AMR Team 03 Project Report

Saloni Pathak, Ahsan Kabir Nuhel, Md Azim Khan

***Abstract*—This project presents the development of an autonomous mobile robot using the Robot Operating System 2 (ROS2) framework for mapping, navigation, localization, and exploration in a partially known environment. The system is structured into four key milestones: generating an occupancy grid map using Simultaneous Localization and Mapping (SLAM), implementing global and local motion planning with A* and the potential field planner, ensuring accurate localization with Monte Carlo Localization (MCL), and enabling autonomous exploration through a frontier-based strategy. The project addresses significant challenges such as sensor noise, localization drift, real-time obstacle avoidance, and efficient path planning. By integrating these components, the robot can autonomously navigate and adapt to dynamic environments while continuously refining its map. This report details the methodologies employed, the challenges encountered, and the solutions implemented to enhance the robot's autonomy and robustness in real-world scenarios. The complete project code is available on GitHub: here.**

***Index Terms*—ROS2, A* Algorithm, Potential Field, SLAM, Autonomous Navigation, Environment Exploration.**

I. Introduction

Autonomous mobile robots are essential in fields like logistics, search-and-rescue, industrial automation, and exploration. Navigating unknown environments, avoiding obstacles, and exploring new areas are key to achieving full autonomy. This project develops an autonomous navigation system for the Robile platform using ROS2, integrating SLAM for mapping, A* for global path planning, the Potential Field method for obstacle avoidance, Monte Carlo Localization for accurate positioning, and frontier-based exploration for complete map coverage. Structured into four milestones—mapping, navigation, localization, and exploration—the project ensures the robot can operate autonomously in real-world conditions. This report outlines the methods used, challenges faced, and solutions implemented, offering insights into building a reliable robotic navigation system.

II. Mapping

Mapping is a fundamental step in autonomous navigation, as it provides the robot with a structured representation of its surroundings, allowing for effective path planning and obstacle avoidance. Throughout this project, we used the Robile 4 platform and implemented SLAM (Simultaneous Localization and Mapping) using the robile navigation package. The mapping process was conducted following the official Robile-AMR documentation [8], ensuring a structured and reproducible workflow.

Initially, we attempted to map an environment containing two obstacles to test the robot's ability to detect and navigate around multiple objects. The mapping process involved launching the robot with:

```
ros2 launch robile_bringup robot.launch.py
```

and starting the mapping node using:

```
ros2 launch robile_navigation online_async.
    launch.py
```

The robot was then manually guided to explore the area while real-time map updates were visualized using RViz2. However, the resulting map (Fig. 1) displayed noticeable distortions, particularly around the obstacle edges, which affected the accuracy of localization and motion planning.

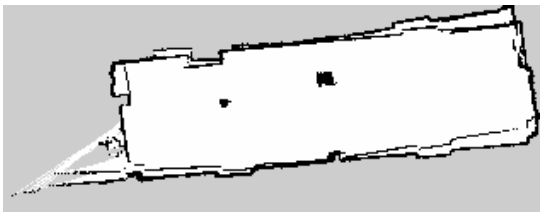


Fig. 1: Generated map with two obstacles (Map 01). Distortions are noticeable near obstacles.

To address these challenges, we opted for a simplified environment with only one obstacle, resulting in a clearer and more precise map (Fig. 2). This improved approach minimized distortions and provided a more reliable occupancy grid for navigation. The final map was saved using:

```
ros2 run nav2_map_server map_saver_cli -f ~/maps
    /map_02
```

Fig. 2: Final generated map with one obstacle (Map 02). Improved accuracy and reduced distortions.

Key improvements observed in the simplified mapping approach included:

- Reduced LiDAR sensor noise by optimizing filter parameters.
- Minimized odometry drift using motion correction algorithms.
- Simplified navigation paths, making path planning more efficient.

*Submitted to the Department of Computer Science at Hochschule Bonn-Rhein-Sieg in Autonomous Systems

‡Submitted in March 2025

The final optimized map served as a solid foundation for path and motion planning, ensuring that the robot could navigate effectively in its environment.

III. Path and Motion Planning

Path and motion planning are essential components of autonomous navigation, enabling a mobile robot to move efficiently towards a goal while avoiding obstacles. This milestone involved integrating a global planner using A* and a local planner using the Potential Field Method (PFM) to ensure smooth and dynamic navigation. The global planner computes an optimal path using a pre-existing map, while the local planner is responsible for real-time obstacle avoidance. The interaction between these two planners allows the robot to navigate both structured and unstructured environments effectively.

A. Global Planner (A*)

The global planner subscribes to the occupancy grid map (/map), which provides a representation of the environment, indicating free spaces, obstacles, and unknown areas. It also subscribes to the start pose (/startpose) and goal pose (/goal-pose) topics, which provide the robot's initial position and the target destination. The A* algorithm is used to compute the shortest path by evaluating possible paths based on a cost function. This function consists of the actual cost of movement from the start to a given node and a heuristic estimate of the cost to reach the goal. The grid-based representation of the environment allows A* to find an efficient path while ensuring collision avoidance. Once the global path is computed, it undergoes path simplification to remove redundant waypoints, reducing unnecessary turns and ensuring smoother navigation. The computed path is then published to the /path_line topic, which the local planner subscribes to for execution.

The A* algorithm operates by searching for the lowest-cost path through a grid representation of the environment. It initializes the starting node and expands the search by evaluating neighboring nodes based on their cost. The cost function consists of two components:

$$f(n) = g(n) + h(n)$$

where $g(n)$ represents the actual cost from the start to the current node and $h(n)$ is a heuristic estimating the cost from the current node to the goal. The Euclidean distance was chosen as the heuristic function to efficiently guide the search towards the goal. As the search progresses, the path is reconstructed from the goal back to the start, ensuring an optimal trajectory. However, raw paths generated by A* often contain unnecessary waypoints, which were removed using a path simplification technique that filters out collinear points.

B. Local Planner (Potential Field Method)

The local planner subscribes to the global path /path_points, the laser scan data (/scan), and the odometry information (/odom) to continuously update the robot's position and detect nearby obstacles. The Potential Field Method (PFM) is used to adjust the robot's movement in real time. This method applies an attractive force pulling the robot toward the goal and a repulsive force pushing it away from detected obstacles. The combined effect of these forces determines the robot's velocity and heading direction. The local planner continuously processes incoming LaserScan data to detect obstacles and compute repulsive forces, ensuring that the robot does not collide with dynamic objects. The repulsive force is inversely proportional to the distance from an obstacle, meaning that the robot reacts more strongly to closer objects. By combining attractive and repulsive forces, the robot is able to adjust its trajectory dynamically, prioritizing both goal-seeking behavior and collision avoidance.

The attractive force is computed using:

$$F_{\text{att}} = k_{\text{att}} \times \frac{P_{\text{goal}} - P_{\text{robot}}}{d_{\text{goal}} + \epsilon}$$

where P_{goal} and P_{robot} are the goal and robot positions, respectively. The repulsive force is calculated as:

$$F_{\text{rep}} = -k_{\text{rep}} \times \left(\frac{1}{r} - \frac{1}{r_{\max}}\right) \times \frac{1}{r^2}$$

where r is the distance to the nearest obstacle. This ensures that the robot repels strongly from nearby objects while maintaining a safe path toward the goal.

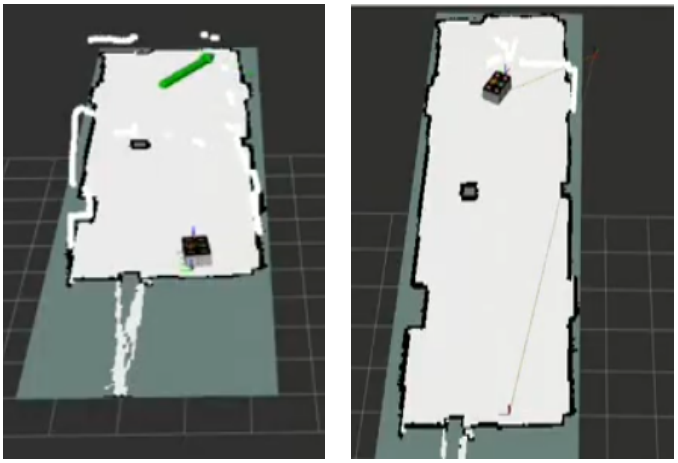
C. Challenges and Solutions

During implementation, several challenges arose that required adjustments. One major issue was the local minima problem, where the robot could become trapped between opposing forces, leading to oscillations or stagnation. To address this, we introduced random perturbations in the force calculation, allowing the robot to escape potential deadlocks. Another challenge was waypoint oversampling, where A* generated excessive waypoints, making the robot's movement inefficient. To counteract this, a waypoint filtering technique was implemented, selecting every n-th waypoint while maintaining smooth trajectory transitions. Additionally, sensor noise from the LiDAR scanner sometimes led to false obstacle detections, causing unnecessary path deviations. To improve accuracy, filtering techniques were applied to smooth out erroneous readings, ensuring a more stable navigation response.

D. Integration of Global and Local Planners

The integration of global and local planners significantly improved the robot's navigation performance. The global planner ensured an optimized high-level trajectory, while the local planner provided dynamic adjustments to avoid obstacles in real-time. Extensive testing in both simulated and real environments demonstrated that the robot could reliably reach its goal while dynamically avoiding unexpected obstacles. The hybrid path planning approach proved effective in structured environments, and its real-time adjustments allowed the robot to handle cluttered spaces efficiently.

This approach ensures that the robot can autonomously operate in partially known environments while continuously adapting to new obstacles.



(a) Initial pose and goal position of the robot.

(b) Robot following the goal using A*.

Fig. 3: Path planning using A*: (a) Initial pose and goal setting, (b) Robot following the computed path.

IV. Localization

Localization is a crucial step in autonomous navigation, allowing the robot to determine its position within the previously mapped environment. In this project, we implemented Monte Carlo Localization (MCL), a particle filter-based approach, using real-time sensor data from LiDAR and odometry to estimate the robot's pose. MCL is widely used in probabilistic robotics as it maintains multiple hypotheses about the robot's location, refining them as new observations are received. The localization process involves four main steps: particle generation, motion update, measurement update, and resampling.

Our provided code implements Monte Carlo Localization (MCL), a particle filter-based localization algorithm that estimates the robot's position within a known occupancy grid map. This method maintains a set of weighted particles representing possible poses of the robot and iteratively refines them based on odometry and LiDAR sensor data. Initially, a set of particles is sampled around a given initial pose, each assigned an equal weight. As the robot moves, the motion model updates particle positions using odometry data, incorporating Gaussian noise to account for uncertainty. The measurement model then evaluates how well each particle's predicted laser scan aligns with actual sensor readings, assigning weights accordingly. A resampling step follows, where particles with higher weights (i.e., those that better match the environment) are duplicated, while lower-weight particles are discarded. The final estimated pose is computed as the weighted average of the particles and published to /estimated_pose, while all particle hypotheses are visualized through the /hypotheses topic.

At the start of localization, we initialized the robot's position using an estimated pose from RViz. This was achieved through the 2D Pose Estimate tool, where an initial hypothesis about the robot's position was set manually. Once the initial pose was provided, the particle filter algorithm generated multiple particles around this location, representing possible poses of the robot. Each particle was associated with a weight, reflecting the likelihood of the robot being at that position. The set of particles was initialized as a Gaussian distribution centered around the estimated pose to provide a reasonable starting point.

As the robot moved, the motion update step adjusted each particle's position using odometry data. Instead of continuously updating particles, we used a threshold of 0.3 meters in displacement and 0.3 radians in rotation to ensure computational efficiency. When the robot's movement exceeded this threshold, all particles were updated accordingly, incorporating small Gaussian noise to account for sensor inaccuracies and real-world uncertainty. This ensured that the particle cloud dynamically adapted to the robot's movement.

The measurement update step refined the particle weights using LiDAR sensor readings. Each particle simulated a laser scan based on its pose and compared it with the actual scan received from the robot's LiDAR. Particles whose simulated scans closely matched the real sensor readings were assigned higher weights, whereas those that significantly deviated were given lower weights. This step effectively discarded incorrect hypotheses and retained the most probable locations.

To maintain a manageable number of particles while preserving accuracy, we performed resampling at each iteration. A stochastic universal resampling method was used, which ensured that higher-weight particles were more likely to be replicated in the next iteration, while low-weight particles were discarded. Additionally, to prevent the algorithm from converging too quickly to a potentially incorrect pose, we reinforced high-weight particles by generating additional hypotheses around them. This provided robustness in dynamic environments where sensor noise and unexpected obstacles could affect localization accuracy.

Several key parameters govern the localization process, such as the minimum movement threshold (dist_thresh = 0.3) and rotation threshold (theta_thresh = 0.4), which ensure updates only occur when the robot moves significantly. The algorithm also integrates stochastic universal sampling to maintain particle diversity while refining the estimate. The sensor model extracts occupied cells from the map and computes expected distances to obstacles, comparing them to actual LiDAR measurements. The difference in expected and observed distances determines particle weights, where lower discrepancies result in higher weights.

Our implementation followed the structure of MCL-based localization in ROS2. The MonteCarloLocalizer node subscribed to multiple topics: /initialpose for setting the robot's initial pose, /odom for receiving odometry data, /scan for laser scan updates, and /map for loading the static environment map. The node also published data to /mcl_pose to output the estimated pose, /mcl_path to visualize the estimated trajectory, and /particlecloud to display the particle distribution in RViz.

One of the major challenges encountered during implementation was the degradation of localization accuracy due to sensor noise and drift in odometry. To mitigate this, we applied adaptive particle reinforcement, where additional particles were generated around high-confidence hypotheses. Another challenge was high computational load, particularly when processing LiDAR data with 150 range readings. To optimize performance, we reduced the number of particles

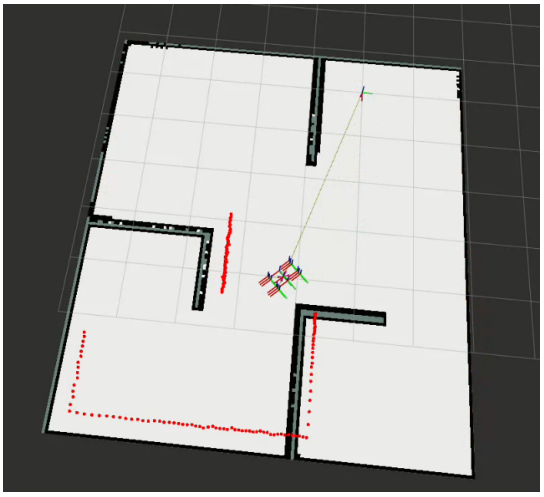


Fig. 4: Robot with particle filter in simulation

dynamically, adjusting them based on localization confidence. Additionally, discrepancies between simulated and real-world conditions led to occasional divergence in localization. This was addressed by tuning the motion noise parameters and improving the laser scan matching process.

In conclusion, our MCL implementation successfully estimated the robot's position with high accuracy, enabling reliable autonomous navigation. The use of adaptive resampling and real-time sensor integration significantly improved the robot's ability to localize itself, even in cluttered environments. Future improvements could include integrating Kalman Filtering for odometry correction and optimizing particle selection strategies to further enhance localization efficiency.

V. Environment Exploration

The final stage of this project involved enabling the robot to autonomously explore unknown environments while incrementally updating the occupancy grid map. The goal was to allow the robot to intelligently identify unexplored areas, generate appropriate exploration goals, and navigate efficiently while avoiding obstacles. The exploration strategy was implemented using a frontier-based approach, where the robot continuously identified the boundary between explored and unexplored regions and set frontier points as new exploration goals. This ensured systematic coverage of the environment while avoiding redundant navigation.

A. Frontier-Based Exploration Strategy

The robot's exploration strategy relied on the occupancy grid map generated by SLAM, which classified the environment into three types of regions: free space (0), occupied space (1), and unknown space (-1). The algorithm identified frontiers by scanning for free space cells that bordered unknown areas. These frontier cells served as potential exploration goals.

Once a frontier was identified, the robot used the A* global path planner to determine the optimal path from its current position to the selected frontier goal while avoiding large obstacles. The computed path was then followed using the local potential field planner, which dynamically adjusted the robot's movements based on real-time obstacle detection. If the A* planner determined that a goal was unreachable due to obstacles, the system automatically discarded the goal and generated a new frontier point. This ensured continuous exploration without requiring manual intervention.

B. Implementation in ROS2

The implementation of the exploration system was based on ROS2 and involved multiple components interacting through topic subscriptions and publications. The key elements included:

1. Occupancy Grid Map Subscription: The robot subscribed to the `/map` topic to receive real-time updates of the environment. The received occupancy grid was processed into a 2D array to classify free space, obstacles, and unexplored areas. This served as the foundation for frontier detection.

2. Frontier Detection: The algorithm scanned the occupancy grid for free-space cells adjacent to unknown areas, marking these as frontier points. The function `find_frontier_cells()` iterated through the occupancy grid and identified potential exploration targets.

3. Goal Selection and Publishing: Once a frontier was detected, it was converted from grid coordinates to real-world coordinates and published as a `PoseStamped` message under the topic `/goal_pose`. A random yaw angle was assigned to the goal to ensure varied exploration paths.

4. Goal Validation: After a goal was generated, the A* planner attempted to compute a path. If the goal was unreachable, an update message was published on the topic `/progress_result`, indicating failure. The system then discarded the invalid goal and generated a new frontier point, ensuring seamless exploration.

5. Navigation Execution: Once a valid goal was selected, the robot followed the computed path using the local potential field planner. The local planner adjusted the robot's movement in real time to avoid obstacles while steering toward the target.

C. Challenges Faced

During implementation, several challenges were encountered. The primary issue was inefficient goal selection, where the robot initially selected frontiers that were too close to obstacles, making navigation difficult. This was mitigated by refining the frontier selection criteria to ensure safer navigation zones. Additionally, in environments with complex layouts, the A* planner occasionally failed to find a valid path due to tight spaces. To address this, the robot was programmed to generate alternative goals dynamically when encountering an impassable frontier. Finally, to improve navigation smoothness, motion parameters in the potential field planner were fine-tuned to prevent excessive oscillations in the robot's trajectory.

D. Results and Observations

The exploration strategy was successfully implemented and tested in simulation using the Robile platform. The key observations from testing included:

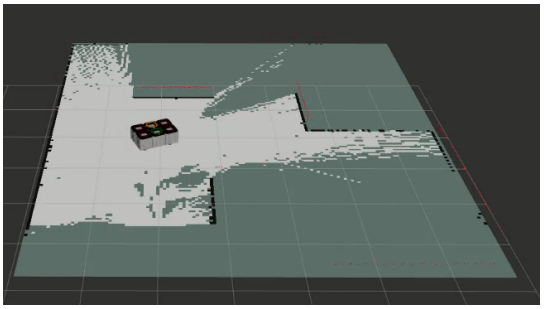


Fig. 5: Robot exploration in simulation

- **Efficient Frontier Identification**: The algorithm accurately detected frontier points, ensuring systematic exploration of the environment.
- **Accurate Path Planning**: The A* planner computed effective paths, allowing the robot to reach unexplored areas while avoiding obstacles.
- **Dynamic Goal Handling**: If a goal was unreachable, the system generated new frontier points, ensuring continuous and adaptive exploration.
- **Obstacle Avoidance**: The local potential field planner enabled the robot to dynamically avoid obstacles in real-time while following the global path.

By the end of the exploration phase, the robot successfully mapped the environment, creating a closed-loop map while efficiently covering all unexplored regions. The integration of SLAM, frontier-based goal selection, A* path planning, and real-time obstacle avoidance allowed the robot to autonomously explore unknown areas without human intervention.

VI. Conclusion

This project successfully implemented an autonomous mobile robot using ROS2 for mapping, navigation, localization, and exploration. The robot built an occupancy grid map, navigated using A* for global path planning and the Potential Field method for local obstacle avoidance, localized itself with Monte Carlo Localization, and explored unknown areas using frontier-based exploration. Challenges such as sensor noise, localization drift, and path optimization were addressed through tuning and algorithmic improvements. The integration of these techniques allowed the robot to adapt dynamically and update its understanding of the environment. This project demonstrates how autonomous robots can efficiently map and explore unknown spaces, contributing to advancements in robotic navigation.

VII. Team Members Contribution

- **Saloni Pathak**: Potential field and A* implementation, Exploration implementation, debugging, testing, and Project report
- **Ahsan Kabir Nuhel**: Mapping, Localization implementation, Exploration optimization, debugging, and testing.
- **Md Azim Khan**: Mapping, A* optimization, Localization optimization, debugging, testing, project report, and video editing.

VIII. Github Repository link

The complete project code is available on GitHub: here.

IX. Project Demonstration Videos

The recorded videos of the project are available at the following links:

- Mapping
- Path and motion planning
- Localization
- Exploration

References

[1] Learn ROS2, "Probabilistic Robotics: Particle Filter." Available: https://www.learnros2.com/probabilistic-robotics/particle-filter
[2] R. Labbe, "Kalman and Bayesian Filters in Python," GitHub. Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/12-Particle-Filters.ipynb
[3] Robile-AMR Documentation, "Demo Localization." Available: https://robile-amr.readthedocs.io/en/latest/source/Tutorial/Demo%20Localization.html
[4] D. Nirwan, "Monte Carlo Localization Implementation," GitHub. Available: https://github.com/debbynirwan/mcl
[5] YouTube, "Particle Filter Explained With Python Code." Available: https://www.youtube.com/watch?v=7Z9fEpJOJdc
[6] YouTube, "The Particle Filter: A Full Tutorial," Sep. 15, 2020. Available: https://www.youtube.com/watch?v=BfKEf2s7Y80&t=1s
[7] YouTube, "Particle Filter Explained," May 8, 2021. Available: https://www.youtube.com/watch?v=gP6MRe_IHFo
[8] Robile-AMR, "Demo Mapping." Available: https://robile-amr.readthedocs.io/en/latest/source/Tutorial/Demo%20Mapping.html