Introduction to Robotics: Mini-Projects Report

Group 01

Miguel Miranda ist1113191
Raquel Cardoso ist199314
Tomás Pereira ist1112273

1 Introduction

This report explores key concepts in mobile robotics [12] through two mini-projects. The first focuses on mobile robot localization, using Bayesian filters to manage uncertainties in perception and action. Accurate localization is crucial for mobile robots to navigate effectively in dynamic environments, as their success relies on determining their position relative to surroundings. Despite advances in sensor technology, challenges such as noise and occlusions can lead to localization errors. This project aims to develop robust methods using Bayesian filters to improve a mobile robot's self-localization accuracy in real-world scenarios. It involves working with ROS [15] packages for mapping and self-localization.

The second project addresses path planning and following, applying Rapidly Exploring Random Tree (RRT) algorithms [8]. Efficient path planning is essential for mobile robots to navigate complex environments while avoiding obstacles. As robots are increasingly deployed in areas like search and rescue and logistics, the ability to quickly determine an optimal path is critical. Traditional algorithms may struggle with dynamic obstacles, resulting in suboptimal paths. This project seeks to implement RRT algorithms to enhance the robot's ability to adaptively plan and follow paths in real-time.

Together, these projects provide hands-on experience with advanced localization and path planning techniques, enhancing both theoretical understanding and practical skills in robotics using ROS.

2 Mini-Project 1

The objective of Mini-Project 1 is to gain practical knowledge of mobile robot localization [14], focusing on using Bayesian filters to manage uncertainty in perception and action. We'll work with key ROS packages, including `robot_localization` [5], `gmapping` [3], and `amcl` [1], to understand the ROS navigation stack. Using the TurtleBot3 Waffle Pi [17], equipped with laser scanning and odometry sensors, we'll also learn to save and analyze data with `rosbag` [6]. By implementing and comparing techniques like the Extended Kalman Filter (EKF) [16] and Monte Carlo Localization (MCL) [9], we aim to grasp the strengths and limitations of each method.

2.1 Extended Kalman Filter (EKF-)based localization

We implemented EKF-based localization using the `robot_localization`'s `ekf_localization_node` as described in [11], integrating data from odometry, an inertial measurement unit (IMU), and a high-precision sensor - the ground-truth. This fusion aimed to accurately estimate the robot's position and orientation, with the high-precision sensor correcting for drift. Odometry data provided absolute linear and angular velocities, while the IMU contributed relative yaw measurements as incremental changes. The high-precision sensor supplied absolute positions and yaw for corrections. Localization was performed in 2D, assuming negligible z-axis variation.

Figure 1 shows the map provided in the dataset and both the real path performed by the robot, in green, and the EKF predicted path, in yellow. To be able to visualise both paths in RVIZ we implemented Python scripts that subscribe the topics `odometry/filtered` and `mocap/mocap_laser_link` and publish the EKF and ground truth positions as a new message type, `PoseWithCovarianceStamped`. In another Python script, we appended these poses as they're published to form the displayed paths. A video of the EKF algorithm in action is available here.

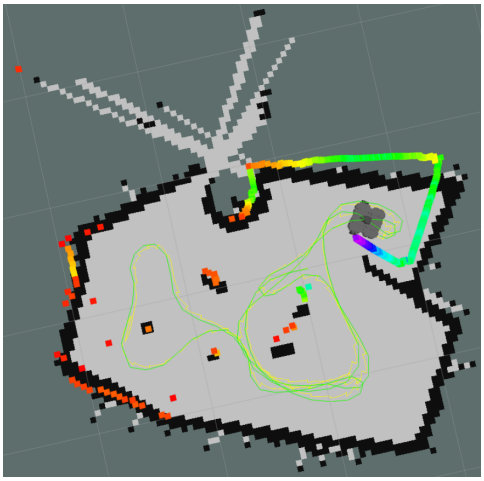


Figure 1: Comparison between EKF estimated path (yellow) and ground-truth path (green)

2.1.1 EKF Errors

To assess the accuracy of the EKF-based localization method, we compared the estimated robot position to the high-precision sensor measurements using the Euclidean distance:

$$d = \sqrt{(EKF_x - GT_x)^2 + (EKF_y - GT_y)^2}$$

This comparison is shown in **Figure 2(a)**. Additionally, to display the error along the robot's path with its associated uncertainties, we computed the confidence interval of the position, derived from the covariance matrix of the estimate. The EKF covariance matrix describes the uncertainties in the position estimate, and the confidence ellipse at each point provides a visual representation of these uncertainties:

$$P = \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix}$$

Although the EKF covariance matrix has 36 elements, only four are essential for computing the confidence intervals and ellipses at each position: P_{xx}, representing the variance in the x-direction; P_{yy}, the variance in the y-direction; and $P_{xy} = P_{yx}$, the covariance between x and y. For the 95% confidence interval along the x-axis, the following expression is used:

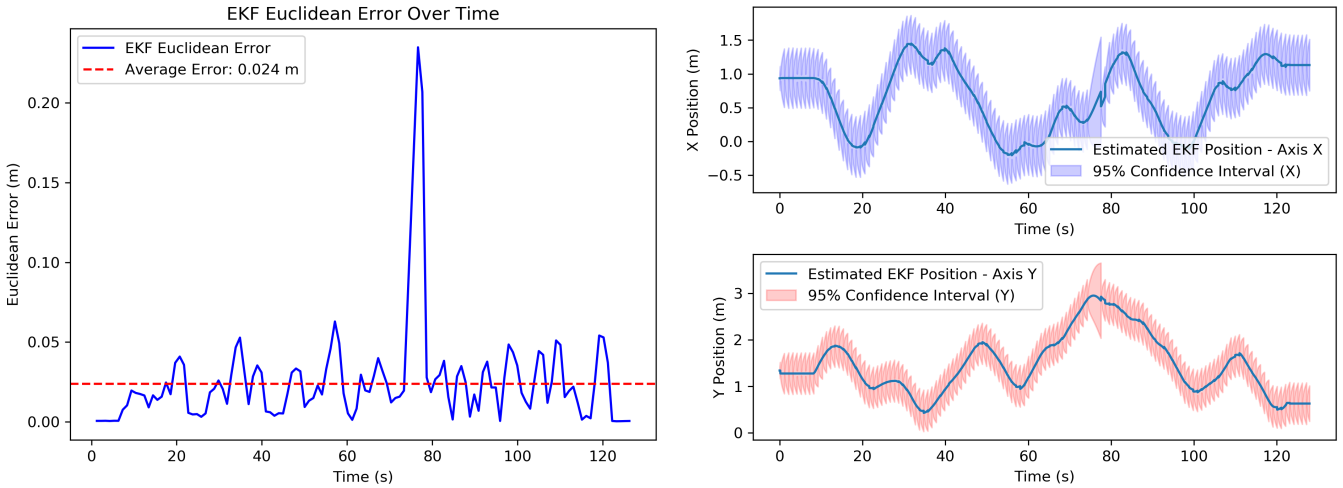
$$\left(\text{EKF}_{\text{position}_x} - 2 \cdot \sqrt{P_{xx}},\ \text{EKF}_{\text{position}_x} + 2 \cdot \sqrt{P_{xx}}\right)$$

The same formula applies to the y-axis, using P_{yy} in place of P_{xx}. This expression quantifies the uncertainty in each EKF-estimated position.

To compute the confidence ellipse, P_{xx} determines the ellipse's width, P_{yy} defines its height, and $P_{xy} = P_{yx}$ is used to compute the eigenvalues and eigenvectors of the covariance matrix. The eigenvalues determine the length of the ellipse's axes, while the eigenvectors set their orientation.

As seen in **Figure 2(a)**, the EKF's error is typically around 5 centimeters, but there is a sharp increase around the 75-second mark. By referring to **Figure 3** and **Figure 4**, it becomes clear that the high-precision sensor stopped providing data for some seconds around this time, as evidenced by the sudden rise in the covariance values and the sudden shift between ground-truth positions and EKF-estimated positons from 70s to 78s. Aside from this interval, the error behaves as expected, with the high-precision sensor consistently correcting the EKF's estimates every second.

Figure 2(b) shows the EKF-estimated positions over time, along with their associated uncertainties. As illustrated in Figure 3, the covariance is reset to zero every second, resulting in a confidence interval of $(\text{EKF}_{\text{position}_{axis}}, \text{EKF}_{\text{position}_{axis}})$ for each axis at these moments, implying no uncertainty. However, during the 70s–75s interval, the confidence interval continues to grow due to the lack of sensor data, leading to increased uncertainty. The confidence ellipse can be seen in **Figure 7**, and will be discussed in its respective section.



(a) EKF Euclidean error on estimated position **(b)** EKF estimated position with 95% Confidence Interval

Figure 2: EKF information display

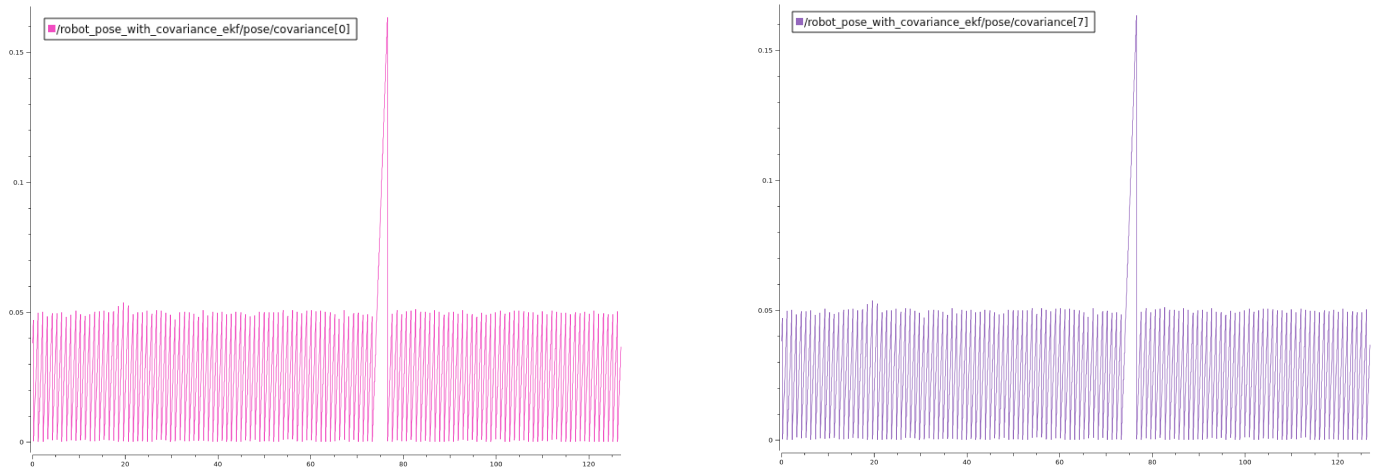


Figure 3: EKF Covariance values for x and y axis

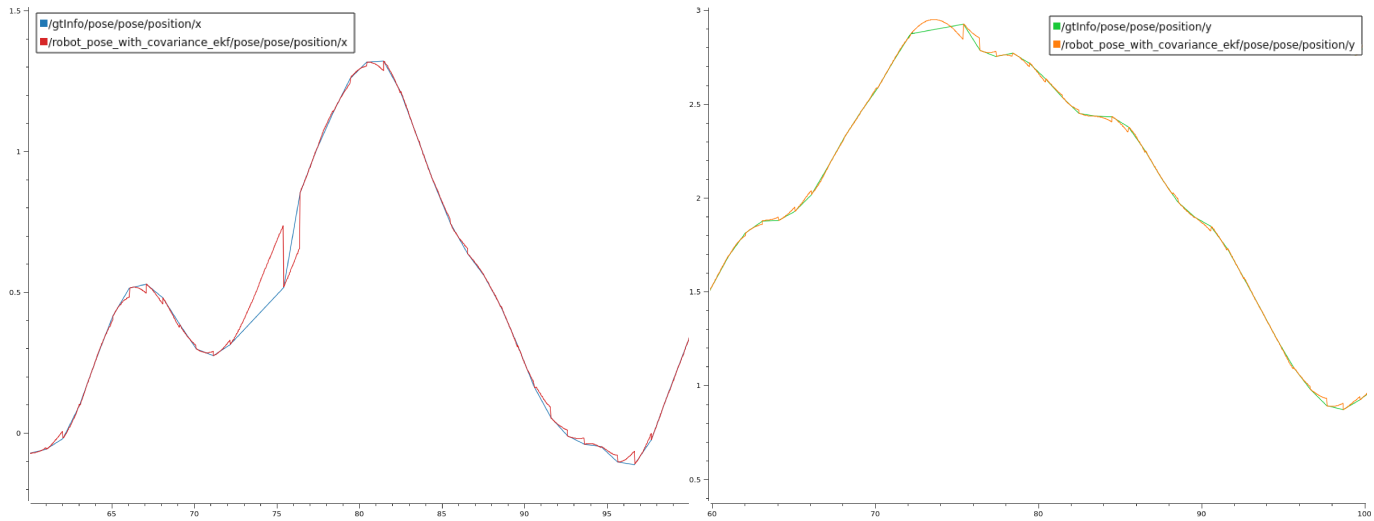


Figure 4: Comparison between EKF estimated position and ground-truth position through time in axis x and y

2.2 Adaptive Monte Carlo Localization (AMCL)

Before implementing the Adaptive Monte Carlo Localization (AMCL) algorithm, we recorded a map of the Lab classroom using a robot controlled with `turtlebot_teleop` [7]. The data collection was done through a `rosbag`, capturing all relevant topics, including laser scan data and odometry information. After collecting the data, we applied the Gmapping algorithm to generate a map from the recorded information. The resulting map is presented in **Figure 5(b)**. For an example of the map creation process using Gmapping, please refer to this video. For localization, we utilized the `amcl` package, applying it to the generated map and the recorded rosbag data. A video of the AMCL algorithm running on the recorded map can be viewed here. We fine-tuned several AMCL parameters to assess their impact on localization performance, focusing on:

1. **Initial Covariance:** Covariance settings represent the uncertainty in the robot's starting position, reflected in how spread out the particles are in the state space. We tested values of initial covariance for position (`cov_xx` and `cov_yy`), increasing them gradually from 0 to 3. While higher covariance values did not prevent the algorithm from converging, they slowed down the convergence process due to the broader spread of particles. On the other hand, a larger covariance spread helped when the initial pose was inaccurate, as it allowed the algorithm to eventually converge despite uncertainty.

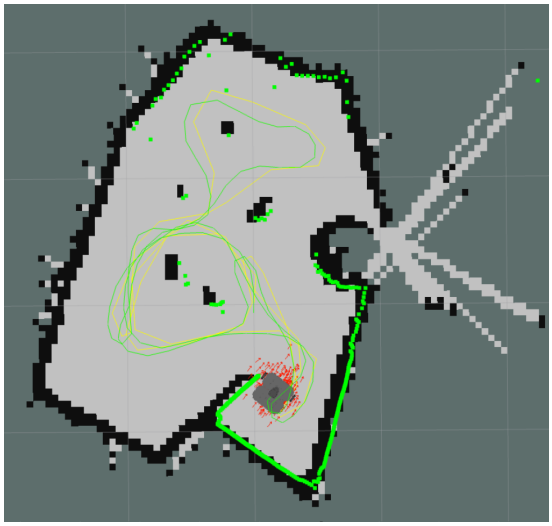
2. **Number of Particles:** We experimented with various configurations for the minimum and maximum number of particles used in AMCL. Specifically, we tested minimum particle (`min_particles`) counts of 100, 200, 500, and 1000, alongside maximum particle (`max_particles`)counts of 1000, 2000, 5000, and 10,000. After evaluating performance, we selected a minimum of 500 and a maximum of 5000 particles, as this configuration provided a good balance between accuracy and computational efficiency. A larger number of particles enhances accuracy, as it provides a denser representation of the state space, improving the algorithm's ability to converge on the correct location, especially in complex environments. However, this comes at the cost of increased computational load, which can slow down real-time performance. In contrast, too few particles can lead to premature convergence on an incorrect pose, as there may not be enough particles to represent the possible positions accurately.

3. **Initial Pose:** The accuracy of the initial pose greatly influences convergence speed. We initially tested the algorithm with an accurate pose (`intial_pose_x` = `intial_pose_y` = `intial_pose_a` = 0), resulting in rapid convergence as the particles were already centered around the true position. However, we also tested AMCL with incorrect initial poses, such as `intial_pose_x` = 3.3, `intial_pose_y` = -2.5, `intial_pose_a` = 0.3, which can be seen here. In this scenario, convergence was slower but still achieved as AMCL gradually refined the particle distribution until the true location was identified. During this process, AMCL created particle clusters around potential positions, which diminished over time as the algorithm became more confident in the robot's location. It's worth noting that in cases like this, where the initial pose is significantly wrong, having a larger number of particles can help the algorithm recover faster, as it increases the likelihood of particles being placed near the true position.

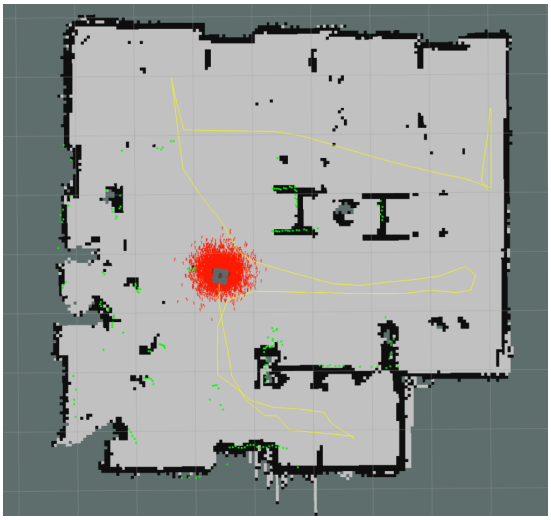
In **Figure 5**, we can see the results of the AMCL algorithm applied to both the provided dataset map (**Figure 5(a)**) and the map generated from the recorded rosbag (**Figure 5(b)**). On the dataset map, we have both the estimated AMCL path and the ground truth path performed by the robot. In **Figure 5(b)**, only the estimated AMCL path is displayed.

For comparison with the EKF, we plotted the AMCL results on the dataset. As shown in **Figure 6(a)**, AMCL's error is significantly higher than that of EKF, with the average Euclidean error being approximately seven times greater. The AMCL error generally fluctuates between 10 centimeters and 25 centimeters, indicating that AMCL accumulates error more quickly than EKF. Unlike EKF, which benefits from frequent updates from high-precision sensors, AMCL does not reset its covariance regularly but relies on its algorithm's convergence over time, as shown in **Figure 6(b)**.

In **Figure 6(b)**, the covariance for AMCL starts off very large but gradually shrinks, reflecting the algorithm's convergence as its confidence interval narrows. **Figure 7** shows the confidence ellipse at a timestamp of $\sim$ 76 seconds, a point where the high-precision sensor was not publishing updates. The elongated shape of AMCL's ellipse suggests greater uncertainty in one direction, while EKF's confidence ellipse appears more circular due to consistent updates from IMU and odometry, even in the absence of data from the high-precision sensor.

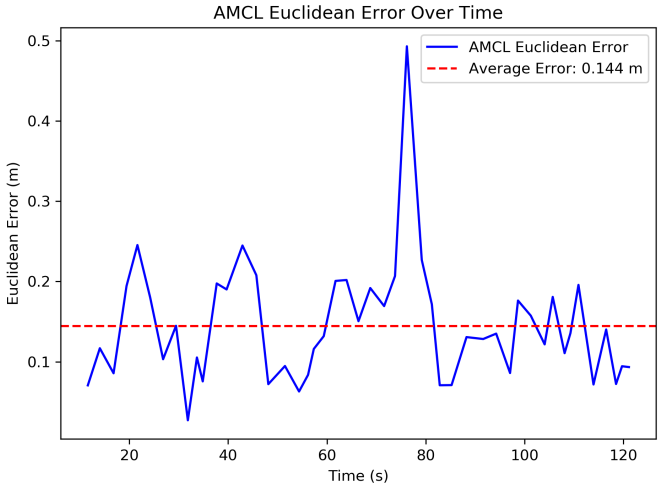


(a) Comparison between AMCL estimated path (yellow) and ground-truth path (green) performed by the robot

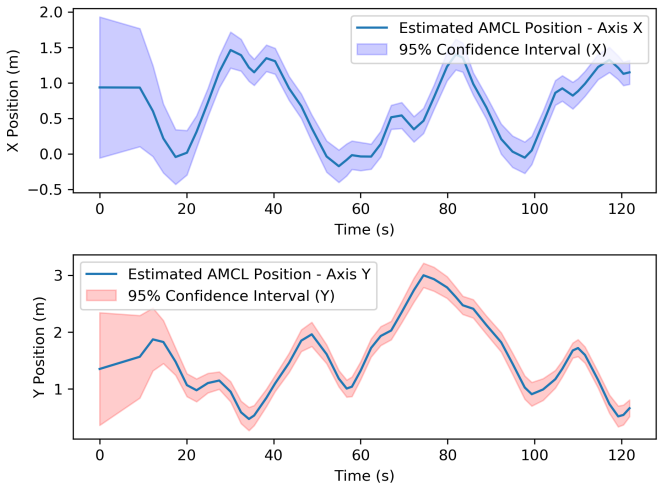


(b) AMCL estimated path (yellow) on recorded bag

Figure 5: AMCL results on the dataset and recorded bag side by side

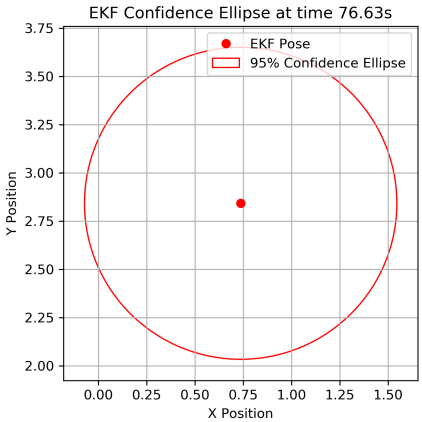


(a) AMCL Euclidean error on estimated position



(b) AMCL estimated position with 95% Confidence Interval

Figure 6: AMCL information display



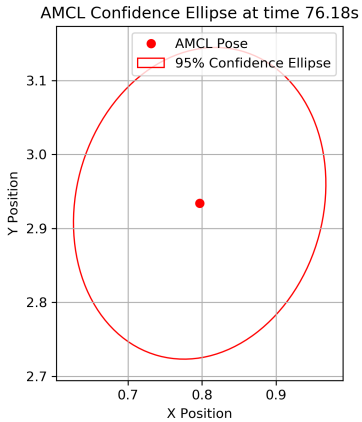


Figure 7: Comparation of EKF and AMCL confidence ellipse at 76s

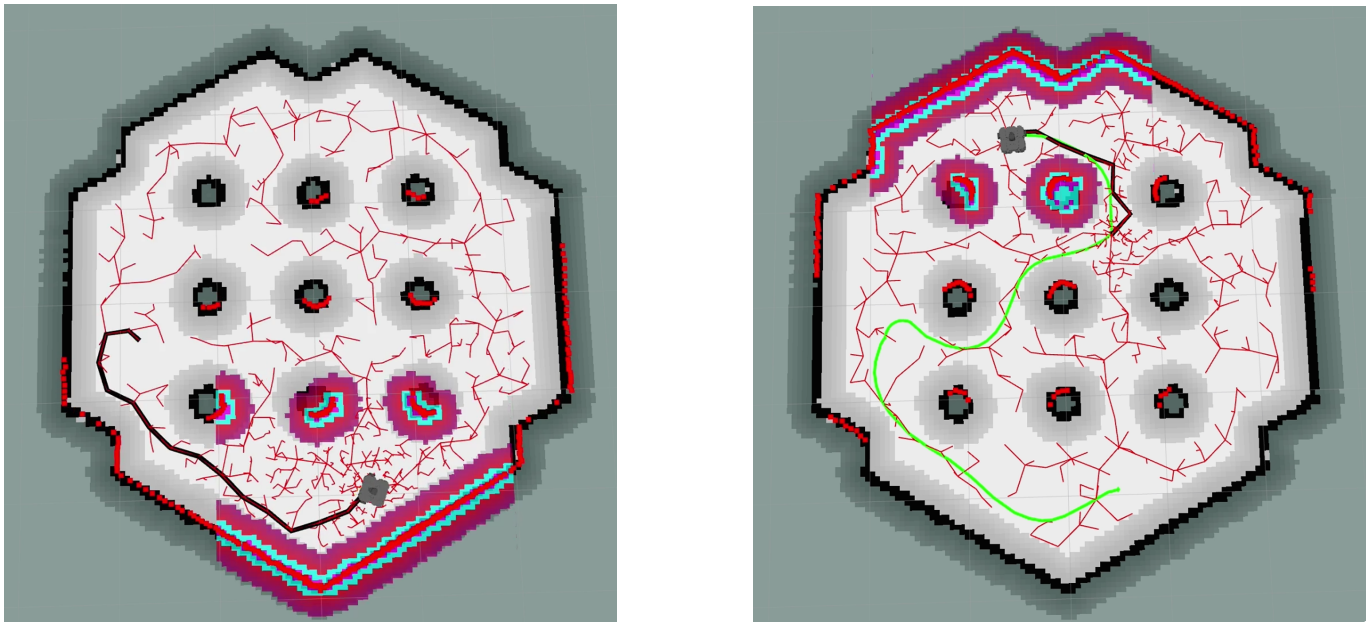
3 Mini-Project 2

The objective of the Mini-Project 2 is to learn how to utilize robot path planning and path following while utilizing multiple available ROS packages. During its execution we'll be using the Rapidly Exploring Random Trees (RRT) [8] as well as optimizing it for better performance and path finding ability. We will also work with local path planning for general obstacle detection not present in the initial cost map [2] of the room. We hope to understand, not only the benefits of this algorithm, but also how to improve them according to our needs for this and future projects.

3.1 Rapidly Exploring Random Tree (RRT)

We employed the Rapidly-Exploring Random Tree (RRT) algorithm to solve path-planning problems in mobile robotics. RRT builds a search tree by sampling random points across the map and linking each new point to the nearest node in the tree, as long as no obstacles obstruct the connection. When the algorithm places a node within a defined proximity to the goal—referred to as goal tolerance—it generates a complete path by tracing connections back to the starting node. Although RRT is conceptually simple, its effectiveness and efficiency can be fine-tuned by adjusting key parameters. These parameters include:

1. **Step:** defines the maximum allowable distance between a randomly generated point and the nearest point in the tree for a valid connection to be established.
2. **Minimum number of nodes:** sets the minimum number of nodes that must be calculated before returning a final path, enabling recalculations and potential improvements, even if a valid path is found with fewer nodes.
3. **Maximum number of nodes:** establishes the maximum number of nodes to prevent the algorithm from running indefinitely if it cannot find a path.



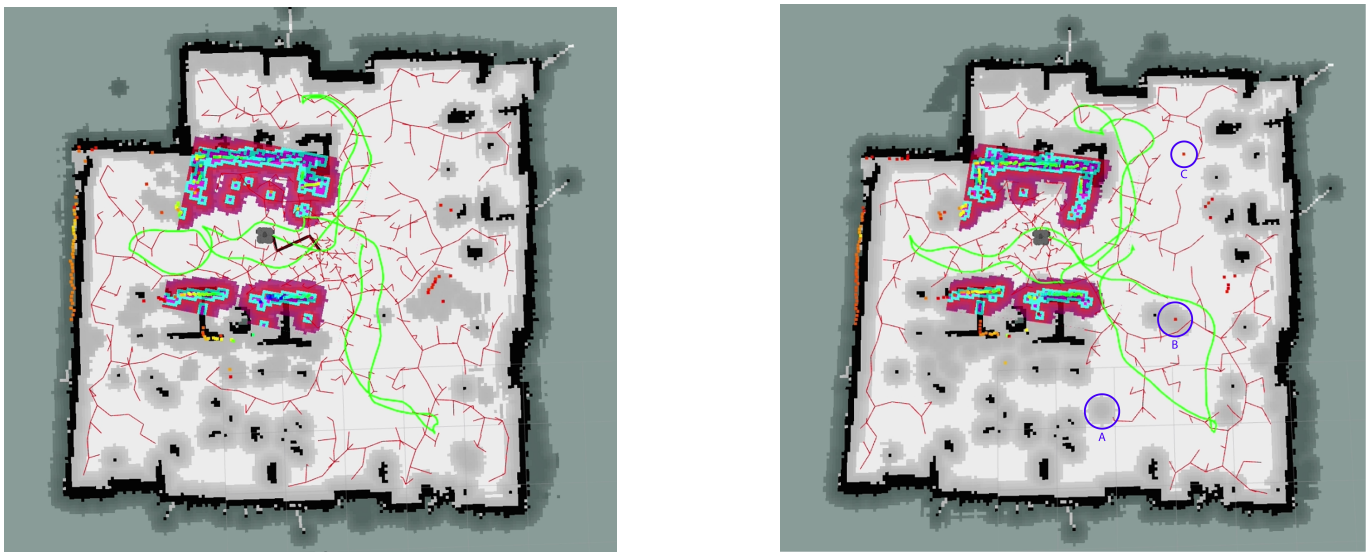
(a) RRT planned path for the first waypoint **(b)** RRT performed path after all waypoints

Figure 8: RRT simulation with multiple waypoints

In oder to conduct multiple goal paths, we implemented a Python script that subscribes to the `move_base` [4] topic, sending the next goal and waiting for a response. Once the robot successfully reaches the specified goal, the script either sends the next goal in the sequence or, upon reaching the final goal, concludes the test and shuts down.

Figure 8 shows the RVIZ visualization of a simulated path planning process using RRT with multiple goal waypoints. In **Figure 8(a)**, the initial planned path is displayed before the robot begins its movement.

In **Figure 8(b)**, the estimated path followed by the robot is shown after it has completed its full trajectory to the final waypoint. A video of the complete simulation can be visualized here.



(a) Robot estimated path without unseen obstacles

(b) Robot estimated path with unseen obstacles

Figure 9: Comparison of estimated paths with and without unseen obstacles on a real-life recording of the RRT algorithm

Figure 9 presents an RVIZ visualization of the RRT path-planning process with multiple goal waypoints in a real-world scenario. In **Figure 9(a)**, the planned path is displayed without unforeseen obstacles, while **Figure 9(b)** shows the effect of additional obstacles that were not initially accounted for. The difference between these two paths illustrates the robot's adjustments to navigate around unexpected obstacles. Videos demonstrating the path-planning outcomes are available as follows: the path without obstacles can be viewed here, and the path with obstacles can be seen here. A live recording demonstrating the robot's performance in an obstacle-present scenario is available here.

This recording provides a clear visualization of the robot's deliberate maneuver around obstacle B. Additionally, as shown in **Figure 9(b)**, obstacle A significantly impacted the robot's node placement, entirely preventing it from navigating through the lower left area of the classroom.

The placement of obstacles seen in **Figure 9(b)** was constrained by existing noise within the map. Due to limited time and high demand for robot usage, we were unable to record a cleaner map with reduced noise. Consequently, to create feasible and non-trivial paths, we strategically positioned the three obstacles as shown.

3.1.1 RRT Improvements

During our development, we identified that the most effective way to improve the overall speed of the algorithm was to optimize the generation of random points. To achieve this, we adopted a σ-Greedy approach. In the greedy selection, we use a cone originating from the robot's current position and pointing toward the goal, with an angle of $\pi/6$ to each side. We set $\sigma = 0.6$ which allows for the generation of random points in other directions, facilitating the exploration of different areas of the map. This enables the algorithm to navigate around large obstacles that may block the direct path to the goal.

3.2 Comparative Analysis

3.2.1 Step based Metrics comparison

We conducted an in-depth analysis of the impact of step size on the performance of the RRT algorithm, focusing on three different step lengths (0.1, 0.3, and 0.5), as this parameter has a significant influence on the algorithm's efficiency in path-planning tasks. We did 50 path calculations for each step size allowing for plenty of data and path recalculation.

Step Size	0.1	0.3	0.5
Best Case	0.121	0.133	0.144
Average	0.296	0.265	0.318
Worst Case	1.234	0.553	0.568

Table 1: Execution time per step size (seconds)

Step Size	0.1	0.3	0.5
Best Case	2217	3282	3648
Average	4352	4710	5318
Worst Case	12678	7688	9086

Table 2: Number of attempted nodes per step size

In Table 1, a smaller step size (0.1) shows an advantage in the best-case scenario, where execution time is slightly reduced. However, in the average and worst-case scenarios, a moderate step size of 0.3 outperforms, likely due to more efficient space exploration without high computational cost. The smallest step size (0.1) slows down average performance as it explores the map gradually, which reduces efficiency.

Table 2 presents the number of attempted nodes for each step size, illustrating the exploration behavior of the algorithm. While a step size of 0.1 requires fewer attempts in the best-case scenario, its worst-case scenario is significantly higher, indicating variability in performance. Out of the 50 path calculations, step size 0.1 shows more frequent cases with high node attempts, despite a lower overall average. Larger step sizes, on the other hand, allow for more direct traversal toward the goal, which reduces the interval between best- and worst-case scenarios, thereby minimizing computational effort.

Step Size	0.1	0.3	0.5
Best Case	76.95	84.64	85.8
Average	83.25	86.96	87.32
Worst Case	86.39	87.65	87.8

Table 3: Percentage of nodes blocked by obstacles

Step Size	0.1	0.3	0.5
Best Case	20	7	5
Average	48	17	12
Worst Case	79	34	24

Table 4: Number of nodes per path

Table 3 demonstrates that smaller step sizes result in fewer nodes blocked by obstacles, indicating improved path accuracy and adaptability in intricate maps. This suggests that smaller steps may enable more precise navigation around obstacles, enhancing node creation efficiency.

However, as Table 4 shows, smaller step sizes increase the total number of nodes required to complete the path. This shows that although smaller steps reduce blocked nodes, they lead to denser paths with higher node counts. This highlights a trade-off between path precision and computational cost. Thus, selecting an optimal step size is essential to balancing computational efficiency with adaptability in complex environments.

3.2.2 AMCL vs RRT Paths

In practice, the robot's estimated path often diverges from the path calculated by RRT due to adjustments made by the local planner. In this section, we examine the fluctuations between the planned path distance and the actual distance traveled, comparing both to the direct Euclidean distance from start to goal.

To calculate the RRT path distance, we traverse the list of nodes at the end of each calculated path and sum the distances between each node and its parent, yielding a final estimated path length. For the robot's actual path distance, we use an AMCL listener, which tracks position updates from the start of execution, calculating the incremental distance travelled since the last position update. This cumulative distance resets each time RRT requests a new path, allowing even partial paths to be compared.

An additional factor influencing path variance is obstacle detection, which may trigger the local planner to take over, resulting in deviations from the global path and a longer overall traveled distance. In **Figure 8(b)**, the difference between the AMCL estimated path and the RRT-calculated path is shown. The black line represents the RRT path between two waypoints, while the green line shows the AMCL path from the starting position. The divergence between the two paths highlights the adjustments the robot makes in response to real-world conditions.

Step Size	0.1	0.3	0.5
ΔAMCL	8.77	5.71	12.21
ΔRRT	0.05	2.07	2.01

Table 5: Distance Deviation in centimeters

Table 5 shows the average deviation between the RRT-calculated path (ΔRRT) and the AMCL-estimated path (ΔAMCL) in relation to the real Euclidean distance from the starting point to the goal. These results were obtained with the local planner parameters `path_distance_bias` = 45 and `goal_distance_bias` = 20, meaning the local planner gave priority to following the calculated path. Given that the average Euclidean distance from the starting point to the goal is 433.66 centimeters the variances shown can be considered rather small. The minimal RRT deviations suggest a strong alignment between the planned and intended paths, limiting interpretability. Conversely, the AMCL path deviations demonstrate significant variability, reflecting the robot's adjustments to dynamic conditions. Notably, AMCL's larger deviations at smaller (0.1) and larger (0.5) step sizes indicate less stable path adherence, while the moderate step size (0.3) results in the smallest deviation. This suggests that a balanced step size optimizes both path accuracy and adaptability, underscoring the importance of AMCL in refining navigation amidst environmental changes.

After conducting various experiments and analyzing the results shown previously, we have decided to proceed with a `step_size` of 0.3. We believe this value provides the best balance of performance and stability without compromising precision in navigation. For the remaining parameters of the RRT algorithm, we established the following settings: a `min_nodes` count of 500, a `max_nodes` count of 5000, and a `goal_tolerance` of 0.2 meters.

3.3 Obstacle Detection

3.3.1 General Obstacle Detection

For general object detection, we utilized a Cost2D map of the room, enabling the robot to effectively avoid all known obstacles and prevent it from getting stuck in any previously identified areas. The `obstacle_range` is set to 2.0 meters, ensuring that any obstacles within this distance are detected and considered in path planning. Additionally, a `raytrace_range` of 3.5 meters allows the robot to anticipate obstacles farther ahead, improving its ability to navigate around them. An `inflation_radius` of 0.25 meters creates a buffer zone around obstacles, adding an extra margin of safety to avoid collisions. Finally, a `cost_scaling_factor` of 3.0 increases the traversal cost near obstacles, prompting the robot to select paths that maintain a safe distance from obstacles. Together, these parameters ensure a clear, functional path across the map for smooth navigation.

3.3.2 Local Path Planning

For the detection of previously unknown objects, we utilize a Dynamic Window Approach (DWA) local planner. Using the robot's LiDAR, this planner detects obstacles that may appear along the calculated path. The DWA planner is set with a maximum translational velocity of 0.17 m/s and a minimum of 0.05 m/s, providing controlled movement as it navigates. The controller operates at a frequency of 10 Hz, updating the robot's movement commands regularly. If an obstacle is detected, the robot initiates its recovery process using the default procedure: first trying to back up, and if this isn't possible, rotating to gather new LiDAR readings.

The `planner_patience` is set to 5 seconds, giving the planner this period to find a feasible path before retrying. The `controller_patience` is set to 15 seconds, after which the controller will initiate recovery if it cannot proceed. A `conservative_reset_distance` of 0.3 meters ensures the robot clears obstacles before resuming its path. The `planner_frequency` is 0.01 Hz, to avoid the generation of new global paths unnecessarily.

To handle oscillations, the `oscillation_timeout` is set to 10 seconds, and the `oscillation_distance` of 0.05 meters helps the robot detect if it is repeatedly moving back and forth. Once the robot is far enough from obstacles, it requests a new path from the global planner and resumes navigation.

4 Conclusion

This project has offered valuable hands-on experience in fundamental robotics concepts, particularly in localization and path planning.

In Mini-Project 1, we focused on mobile robot localization, applying the Extended Kalman Filter (EKF) and Adaptive Monte Carlo Localization (AMCL) methods to enhance understanding of state estimation. Through the implementation of EKF-based localization, we gained insights into integrating data from multiple sensors to manage uncertainty and achieve accurate positional estimates. The AMCL experiment underscored the importance of parameter settings, such as initial covariance and particle count, in determining convergence accuracy and speed.

Mini-Project 2 expanded our understanding by applying the Rapidly-Exploring Random Tree (RRT) algorithm for path planning. This experience highlighted the trade-offs between path accuracy and computational efficiency, particularly when adapting to new or unexpected obstacles. We saw the significance of factors like step size and node density in optimizing the algorithm's performance and ensuring reliable navigation.

Together, these projects have strengthened our technical foundation in robotics. Yet, there are still areas for improvement. While we achieved notable progress in localization and path planning, adding vision-based data could further enhance localization accuracy, especially in challenging terrains. Exploring alternative algorithms such as RRT* [13] or RRT-A* [10] might also improve path planning efficiency. These learnings provide a robust starting point for future work, as continued refinement and testing in more varied environments could elevate the robot's adaptability and performance.

References

[1] amcl - ROS Wiki — wiki.ros.org. https://wiki.ros.org/amcl. [Accessed 21-10-2024].

[2] costmap_2d - ROS Wiki — wiki.ros.org. https://wiki.ros.org/costmap_2d. [Accessed 21-10-2024].

[3] gmapping - ROS Wiki — wiki.ros.org. https://wiki.ros.org/gmapping. [Accessed 21-10-2024].

[4] move_base - ROS Wiki — wiki.ros.org. https://wiki.ros.org/move_base. [Accessed 21-10-2024].

[5] robot_localization - ROS Wiki — wiki.ros.org. https://wiki.ros.org/robot_localization. [Accessed 21-10-2024].

[6] rosbag - ROS Wiki — wiki.ros.org. https://wiki.ros.org/rosbag. [Accessed 21-10-2024].

[7] turtlebot_teleop - ROS Wiki — wiki.ros.org. https://wiki.ros.org/turtlebot_teleop. [Accessed 21-10-2024].

[8] Devin Connell and Hung Manh La. Dynamic path planning and replanning for mobile robots using rrt. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1429–1434, 2017.

[9] Dellaert et al. Monte carlo localization for mobile robots. [Accessed 21-10-2024].

[10] Jiadong Li, Shirong Liu, Botao Zhang, and Xiaodan Zhao. Rrt-a* motion planning algorithm for nonholonomic mobile robot. In *2014 Proceedings of the SICE Annual Conference (SICE)*, pages 1833–1838, 2014.

[11] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014. [Accessed 21-10-2024].

[12] Ulrich Nehmzow. Mobile robotics: A practical introduction. 01 2003. [Accessed 21-10-2024].

[13] Iram Noreen, Amna Khan, and Zulfiqar Habib. Optimal path planning using rrt* based approaches: A survey and future directions. *International Journal of Advanced Computer Science and Applications*, 7, 11 2016. [Accessed 21-10-2024].

[14] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University - Computer and Information Sciences*, 34(8, Part B):6019–6039, 2022. [Accessed 21-10-2024].

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, January 2009. [Accessed 21-10-2024].

[16] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, 2004. [Accessed 21-10-2024].

[17] WafflePi3 Turtlebot. ROBOTIS e-Manual — emanual.robotis.com. https://emanual.robotis.com/docs/en/platform/turtlebot3/features/. [Accessed 21-10-2024].