ROS AI Crew

Amol Tatkari, Ayushi Arora, Behrouz Ghamkar, Ujjwal Patil

***Abstract*—This report provides a technical overview of the project to develop an autonomous robotic navigation system using the Robile platform and ROS. Key components include A* for global path planning, Potential Fields for local obstacle avoidance, and particle filter-based localization. A frontier-based exploration strategy enables autonomous navigation of unexplored areas. The system was tested in simulations, demonstrating effective real-time navigation, obstacle avoidance, and exploration. Full project details, including the code repository and video demonstrations, are available in the appendix for further review.**

***Index Terms*—Autonomous Systems, Path Planning, A* Algorithm, Potential Fields, Particle Filter, Frontier-based Exploration, Robot Localization, Obstacle Avoidance, ROS, Mobile Robots**

I. Introduction

Autonomous mobile robots play a pivotal role in industries such as logistics, exploration, and surveillance, where the ability to navigate unknown environments without human input is essential. This project aims to design and implement an autonomous navigation system for the Robile platform using the Robot Operating System (ROS). Our approach combines global path planning, local obstacle avoidance, and accurate localization to ensure effective and autonomous robot navigation. The A* algorithm is used for global path finding, ensuring optimal routes, while the Potential Fields method handles local dynamic obstacle avoidance. To maintain accurate localization throughout the robot's movement, a particle filter is employed. Furthermore, a frontier-based exploration strategy is integrated to allow the robot to autonomously explore uncharted areas. The report is divided into three main sections. Section 2 - Path and Motion Planning covers the global planner using the A* algorithm and the local planner using Potential Fields for dynamic obstacle avoidance, Section 3 - Localization describes the implementation of a particle filter to accurately track the robot's position within its environment, and finally Section 4 - Environment Exploration covers the implementation of an exploration algorithm that guides the robot to unexplored regions, using frontier-based exploration and a SLAM component for map generation.

II. Path and Motion Planning

A. Global Planner - A*

The A* algorithm is a fundamental part of the robot's autonomous navigation system, enabling it to find the shortest path from a starting point to a goal in a 2D occupancy grid environment. This grid is a map representation of the robot's surroundings, where each cell is classified as free (traversable), occupied (obstacle) or unexplored. The algorithm computes a collision-free path by navigating through the grid, taking into account both the robot's physical dimensions and the distribution of obstacles.

The path planning task involves calculating a valid sequence of moves from the robot's current position S to a desired goal position G, both of which are defined by grid coordinates. The robot's environment is continuously updated by sensor data, which is used to populate the occupancy grid with information about free spaces and obstacles. The goal of the A* algorithm is to find the most efficient path from S to G while avoiding collisions with any obstacles.

A* works by combining the advantages of Dijkstra's algorithm with a heuristic approach to guide the search toward the goal more quickly. The algorithm uses a cost function $f(n)$, which is a sum of two components: the actual cost $g(n)$ from the start to the current node, and a heuristic estimate $h(n)$, which is the predicted cost from the current node to the goal. In this implementation, the Euclidean distance between the current node and the goal is used as the heuristic:

$$h(n) = \sqrt{(x_{goal} - x_{current})^2 + (y_{goal} - y_{current})^2}$$

The algorithm starts by initializing the starting node with a cost of $g(S) = 0$ and estimating its total cost $f(S)$. This node is placed in a priority queue, and the algorithm proceeds by expanding nodes from the queue, selecting the node with the lowest total cost f. The neighboring nodes of each expanded node are evaluated, and their costs are updated accordingly. If a cheaper path to a neighboring node is found, the algorithm updates the node's cost and continues the search. This process continues until the goal is reached or no more nodes are available.

To ensure collision avoidance, the algorithm includes a mechanism to check that the robot's movement will not bring it too close to obstacles. The `check_threshold` function verifies that all neighboring cells within a certain distance (threshold) are free of obstacles before allowing the robot to move into a new position. This takes into account the robot's size and ensures a safe buffer zone around any obstacles.

Once the algorithm finds a valid path, the sequence of grid cells is converted into a format that can be understood by the robot's navigation system. The `astarpath_to_rospath` function translates the path into a ROS `Path` message, which is then published for the robot to follow. The robot can then

*Submitted to the Department of Computer Science at Hochschule Bonn-Rhein-Sieg in partial fulfilment of the requirements for the degree of Master of Science in Autonomous Systems

†Supervised by Supervisor 1 (Affiliation) and Supervisor 2 (Affiliation)

‡Submitted in Month 20XX

use this path to navigate from its current position to the goal while avoiding obstacles.

A* provides a robust and efficient method for pathfinding in real-time environments. While it computes optimal paths under typical conditions, its performance can be affected by the size and complexity of the environment. In cases where the grid is large or densely packed with obstacles, performance may degrade, but the algorithm remains highly effective for the type of navigation required in this project. Future improvements could include dynamic re-planning or hierarchical pathfinding to further optimize its performance.

In summary, the A* algorithm enables the robot to autonomously navigate through a cluttered environment, ensuring safe and efficient movement from a start position to a goal while avoiding obstacles. Its real-time pathfinding capabilities make it a key component of the robot's exploration and navigation systems.

B. Local Planner - Potential Fields

The obstacle avoidance system for the robot relies on a potential field method, which is designed to guide the robot towards a target while avoiding obstacles. This system works in conjunction with the A* pathfinding algorithm. Once the A* algorithm generates a path, the obstacle avoidance system receives this path as a series of waypoints (*PoseStamped*) and attempts to follow them. The potential field method ensures smooth and safe navigation by dynamically balancing attractive forces pulling the robot towards the goal and repulsive forces pushing it away from obstacles.

1) Attractive and Repulsive Forces: The potential field method relies on two key components: attractive forces and repulsive forces. The attractive force pulls the robot towards the next waypoint, while the repulsive forces steer the robot away from obstacles detected by the *LaserScan*.

- **Attractive Force:** The attractive force is calculated as a vector pointing from the robot's current position to the current waypoint. The magnitude of this force depends on the distance to the waypoint. When the robot is far from the waypoint, the attractive force is stronger, guiding it toward the goal. As the robot approaches the waypoint, the force weakens, allowing for finer adjustments. If the robot is close to the waypoint (within a threshold distance), it is considered to have "reached" the waypoint, and the system moves on to the next one.
  The formula used to compute the attractive force is:
  $$\mathbf{F}_{att} = k_{att} \cdot \frac{\mathbf{P}_{goal} - \mathbf{P}_{robot}}{d_{goal} + \epsilon}$$
  where k_{att} is the attractive coefficient, $\mathbf{P}_{goal}$ is the position of the goal, $\mathbf{P}_{robot}$ is the current position of the robot, and d_{goal} is the Euclidean distance to the goal. A small epsilon (ϵ) is added to avoid division by zero.
- **Repulsive Force:** The repulsive forces are generated based on the proximity of obstacles detected by the robot's *LaserScan*. Obstacles within a certain range (less than 1 meter) exert a repulsive force on the robot, pushing it away. The closer the obstacle, the stronger the repulsive force. Obstacles farther away than the threshold distance are ignored, allowing the robot to focus on nearby hazards.
  The repulsive force for each detected obstacle is computed as:
  $$\mathbf{F}_{rep} = -k_{rep} \cdot \left(\frac{1}{r} - \frac{1}{r_{max}}\right) \cdot \frac{1}{r^2} \cdot \mathbf{d}$$
  where k_{rep} is the repulsive coefficient, r is the distance to the obstacle, r_{max} is the maximum sensor range, and $\mathbf{d}$ is the unit vector pointing from the obstacle to the robot. The total repulsive force is the sum of all individual forces from nearby obstacles.

2) Path Following and Obstacle Avoidance: The system processes waypoints one at a time. For each waypoint, the robot calculates both attractive and repulsive forces. It combines these forces to determine the overall direction of movement. The robot attempts to minimize the angular difference between its current heading and the desired direction (total force vector). If the waypoint is directly in front of the robot, the robot proceeds forward, but if the direction requires turning, the robot first rotates toward the waypoint.

If the repulsive forces from obstacles significantly affect the total force vector, the robot adjusts its movement to avoid collisions. This dynamic interaction between attractive and repulsive forces ensures that the robot remains on track while maintaining a safe distance from obstacles.

The robot's movement is controlled using the following process:

- **Linear Velocity**: The robot moves forward based on the magnitude of the total force and the distance to the waypoint. The closer the robot gets to the waypoint, the slower it moves, preventing overshooting.
- **Angular Velocity**: The robot's angular velocity is adjusted based on the angle between its current heading and the direction of the total force vector. If this angle exceeds a certain threshold (90 degrees), the robot first rotates in place until it is aligned with the target.

3) Handling Dynamic Obstacles: While the system is primarily designed for static environments, the *LaserScan* data ensures that it can handle dynamic obstacles as well. As the robot navigates through the environment, any newly detected obstacles immediately generate repulsive forces, allowing the robot to respond in real-time. This makes the obstacle avoidance system reactive and adaptive to changes in the environment.

III. Particle Filter

The particle filter is a variant of Bayes filter based on importance sampling and application of particle filter to mobile robot localization is known as Monte Carlo Localization [1]. Using particle filter we estimate the robot pose (x, y, θ) given map and sensor measurements. In particle filter, we initially generate n particles which are our hypothesis about where robot could be in the map, then each particle is moved by robot

odometry. The each particle assigned with the importance weight and then in the final step particles are resampled using their importance weight. We will explain our implementation details which are divided into 4 sub-sections.

A. Implementation

1) Particle Generation: In our implementation we generate particles/hypotheses at and around high importance weight particles after resampling. The particle is represented as a tuple consisting of pose(i.e. state hypothesis) and importance weight. The set of N particles X is represented as below,

$$X = [(x_i, y_i, \theta_i, W_i)] where, i = 1, 2, 3, ...N$$

At start, we generate 9 particles uniformly around initial pose estimate. We assign weight of 1/9 to each particles implying each particle has same importance at the start. We selected number of particles as 9 in order to optimize performance and accuracy of localisation.

We also generate particles after resampling around 3 high important particles by sampling from a Gaussian distribution. We generate 3 particles around each high importance particles keeping total number of particles same.
2) Motion Update: : To predict the movement of particles as robot moves, we are using odometry model given in [1]. We are not moving the particles continuously as robot moves. Insetad, we have considered a threshold of 0.3 m in distance and 0.3 radians in angular rotation. After aevery 0.5 seconds we check whether robots moved or rotated more than corresponding theshold, particles positions are updated using the odometry update. In order to consider uncertainity, we have considered a gaussian noise in translation and rotation.
3) Measurement Update: In [1], authors discussed different sensor/observation models like, beam-based models, likelihood fields, feature-based models etc. But most of these models requires pre-estimation of model parameters. Hence, in our implementation we try match how close the laser scan of each particle matches with the actual laser elements.

The laser scan on the robile gives 150 range readings and matching these 150 points with each particle scan would be computationally expensive and affect real-time performance. To overcome
4) Resampling: In our algorithm we use a stochastic universal sampling algorithm given in [1] to sample particles as per their weight.

B. Objective

The objective of this task is to implement the particle filter and integrate it with the robile

C. Particle Filter

D. Results and Evaluation

Write about Particle Filter here.

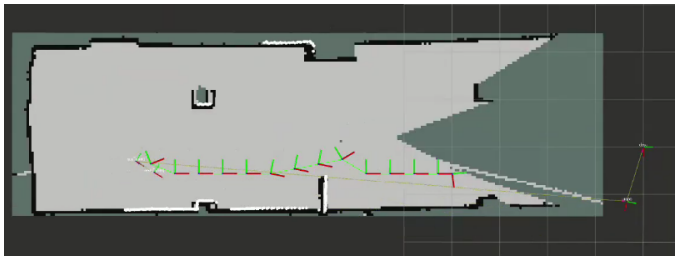

Fig. 1: Path generated by the A* planner to the selected frontier goal, showing the robot's trajectory through the environment

IV. Exploration

A. Exploration Strategy

The exploration strategy in this project is based on frontier-based exploration, where the robot focuses on navigating the boundary between explored and unexplored areas, known as frontiers. A frontier is identified as free space adjacent to unknown regions, where further exploration can yield new information about the environment.

The OccupancyGrid provided by the SLAM system is key to this strategy. In this grid:

- 0: Free space (explored)
- 1: Occupied (obstacle)
- -1: Unknown space (unexplored)

Frontiers are detected by scanning the map for cells marked as free space (0) that have at least one neighboring cell marked as unknown (-1). These frontier cells are then targeted as exploration goals, guiding the robot towards new areas while avoiding obstacles.

To navigate these frontiers effectively, the exploration algorithm is integrated with the A* global path planner and the potential field (PF) planner for local navigation:

- **A*** computes a global path, finding the optimal trajectory from the robot's current location to the frontier goal, while avoiding large obstacles marked as 100.
- The **PF planner** is used for local obstacle avoidance, ensuring smooth navigation between waypoints generated by A*. It continuously adjusts the robot's movements in real-time to avoid nearby obstacles while following the planned path.

If an unreachable goal is detected by the A* planner (e.g., blocked by obstacles or in an inaccessible region), the system automatically generates a new frontier goal, ensuring continuous exploration without manual intervention.

B. Exploration Component Implementation

The exploration component processes the real-time occupancy grid map from the SLAM system to detect frontiers and generate exploration goals.

- **Occupancy Grid Map Subscription:** The robot subscribes to OccupancyGrid data from the `/map` topic,

continuously updated by the SLAM system (Section 3). The data is converted into a 2D array, classifying cells as free space, obstacles, or unexplored areas. This grid-based format is essential for frontier detection and exploration.

- **Frontier Detection:** Frontier detection scans the grid for free space cells (value 0) with unexplored neighbors (value -1). These cells are marked as frontiers, guiding the robot to unexplored areas efficiently while avoiding redundant exploration.
- **Goal Generation:** When a frontier is found, it is selected as the next goal, converted to real-world coordinates, and published as a `PoseStamped` message with random yaw. This ensures the robot explores new areas with varied paths.
- **Invalid Goal Handling:** If a goal is unreachable due to obstacles, the A* planner identifies it as invalid, and a new frontier is selected. This ensures smooth exploration even in dynamic environments.

C. Results and Evaluation

The exploration algorithm was tested in simulated environments using the Robile platform. The robot successfully generated goals based on frontier detection and was able to navigate toward these goals using a combination of the potential field planner and global path planner. The key observations include:

- **Efficient Frontier Detection:** The robot accurately identified frontiers by detecting free space (0) adjacent to unknown areas (-1), allowing for systematic exploration of the environment. This ensured that unexplored regions were consistently targeted.
- **Accurate Path Planning:** The A* planner effectively computed global paths to frontier goals, avoiding obstacles (100) while guiding the robot to unexplored areas. The PF planner ensured smooth local navigation, avoiding collisions and ensuring real-time obstacle avoidance.
- **Dynamic Goal Handling:** If a goal was unreachable due to obstacles, the system quickly generated new frontier goals, ensuring uninterrupted exploration and minimizing downtime.

Overall, the integration of frontier-based exploration, A*, and the PF planner proved to be a highly effective strategy, enabling the robot to autonomously explore unknown environments in a structured and efficient manner.

TEAM MEMBERS CONTRIBUTION

- Amol Tatkari : Particle Filter implementation, Debugging, Testing and Technical Report
- Ayushi Arora : Exploration implementation, Debugging, Testing and Technical Report
- Behrouz Ghamkhar : A* and Pf planner implementation, Debugging, Testing, Repository management and Technical Report
- Ujjwal Patil : A* and Pf planner implementation, Debugging, Testing, Robot Communication and Technical Report

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

APPENDIX

A. Code Repository

The complete code for the project, including all source files and configuration, is available in the following Git repository:

- https://github.com/HBRS-AMR/amr-project-ros_ai_crew

B. Recorded Videos

The recorded videos showcasing the project implementation and the robot's exploration capabilities are available at these links:

- Video 1
- Video 2
- Video 3