Planning and Control for Mobile Robot Navigation

Brian Lim Tze Zhen
Dept. of Mechatronic and Cyber-Physical Systems
Deggendorf Institute of Technology
Cham, Germany
brian.lim-tze-zhen@stud.th-deg.de

***Abstract*—This paper presents the development of an autonomous navigation system for a differential-drive mobile robot using the Robot Operating System (ROS) Noetic and Gazebo simulator. The system integrates global path planning via the A* algorithm, local path tracking through a unicycle kinematic controller, and real-time obstacle avoidance using a potential field method. Each module is implemented as a dedicated Python-based ROS node, allowing modularity and clear separation of concerns. The robot's environment is represented as an occupancy grid map generated by GMapping SLAM, while RViz is used for visualization and goal definition. Simulation results confirm that the robot can effectively navigate to a user-defined goal while responding to static obstacles in real time. Performance is evaluated through trajectory plots, time-to-goal metrics, and average speed, demonstrating both system effectiveness and practical control tradeoffs.**

***Index Terms*—Mobile robot navigation, A* algorithm, Potential field method, Kinematic controller, ROS Noetic, Gazebo simulation**

I. Introduction

Autonomous navigation is one of the crucial parts for mobile robots to operate in structured environments. This project focuses on the implementation and simulation of a navigation system for a differential-drive mobile robot (Turtlebot3) using the Robot Operating System (ROS) NOetic and Gazebo. The purpose is to enable the robot to reach a goal while avoiding obstacles autonomously.

The proposed system architecture includes three main functionalities: global path planning, local planning, and obstacle avoidance. Global planning was implemented by using the A* algorithm [1] applied to an occupancy grid map, which results in generating a collision-free path from the robot's current location to a user-defined goal by interacting with Rviz. Local path tracking is handled by a kinematic controller based on the unicycle model, which continuously computes velocity commands to follow the waypoints published by the global planner. For obstacle avoidance, a potential field method [2] was implemented using the LIDAR data to apply repulsive forces when objects were nearby and attracted by attractive forces.

All navigation components are implemented as modular Python scripts and deployed as individual ROS nodes. The complete system is validated in simulation using Gazebo, with RViz used for visualization and user interaction. This report presents the design methodology, implementation details, and simulation outcomes, and discusses the system's strengths, limitations, and areas for future improvement.

II. System Architecture

The navigation system is organized into four modular ROS nodes, each handling a distinct part of the robot's behavior. This separation ensures clarity, flexibility, and ease of maintenance. The nodes are as follows:

global_planner.py: This node produced global path planning using the A* search algorithm. It subscribed to `/map`, `/odom`, and `/move_base_simple/goal`. Once a user-defined goal was received through RViz, it computed a collision-free path on the occupancy grid and published it to `/planned_path`.

navigator.py: This node worked as the local waypoint tracker. It followed the global path by applying a unicycle kinematic controller. It also detected nearby obstacles using LIDAR data from `/scan`. When an obstacle was too close, it triggered an override from the kinematic controller to the potential field planner via a ROS service call.

potential_fields.py: This node implemented the potential field method for obstacle avoidance. It produced repulsive forces from nearby obstacles and attractive forces to pull toward the goal. The resulting control command was published to `/cmd_vel`. It temporarily took over the kinematic controller when triggered.

kinematic_controller.py: This node represented the unicycle model-based velocity control. The current pose and the next waypoint were converted into suitable linear and angular velocity commands, which were used by `navigator.py` for local path following.

Moreover, each node communicated through standard ROS topics and services. Finally, a real-time visualization was performed in RViz, and the simulation was conducted using Gazebo.

III. Methodology

This section describes the implementation strategy for each core component of the navigation system.

A. Global Planning with A* Algorithm

The `global_planner.py` node implements the A* algorithm on a 2D occupancy grid map obtained via *GMapping* SLAM [3]. The robot subscribes to `/map` for the occupancy grid, `/odom` for the current position, and `/move_base_simple/goal` for user-defined goals in *RViz*.

Obstacle inflation [4]is applied to the occupancy grid using the `grey_dilation` function from the `scipy.ndimage` library, creating a safety buffer around obstacles. The inflated map is then used by the planner to generate a safe path, which is published to `/planned_path` as a `nav_msgs/Path` message.

B. Local Tracking with Kinematic Controller

The `kinematic_controller.py` script computes velocity commands based on a unicycle kinematic model. Given the robot's pose and the next waypoint, the controller generates linear and angular velocities to guide the robot along the global path.

Waypoint pruning is used to remove already-passed waypoints, ensuring smooth and efficient tracking. This controller publishes velocity commands to `/cmd_vel` unless overridden by obstacle avoidance logic.

C. Obstacle Avoidance using Potential Fields

The `potential_fields.py` node uses LIDAR data from `/scan` to compute repulsive forces from nearby obstacles and attractive forces toward the goal. The combined force vector is translated into a velocity command that steers the robot safely while maintaining progress toward the target.

A threshold distance determines whether the potential field planner should take control, which is coordinated through a ROS service call.

D. Reactive Control Switching with Navigator Node

The `navigator.py` node acts as a mediator between the controller and the potential field planner. It monitors obstacles using LIDAR and triggers the potential field override when obstacles are within 0.4 meters. A ROS `Trigger` service is called to request repulsive-action commands.

The navigator ensures that during active avoidance, velocity commands from the kinematic controller are suppressed, maintaining modular responsibility across components.

IV. Simulation Setup

The navigation system was developed on **ROS Noetic** with **Ubuntu 20.04**, and validated in the **Gazebo** simulator. Real-time visualization and control were handled using **RViz**.

The robot platform was a **TurtleBot3 Burger**, modeled as a differential-drive mobile robot equipped with a 2D LIDAR. The testing environment was a custom world defined in `simulation.world`, designed to represent a structured indoor layout with static obstacles and corridors.

To generate the static map used for global planning, the **GMapping** SLAM algorithm was run once during an initial exploration phase. The resulting occupancy grid was saved using `map_server` and reused for subsequent simulations. This allowed the navigation system to operate on a known environment without rerunning SLAM every time.

The full navigation workflow proceeds as follows:

1) `simulation.launch` – Launches the robot in Gazebo with the preloaded map.
2) `global_planner.py` – Subscribes to the saved map, applies obstacle inflation, and computes a global path using A*.
3) `navigator.py` – Handles waypoint following and monitors LIDAR for nearby obstacles.
4) `kinematic_controller.py` – Computes velocity commands based on the unicycle model and current waypoint.
5) `potential_field.py` – Activated when obstacles are detected within a safety threshold. Computes repulsive and attractive forces and overrides motion commands.
6) RViz – Used to send goal positions and visualize paths, sensor data, and navigation behavior.

This setup enables the robot to reach user-defined goals while dynamically avoiding obstacles, all within a realistic simulation environment.

V. Results

A. Trajectory Plots

Two test scenarios were conducted to evaluate the performance of the navigation system: one in an open environment without obstacles and another in a structured environment with static obstacles.

Figure 1 shows the robot's trajectory in a clear, obstacle-free environment. The green line represents the global path planned by the A* algorithm. The robot follows this path precisely using the kinematic controller, without needing any local adjustments. This test demonstrates the system's accurate waypoint tracking under ideal conditions.

Figure 2 illustrates the trajectory in an environment containing multiple obstacles. Despite the complexity, the robot successfully reaches the goal by leveraging potential field-based obstacle avoidance. The global path remains intact, but local deviations can be observed as the robot responds to nearby obstacles in real-time. These experiments confirm that the robot can navigate both in free space and in cluttered environments while maintaining goal-directed behavior and collision avoidance.

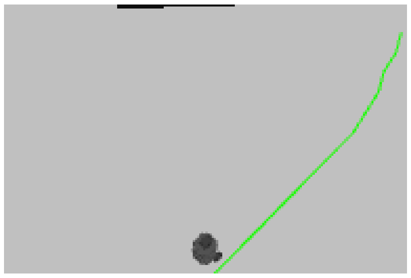
Fig. 1. Trajectory without obstacles.

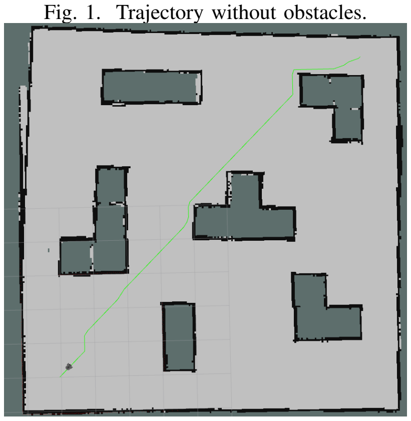
Fig. 2. Trajectory with obstacles avoidance

B. RViz and Gazebo Screenshots

Figures 3 and 4 illustrate the simulation setup and the real-time navigation behavior of the robot. The Gazebo environment provides a structured test scenario, while the RViz visualization demonstrates the planned path and the interaction of attractive and repulsive forces during execution.

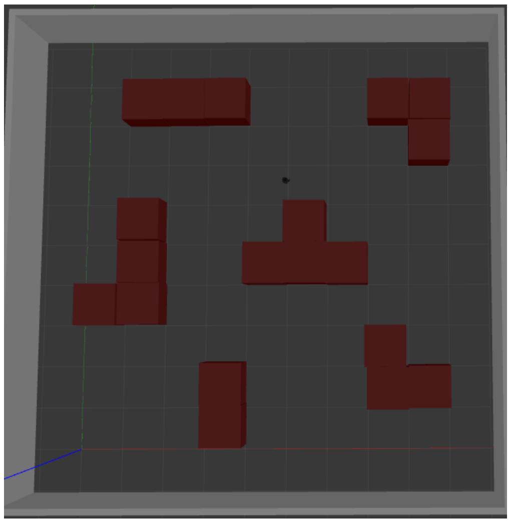
Fig. 3. Gazebo simulation: TurtleBot3 navigating the world

C. Performance Metrics

The robot successfully reached the goal using the implemented kinematic controller. Key performance indicators were measured from the moment the global planner received the goal until the robot arrived at the final waypoint. These are summarized below:

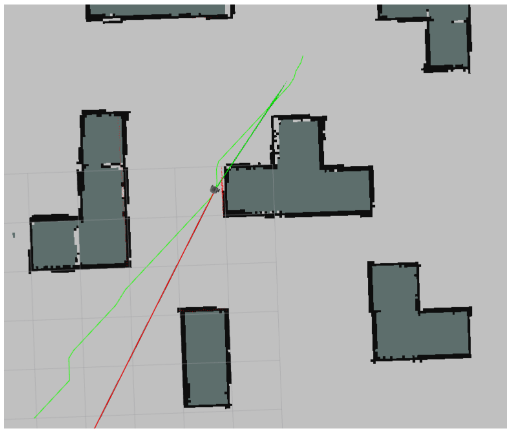
Fig. 4. RViz visualization: global path, local path, LIDAR rays

- **Time to Goal:** 103.53 seconds
- **Path Length:** 8.91 meters
- **Average Speed:** 0.09 m/s

The average speed was computed as the total Euclidean path length divided by the elapsed time to goal completion.

VI. Summary of Experimental Observations and Tuning

During experimentation, several observations were made regarding the robot's behavior and controller performance, which informed subsequent parameter tuning and system refinements.

- **Trajectory Smoothness:** The planned path generated using the A* algorithm was initially jagged due to high waypoint density. Downsampling the path by selecting every third point (step = 3) helped smooth the trajectory without sacrificing path accuracy.
- **Controller Gains:** The unicycle model-based kinematic controller required careful gain tuning. The final gains used were:
  - $\kappa_\rho = 0.5$ (linear distance gain)
  - $\kappa_\alpha = 1.0$ (heading correction gain)
  - $\kappa_\beta = -0.3$ (final orientation correction)

  These values provided a good balance between stability and responsiveness, minimizing overshoot while maintaining progress along the path.
- **Waypoint Tolerance:** A threshold $\epsilon = 0.1$ m was used to determine proximity to waypoints. A smaller threshold led to oscillation or stalling at waypoints, while a larger value caused premature transitions and reduced tracking precision.
- **Obstacle Avoidance:** In scenarios involving close obstacles, the robot occasionally deviated from the planned path. This was mitigated by integrating a potential field module that locally overrides the path-following controller using repulsive forces based on LIDAR data.

- **Timing Metrics:** The system recorded both path length and time-to-goal to evaluate controller efficiency. The average speed metric provided additional insight into tuning effectiveness and real-world traversal speed.
- **Stability Improvements:** Motion instability near sharp turns was reduced by limiting maximum angular velocity through gain tuning.

Overall, empirical tuning based on observed motion behavior proved essential to achieving reliable navigation and smooth motion across varied environments.

VII. Conclusion

This project successfully demonstrated a modular and efficient navigation system for a differential-drive mobile robot using ROS Noetic and Gazebo. By integrating a global planner based on the A* algorithm, a unicycle model-based kinematic controller, and a potential field method for real-time obstacle avoidance, the robot was able to reach user-defined goals while navigating through dynamic environments.

Key achievements include:

- Real-time path planning and execution within a simulated world.
- Robust obstacle avoidance through potential field override.
- Tuned control parameters for smooth and stable motion.
- Quantitative performance evaluation (e.g., time to goal, average speed).

VIII. Code Availability

The complete source code, including controller implementations, launch files, and demo materials, is available at:

https://github.com/Brian-Lim-Tze-Zhen/
Autonomous-System-Planning-and-Control-for-Mobile-Robot-Navigation

References

[1] GeeksforGeeks, “A* search algorithm - data structures and algorithms,” https://www.geeksforgeeks.org/dsa/a-search-algorithm/, 2025, accessed: 2025-07-07.

[2] H. Choset, “Chapter 4: Potential field methods,” https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf, 2025, lecture Notes, CMU 16-899: Motion Planning, Accessed: 2025-07-07.

[3] ROBOTIS, “Turtlebot3 simulation using slam,” https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/, 2025, accessed: 2025-07-07.

[4] Open Source Robotics Foundation, “costmap_2d — ros wiki,” http://wiki.ros.org/costmap_2d, 2025, accessed: 2025-07-07.