the impact of varying the number of particles. Increasing the number of particles to 300 (see **Figure 6**) led to more precise location estimates, although with an associated increase in computational resource usage. Nonetheless, excessively large numbers of particles introduced confusion within the algorithm, diminishing its performance. Decreasing the number of particles to 10 (see **Figure 5**) leads to less precise location estimates and requires less processing power and memory.

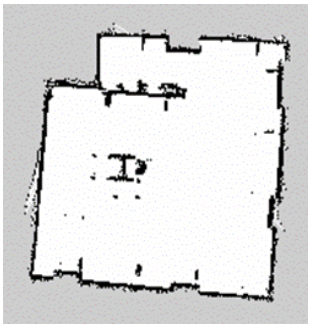

Figure 5

Figure 6

maxUrange

The maxUrange parameter defines the distance to which the laser will detect obstacles. A smaller range value like 3 (see **Figure 7**) results in a smaller laser range and consequently a smaller map size.

Figure 7

c. AMCL

Adaptive Monte Carlo Localization (AMCL) [3] is a fundamental algorithm in the field of robotics, specifically designed for solving the complex problem of Simultaneous Localization and Mapping (SLAM). AMCL plays a pivotal role in allowing a mobile robot or autonomous system to accurately determine its own position within an unknown environment while simultaneously creating a map of that environment.

At its core, AMCL is a probabilistic approach that leverages the power of particle filtering to estimate the robot's pose, which includes its position and orientation, in relation to a pre-existing map. This algorithm dynamically adjusts the number of particles it employs, adapting to the changing uncertainty and complexity of the environment. By doing so, AMCL ensures an optimal trade-off between computational efficiency and localization accuracy.

AMCL subscribes to laser scan data and an existing map, typically represented as an occupancy grid. It uses the laser data to compare the perceived environment with the known map, refining the robot's pose estimate as it moves. Additionally, it often publishes both the estimated pose and a cloud of particles that collectively represent the potential robot poses.

Dataset

In our dataset analysis we created two distinct graphs. The first graph (see **Figure 8**) serves to contrast the evolution of x and y positions over time, highlighting the disparity between odometry and AMCL estimations. The second graph (see **Figure 9**) offers a visual representation of the distinct paths traversed, with one path derived from odometry data and the other from AMCL pose estimates.

Upon examining the graphs, a noticeable consistency in their general trajectory becomes evident. However, we can see subtle disparities between the two paths. These deviations, while minor, represent the inherent intricacies of the localization process.

The nuanced variances between the odometry and AMCL pose can be attributed to the influence of uncertainty within the AMCL algorithm. AMCL leverages data from laser sensors to refine its localization estimates. As a result, these small deviations reflect the algorithm's endeavor to reconcile and align data from diverse sources.

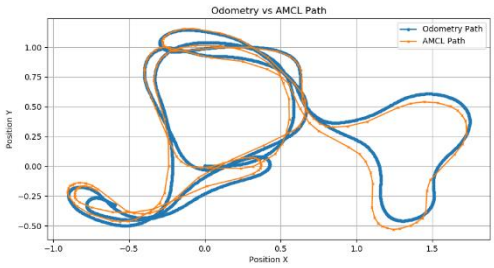

Figure 8

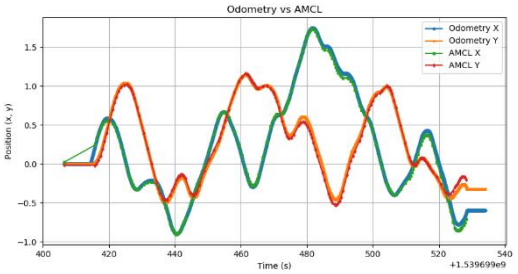

Figure 9

Real Robot

For the real robot we repeated the process we did for the dataset and obtain the very similar result to the dataset (see **Figure 10**).

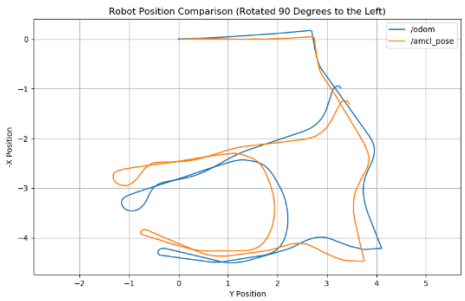

Figure 10

We also examined the covariance of the X, Y, and Theta variables (see **Figure 11**) to understand the uncertainty associated with the `amcl_pose` (robot's estimated position and orientation) over time.

The initial part of the graph shows significant fluctuations in covariance. These variations are mainly due to the initial covariance settings, reflecting the level of confidence or uncertainty in the robot's pose when data collection begins.

Early in the robot's path, it predominantly moves in the y-axis direction. At around the 10-second mark in the graph, a spike is evident. This spike signifies a temporary increase in uncertainty as the robot moves along the y-axis.

As the robot's path progresses, it changes its movement direction to the x-axis. This transition is observed around the 40-second mark in the graph, where another spike occurs. This spike indicates a brief increase in covariance, reflecting the moment when the robot changes direction and starts moving along the x-axis.

Subsequently, the covariance values stabilize and consistently remain close to zero. This pattern suggests that the `amcl_pose` estimates become more reliable and dependable. The low covariance values indicate minimal uncertainty in the robot's pose during this phase of the path.

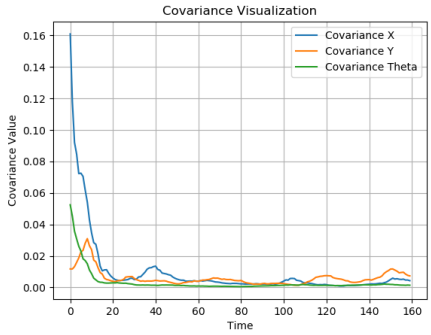

Figure 11

3. Mini project 2

a. Fundamental Concepts

For the second mini-project it's crucial to understand some important concepts, such as the local and global planner and the RRT algorithm. First, we understood the difference between a global planner and a local planner.

A global planner defines which trajectory the robot will have to follow from its current location to the destination point, considering the constraints of the environment, such as known obstacles. The result of this is a sequence of points that the robot will follow. An example of a global planner is move_base. In this mini-project we're going to implement the RRT algorithm to find those paths.

A local planner is more focused on avoiding immediate obstacles using real-time readings from the various sensors, making small corrections to the trajectory calculated by the global planner. This planner focuses more on reaching the next point on the trajectory. An example of a local planner is the DWA.

Then we understood how the planners relate to the amcl and map_server nodes. The global planner uses the map and considers the representation of the environment provided by map_server in order to calculate the best trajectory to the destination point. The amcl node also plays an important role in both the global planner and the local planner as it indicates the correct current position of where the robot is, which is crucial for calculating safe and efficient trajectories.

b. RRT algorithm

The RRT algorithm [4] [5] generates points randomly and connects them to the nearest available point, while ensuring that the points don't fall inside obstacles. It continues this process until a point is

generated within a specified goal area or until a limit is reached. Essentially, it's a method for creating a path in a cluttered environment. On top of that, we made some improvements to the algorithm.

The first optimization we've made to the rrt code was to generate random nodes so that a specific percentage of them were generated in the direction of the end goal and not completely random. this resulted in the generation of paths with several changes of direction (see **Figure 12**).

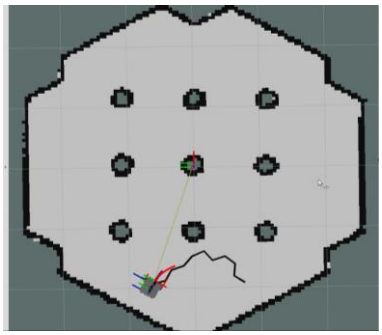

Figure 12

The second optimization we've made was to instead of the robot always following every new path it finds, it saves the distance of the current path and only updates to a new one if it has a shorter distance. This has had a very positive impact, however we were faced with the fact that if the robot collides it will not be able to calculate a new path shorter than the current one and will get stuck. We didn't include this optimization in the final version of the code because of that problem.

Another big optimization we did was to change the collision detector so that it considers an invalid point all the points with a cost (from the costmap) bigger than 220. This really improved our algorithm because it guarantees that the path is never going to be close to an obstacle.

The final optimization was the implementation of an adaptive step. This consists of changing the value of the step depending on the distance between the current location and the goal. In other words, the greater the distance, the greater the step. This resulted in a great improvement as the paths became straighter (see **Figure 13**).

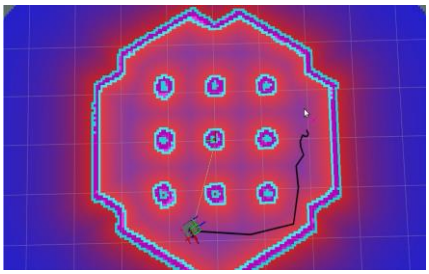

Figure 13

c. Expected results

The three main objectives of this mini project were:

- planned paths – in RVIZ;
- actual robot path (estimated by the robot) – in RVIZ;
- comparative analysis of the planned paths based on a set of relevant metrics.

d. Simulation results

Along with the optimizations to the algorithm, it was also necessary to adjust the parameters relating to both planners and the generation of the costmap. The inflation_radius and cost_scaling_factor parameters were the ones that when combined with the algorithm generated the safest and more efficient paths.

The inflation_radius is the distance from the obstacles at which the costmap will inflate the cost value. This means that as you move away from obstacles, the cost associated with these areas will decrease. Decreasing to a value like 0.5 makes that region smaller (see **Figure 14**). On the contrary, increasing the value of inflation_radius to 5 (see **Figure 15**) will mean that a larger region around obstacles is considered high cost. This means that the robot will have a greater margin of safety when planning its trajectory to avoid collisions.

Figure 14

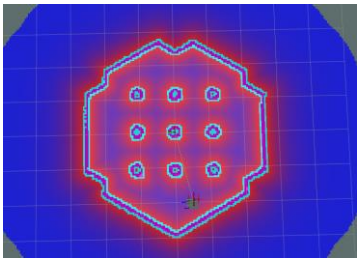

Figure 15

The cost_scaling_factor is a parameter used to adjust the costs assigned to different cells in the costmap. Since the cost_scaling_factor is multiplied by a negative in the formula, decreasing the value to 0.1 increases the resulting cost values (see **Figure 16**). In contrast, increasing the factor to 3 will decrease the resulting cost values (see **Figure 17**).

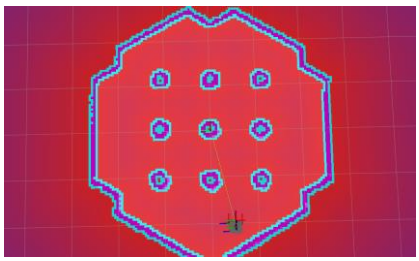

Figure 16

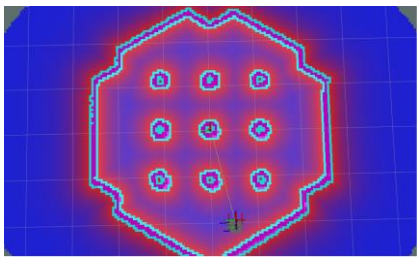

Figure 17

For our testing we fixed the inflation_radius to 2.5 and the cost_scaling_factor to 1.3. In addition to these parameters mentioned above, we have also changed the occdist_scale and path_distance_bias parameters. For the first parameter, we increased its value to 0.12, which worked. The second was increased to 42. With this, we found that the robot stayed close to the path it was given.

We then carried out tests with a higher degree of difficulty, where we placed an obstacle in the middle of the path that the robot had planned (see **Figure 18**). We note that the results were very positive, as the robot detected it and calculated a new path very quickly (see **Figure 19**).

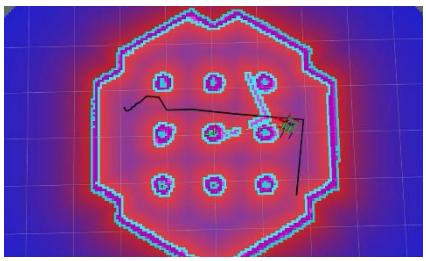

Figure 18

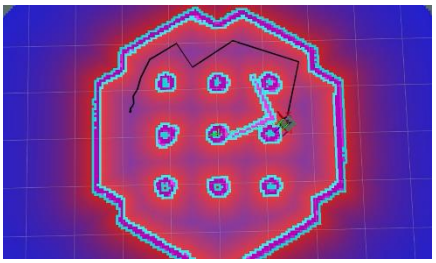

Figure 19

e. Real-robot results

Regarding the results with the real robot, we were also able to obtain very good results with the parameters defined in the simulation. We were able to successfully make the robot go to the selected goals consecutively and avoid obstacles that were placed when the robot was already moving.

In **Figure 20** we see that the robot's initial_state position is at coordinate point (3, -2.5). Then, to adjust the robot's position in Rviz using "Estimate Pose", we see that the position has been changed to the point (1.5, -2.5). In the script, we then set the first goal at point (2. 5, -5), followed by points (0, -5) and (1.8, -2). Looking at this graph, we can see that, in general, the paths taken by the robot were very efficient and safe, except for point (0, -5) where the robot still performed the recovery behaviors until it reached the goal.

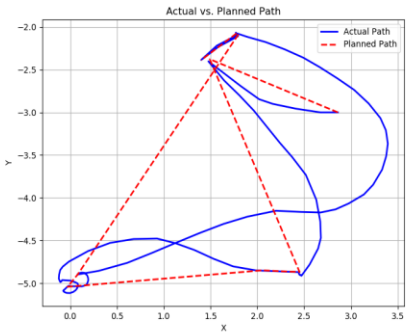

Figure 20

Another test we carried out was to create an environment in which the robot's goal was a reachable point in the middle of obstacles (see **Figure 21**). With this test we were also able to conclude that the odometry sensors and the laser are too sensitive when it comes to detecting obstacles. In order to solve this problem, we thought of optimizing our algorithm so that when the goal was not reachable, the robot would move to a point close to it and leave it up to the local planner to solve the rest of the path there. To visualize this situation you can go to https://youtu.be/bx7C3ee7VIQ.

Figure 21

4. Conclusion

This report documents our exploration into topics such as robot localization, environmental mapping, and path planning. Throughout this project, we gained a comprehensive understanding of these core aspects in robotics. We delved into localization techniques, including the Extended Kalman Filter (EKF) and Adaptive Monte Carlo Localization (AMCL), to enable robots to pinpoint their positions accurately. Additionally, we learn to use Gmapping for the creation of environmental maps. Finally, path planning came to life with the Rapidly Exploring Random Trees (RRT) algorithm, allowing robots to navigate real live terrains efficiently.

As we reflect on our journey, we recognize the potential for further improvement. Our endeavors could encompass refining data collection techniques and conducting more experiments with RRT to enhance its performance.

5. Bibliography

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

[2]https://iopscience.iop.org/article/10.1088/1757-899X/705/1/012037/meta

[3]https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9bd41c7874b7c07519d1af1b9a7a74a866cbcca9

[4]https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378

[5]https://www.researchgate.net/profile/Canberk-Suat-Gurel/publication/325473185_ROS-based_Path_Planning_for_Turtlebot_Robot_using_Rapidly_Exploring_Random_Trees_RRT/links/5b100c850f7e9b4981ff0c41/ROS-based-Path-Planning-for-Turtlebot-Robot-using-Rapidly-Exploring-Random-Trees-RRT.pdf