Enhancing Autonomous Exploration and Navigation in Unknown Environments: A Software Architecture for Turtlebot Robot

*

Preeti Verma
Universitat De Girona
Girona, Spain

Moses Ebere
Universitat De Girona
Girona, Spain

Joseph Adeola
Universitat De Girona
Girona, Spain

***Abstract*—This research presents a software architecture designed to enable autonomous exploration and navigation for the Turtlebot robot in uncharted environments. The architecture integrates cutting-edge technologies, including a frontier-based exploration algorithm combining Multiple Rapidly-exploring Random Trees (MRRT) and Mean Shift clustering. With a high-resolution 2D Lidar, the robot achieves precise obstacle detection and mapping. The path planning module utilizes the Open Motion Planning Library (OMPL) and a customized Dubins state space, generating collision-free paths that navigate dynamic constraints. Optimization techniques, such as a comprehensive cost function and post-processing methodologies, enhance path quality. The low-level controller, employing the Dynamic Window Approach, ensures smooth navigation and real-time obstacle avoidance. The software architecture is versatile, accommodating real-world and simulated scenarios for comprehensive evaluation. Experiments were conducted in both real-world and simulated scenarios to assess its performance. The results demonstrate improved exploration capabilities and collision-free path planning in dynamic environments. By addressing the challenges of unknown environments, this research contributes to the field of autonomous robotics, offering insights into the practical implementation of autonomous exploration and navigation systems.**

***Index Terms*—autonomous, robotics, exploration, path-planning**

I. Introduction

Robotic exploration and navigation in unknown environments pose significant challenges for autonomous robots. Robots must be capable of mapping their surroundings, planning obstacle-free paths, and navigating smoothly to achieve effective autonomous operation. This research project focuses on designing a software architecture for a Turtlebot robot that enables autonomous exploration and navigation.

The software architecture addresses vital areas, including the exploration algorithm, sensors utilized, path planning algorithm, and low-level controller. The chosen exploration algorithm is based on the Frontier-based exploration strategy, which involves identifying frontiers as the boundary regions between free and unexplored space. The Multiple Rapidly-exploring Random Trees (MRRT) algorithm is employed to efficiently discover and explore the frontiers [1]. The software architecture incorporates a 2D Lidar sensor on the Turtlebot robot, providing range measurements for obstacle detection and accurate mapping, enhancing exploration and navigation.

Effective path planning plays a crucial role in autonomous navigation. The software architecture leverages the Open Motion Planning Library (OMPL) to generate collision-free paths. Paths are planned in a Dubin's state space, considering the robot's motion capabilities and constraints. A comprehensive cost function, incorporating path length and obstacle clearance, defines the planning objective. Post-processing techniques, such as path smoothing methodologies, are also investigated to improve constraint-friendliness.

The low-level controller plays a crucial role in executing planned paths and facilitating smooth navigation while accommodating the dynamic constraints of the robot. The Dynamic Window Approach is adopted as the primary local planner, guiding the robot toward points of interest while effectively avoiding obstacles. A robust cost function is employed to enhance the low-level controller's decision-making process, considering key parameters such as heading, clearance, and velocity. A simple obstacle avoidance technique is implemented to provide redundancy during navigation as an additional safety measure.

Furthermore, the software architecture is designed to be versatile, applicable in both real-world scenarios and the Stonefish simulator. This flexibility allows for comprehensive testing and evaluation, ensuring the system's performance and robustness are thoroughly assessed. By addressing the challenges associated with autonomous exploration and navigation, this research project aims to contribute to advancing robotic systems in unfamiliar environments.

II. Related Work

Several research works have explored similar strategies for autonomous exploration and navigation in unknown environments, focusing on the integration of exploration algorithms, sensors, path-planning algorithms, and low-level controllers.

This section provides a concise overview of recent publications that align with the objectives of our project.

Liu et al. [2] presented by a study on efficient autonomous exploration and mapping methods using an incremental caching topology-grid hybrid map (TGHM). The TGHM combines a topology map for exploration information and a grid map for navigation and localization. Candidate target points are rapidly generated using geometry rules. The TGHM is then established, and the information gained is evaluated for each candidate topology node. The highest evaluated node becomes the next target point, and the topology map is updated after each motion. Simulations and experiments validate the algorithm's performance in robot autonomous exploration and map construction.

Gao et al. [3] proposed a new approach for autonomous exploration in unknown scenarios based on frontiers. While previous frontier strategies focus on distance and size of unknown spaces, this approach incorporates robot heading information and coarse graph representation to improve exploration efficiency. Experimental validation was performed in a cluttered office environment, demonstrating superior performance in terms of coverage and efficiency. The results highlight the potential of this modified exploration method for addressing challenges in complex and unknown environments.

In a recent study by Johnson et al. [8], a Frontier-based exploration strategy was employed to enable an autonomous drone to explore unknown terrains. The authors utilized the Rapidly-exploring Random Trees (RRT) algorithm to identify frontiers and plan exploration paths. By leveraging onboard sensors, including LiDAR and RGB-D cameras, the drone successfully built maps of the environment and navigated through complex terrain. Another relevant work by Chen et al. [9] focused on autonomous exploration in underground mines using a mobile robot. The authors adopted a Frontier-based approach combined with the Adaptive Monte Carlo Localization (AMCL) algorithm to enable the robot to explore and map the environment efficiently. The integration of multi-modal sensors, such as 2D and 3D LiDAR, facilitated accurate perception and obstacle detection, leading to robust exploration and navigation capabilities.

In the field of path planning, Smith et al. [10] proposed a novel algorithm based on the Rapidly-exploring Random Trees Connect (RRT-Connect) approach for generating collision-free paths in dynamic environments. The algorithm dynamically adjusted the planned path based on real-time sensor inputs to avoid moving obstacles. The authors demonstrated the effectiveness of their approach through simulations and real-world experiments using a ground mobile robot.

Gao et al. [11] presented a path-planning algorithm that considered dynamic constraints and energy efficiency for an autonomous underwater vehicle (AUV). By incorporating the Dubins path planning technique and optimizing the path based on energy consumption, the AUV efficiently explored and navigated underwater environments while conserving energy. In terms of low-level control, Li et al. [12] proposed a robust local planner based on the Dynamic Window Approach for the autonomous navigation of ground robots in cluttered environments. The authors integrated multi-sensor fusion, including LiDAR and RGB-D cameras, to enhance obstacle detection and avoidance capabilities. The proposed local planner effectively steered the robot, ensuring smooth and collision-free navigation.

III. Methodology

A. Automatic Exploration Strategy

Exploration aims to expand the map by directing robots toward unexplored areas using frontier-based strategies. Frontier edges, which separate known and unknown space in the map, guide the robot's exploration. When a frontier edge is found, the robot is assigned a point, usually the centroid, for exploration. However, processing the entire map for frontier edge extraction can be computationally demanding, particularly as the map size increases. Efficient methods are researched to detect frontier edges and optimize computational resources. This project employs multiple Rapidly-exploring Random Trees (RRTs) as the exploration strategy due to their inclination towards unexplored areas, making them well-suited for the task. The adaptability of RRTs to higher dimensional spaces ensures versatility for future applications.

The exploration strategy consists of three modules: the RRT-based frontier detector, the filter, and the robot task allocator. The frontier detector identifies frontier points and transfers them to the filter module. In this project, the mean shift clustering algorithm is utilized within the filter module to cluster and store the frontier points. Additionally, the filter module eliminates invalid and outdated frontier points. The task allocator module receives the clustered frontier points from the filter module and assigns them to a robot for exploration. The schematic of exploration strategy is shown in Fig. 3.

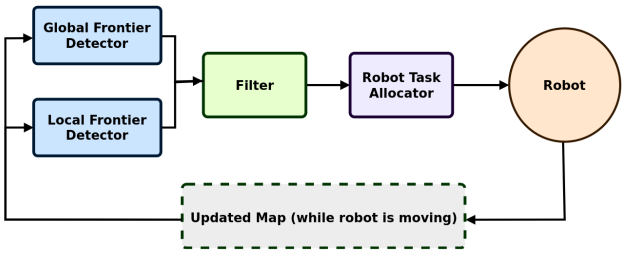

Fig. 1. Schematic of Exploration strategy

a) RRT-based frontier detector: The RRT-based frontier detector module plays a crucial role in identifying frontier points during the exploration process. A point is considered a frontier point if the growing RRT tree reaches it and resides in the uncharted region of the map. The map is represented as an occupancy grid, where points in the unknown region are assigned a cell value of -1. Each point can be classified as unknown, free, or occupied by examining the cell value, enabling effective frontier detection. Notably, our approach

initializes the occupancy grid map with solely unknown cells, gradually updated as the robot explores and marks obstacles and known areas. To enhance exploration efficiency, the two distinct types of frontier detectors are utilized: a local frontier detector and a global frontier detector.

Algorithm 1 Local Frontier Detector

1: **Initialization**: $V \leftarrow \{x_{\text{init}}\}$ $E \leftarrow \varnothing$
2: **while** True **do**
3: $x_{\text{rand}} \leftarrow$ SampleFree
4: $x_{\text{nearest}} \leftarrow \text{Nearest}(G(V,E), x_{\text{rand}})$
5: $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}}, \eta)$
6: **if** FrontierCell$(grid_map, x_{\text{new}})$ **then**
7: PublishPoint(x_{new})
8: **reset the tree**
9: $V \leftarrow \{x_{\text{current}}\}$; $E \leftarrow \varnothing$
10: **else**
11: $V \leftarrow V \cup \{x_{\text{new}}\}$
12: $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$
13: **end if**
14: **end while**=0

Algorithm 1 presents the local frontier detector, which resembles the RRT algorithm. It commences with a single initial vertex, x_{init}, and an empty edge set ($E = \varnothing$). A random point, x_{rand}, is sampled in each iteration, and the nearest vertex, $x_{nearest}$, is determined. A new point is generated, and its status as a frontier point is assessed using the FrontierCell function, considering its eight-connectivity neighbors. Upon confirmation, the point undergoes filtering, and the tree is reset, deviating from conventional RRT implementations. This process recurs for each robot motion, with the tree originating from a fresh initial point corresponding to the robot's current position, $x_{current}$, after each reset. The local detector empowers rapid detection of neighboring frontier points amidst robot motion.

The global frontier detector follows the same structure as the local frontier detector (Algorithm 1), with the exception of removing the reset tree line (line 9). Unlike the local detector, the tree in the global detector continues to grow throughout the entire exploration process, similar to the behavior of RRT. The global frontier detector is designed to identify frontier points across the entire map, including regions distant from the robot. The expansion of global tree nodes is exhibited in Fig. 2.

b) Filter: The filter module utilizes the mean-shift algorithm for clustering the frontier points received from both the local and global frontier detectors. This algorithm helps identify the centers of the clusters, while discarding the remaining points. Clustering is essential to reduce the number of frontier points, especially when the detectors generate multiple points in close proximity. By applying mean-shift clustering, computational resources are optimized, and only relevant map information is retained. Furthermore, the filter module also eliminates invalid and outdated frontier points during each iteration.

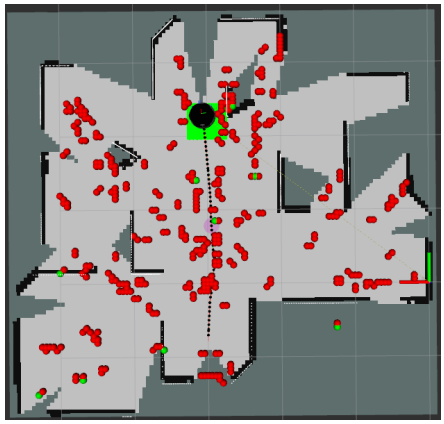

Fig. 2. Expansion on global tree nodes, indicated by red spheres, in environment while exploration

c) Robot Task Allocator: The robot task allocator module receives filtered frontier points from the filter module and assigns them to individual robots. The design of this module is inspired by [21]. Assigning frontier points is based on the following considerations:

- Navigation cost (N): It measures the expected distance a robot needs to travel to reach a frontier point. The navigation cost is computed by taking the norm of the difference between the robot's current position and the location of the frontier point.
- Information gain (I): It quantifies the area of the unknown region that is expected to be explored by a specific frontier point. The information gain is determined by counting the number of unknown cells within a sensor-defined radius, referred to as the information gain radius. The subtraction of obstacle cells serves to penalize the presence of obstacles, reducing the information gain in areas with more obstacles. The area is calculated by multiplying the number of cells within the radius by the area of each cell, based on the map resolution. The equation of information gain is as follows:

$$I = (\text{unknown\_cells\_count} - 1.5 \times \text{obstacle\_cells\_count}) \times \text{cell\_area} \tag{1}$$

- Revenue from a frontier point (R): The revenue reflects the value or desirability of exploring a particular frontier point. It is determined by considering both the navigation cost and the information gained at each frontier point. It takes the navigation cost and information gain values of each frontier point as inputs. The information gain and navigation cost values are normalized to ensure a fair comparison. The revenue for each frontier point is computed by employing following equation:

$$\text{revenue} = \frac{\text{information\_gain\_norm}}{\text{navigation\_cost\_norm}} \tag{2}$$

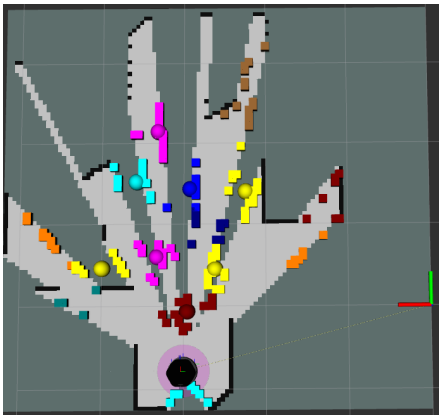

Fig. 3. Clustering of frontiers using Mean-Shift algorithm

B. Path Planner

The planner component of our research focuses on generating collision-free paths from a given start position to a goal position within the defined domain. We employ the Open Motion Planning Library (OMPL) to implement our planner and ensure efficient and effective path planning. The following subsections detail the critical aspects of our planner implementation.

a) State Validity Checker: To ensure the validity of positions or paths concerning the environment, we utilize the StateValidityChecker class. This class examines the validity of a given position or path by utilizing an occupancy map, a 2D array that characterizes the environment's occupancy. The map's resolution and origin are specified. The StateValidityChecker class determines collision-freeness by converting world positions to map coordinates and evaluating the occupancy in the surrounding area. Moreover, the check_path() method discretizes a path and verifies the validity of each position along the path.

b) Path Planning: The path planning process utilizes the OMPL library and the state validity checker to generate collision-free paths. We define an SE2 state space, representing the robot's configuration, and set its bounds according to the specified domain. To optimize the path, we employ the PathLengthOptimizationObjective and set it as the objective for minimizing path length. The RRT* planner (og.RRTstar) is used as our optimizing planner, considering a range parameter that limits the length of added motions during planning. A goal bias is introduced to allow probabilistic selection of the actual goal state. Once the planner is set up with the problem definition, we attempt to solve the planning problem within a specified maximum time. If a solution is found, we retrieve the solution path and simplify it using the PathSimplifier class from OMPL. The resulting path is a list of poses ([x, y]) representing the collision-free trajectory from the start position to the goal position.

c) Optimization Objectives: In this project, we delve into multiple optimization objectives intending to enhance the quality of paths and achieve specific goals. Our primary focus lies in maximizing the clearance of paths from obstacles. To address this objective, we introduce the ClearanceObjective class, wherein the cost assigned to each state is inversely related to its clearance value. By adopting this approach, we ensure that states with higher clearance possess lower associated costs. Furthermore, we explore a balanced objective encompassing path length and clearance optimization. This balanced objective allows us to strike a harmonious trade-off between minimizing path length and effectively avoiding obstacles.

C. Low Level Controller

Considering dynamic constraints, the controller module incorporates proportional control and a dynamic window approach to facilitate the robot's movement between map points. It determines the appropriate linear and angular velocities for the robot to achieve the goal position.

The controller implementation follows a specific logic that can be outlined as follows:

- Compute the dynamic window of permissible velocities based on the robot's current state and maximum acceleration limits.
- Predict the robot's trajectory within a specified time frame, considering the current state, linear and angular velocities, and configuration parameters.
- Evaluate the predicted trajectory's cost using various components such as heading, velocity, clearance, and distance to the goal.
- Select the trajectory with the minimum cost from the dynamic window as the optimal local plan.
- Apply the linear and angular velocities obtained from the best local plan as control inputs to guide the robot.
- Repeat the process continuously, updating the control inputs to steer the robot toward the goal position.

IV. Implementation

The implemented project adopts a ROS (Robot Operating System) package structure, consisting of five nodes: exploration, controller, planner, integration, and Simultaneous Localization and Mapping (SLAM) node. These ROS nodes are implemented using the Python programming language. The SLAM node employed in this project, building upon the hands-on-localization project, serves the purpose of map generation and robot localization. It utilizes pose-based SLAM, leveraging odometry, LiDAR, and Inertial Measurement Unit (IMU) sensor measurements to directly estimate the robot's pose. Based on this pose estimation, a map is constructed.

Within the software architecture, the exploration node plays a crucial role by assigning a specific goal to the robot. This goal is determined by calculating the centroid of the best view frontiers. Once the goal is assigned, the planner node comes into action, employing the RRT* algorithm to generate a path that leads the robot towards the designated goal. Finally, the controller node takes control of the robot, ensuring smooth navigation and guiding it towards the exploration goals.

V. Simulation and Experimental Setup

A. Simulation

The simulation involved conducting simulations using the Stonefish simulator, which offers realistic robotic movements, a physics engine, and the generation of sensor data with added noise.

a) Simulation Environment: Two distinct environments were utilized for the simulations. The first environment is illustrated in Fig. 4, consisted of a rectangular space with an 8.0-meter by 8.0-meter ground plane. This space was enclosed by four walls, forming a closed rectangular boundary.

The second environment is exhibited in Fig. 5, featured an open map design with fewer objects and a structured layout. While maintaining a spacious environment, it offered a different level of complexity compared to the first environment.

In both the simulation environment, the range of the Lidar sensor is from a minimum distance of 0.2 m to a maximum distance of 12.0 m. These chosen environments enabled the evaluation of robotic behaviors and performance in varied settings, allowing for comprehensive exploration and analysis of navigation, obstacle avoidance, and object manipulation capabilities.

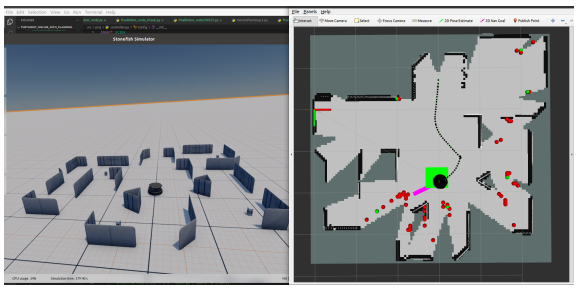

Fig. 4. First Simulation Environment

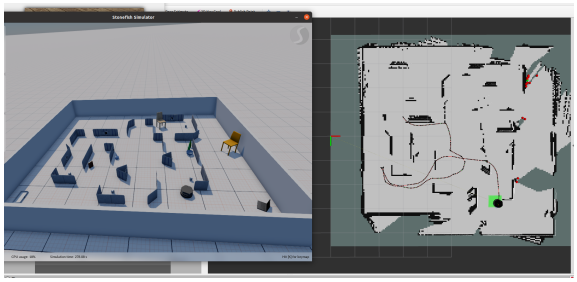

Fig. 5. Second Simulation Environment

B. ExperimentaL Setup

The mobile robot platform utilized in the experimental setup is the Kobuki base. The Slamtech RPLIDAR A2 lidar sensor [25] is employed, which offers a coverage area of 360 degrees and can measure distances up to 6 meters to perceive its environment. Data acquisition is facilitated by using a Raspberry Pi, a single-board computer. The physical environment in which the real experiments take place is depicted in Fig. 6. Furthermore, Fig. 7 and Fig. 8 illustrates the Occupancy grid obtained while exploring the real map. The grid map images are not sharp because of inherent drift obtained from the slam node.

Fig. 6. Real-world Environment Setup Map

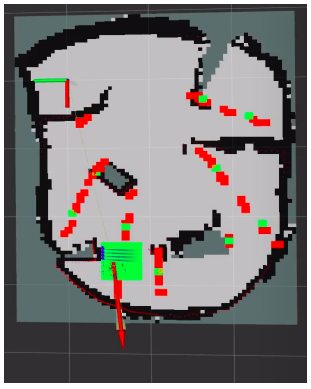

Fig. 7. Occupany grid map while exploring

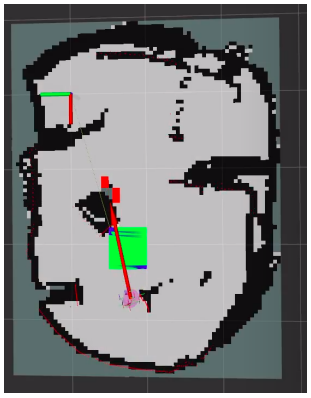

Fig. 8. Occupancy grid map when exploration is almost complete

VI. Result

During the exploration process, frontier detection relies on the adoption of the Rapidly Exploring Random Trees (RRT) approach. The implementation of the frontier detector involves utlizing specific variables, which are as follows: a maximum iteration value of 1500, a delta value of 1, and a goal bias of 0.1. The proximity threshold, i.e., bandwidth, for clustering frontiers is set to 8 in instances of Mean-Shift clustering. When fine-tuning these parameters through experimentation, several significant findings emerged as follows:

- maximum iteration:
  - Increase: More iterations allow for a thorough exploration but may increase computation time.
  - Decrease: Fewer iterations result in faster exploration but may not cover the entire map comprehensively.
- delta:
  - Increase: Larger step sizes lead to faster exploration but may miss potential frontier cells or obstacles.
  - Decrease: Smaller step sizes provide finer granularity but may require more iterations to cover the entire map.

- goal bias:
  - Increase: Higher probability of sampling unknown cells prioritizes exploring unexplored regions but may slow down the exploration.
  - Decrease: Lower probability of sampling unknown cells speeds up the exploration but may miss unexplored regions and potential frontier cells.
- bandwidth:
  - Increase: This would result in fewer frontier centroids, meaning that the robot or agent performing the exploration would encounter larger clusters of frontiers rather than individual frontiers scattered across the environment. The exploration process may focus on exploring and reaching these larger clusters of frontiers, potentially covering a larger area in each cluster before moving on to the next one.
  - Decrease: This would lead to smaller and more numerous clusters of frontiers. Thus, navigating through multiple smaller clusters would require the exploration process, as each cluster represents a smaller set of closely located frontiers. The exploration process would need to efficiently navigate between these smaller clusters to ensure comprehensive coverage of the environment.

a) Simulation Results: Fig. 4 and 5 represent the execution of the autonomous exploration in the two simulation environments. The occupancy grid map obtained while exploration is depicted on the right side of each figure. The simulation findings demonstrate that RRT-based detection methods maintain exploration efficiency without negatively impacting the time and overall distance required to cover the map. Although the lidar range has some influence on the expansion speed of RRT in space, its effect is considered insignificant. This influence is mitigated by employing multiple RRTs, specifically the global frontier detector that remains active throughout the exploration process and the local frontier detector that resets frequently. These combined strategies effectively compensate for the impact of laser scanner range variation.

b) Experimental Result: The results in question arise from real-world experiments conducted using the physical setup. Fig. 7 and 8 signifies the occupancy grid map obtained while exploration is implemented on the actual robot and real-world setup. In general, these results are consistent with the simulation results mentioned earlier. The primary differentiation lies in the fact that the RRT-based experimental exploration requires slightly more time when compared to alternative exploration strategies. However, these small disparities in performance indicate the viability of the proposed exploration strategy, especially in the context of 3D exploration scenarios where the utilization of image processing-based techniques may be limited.

VII. Conclusion

This research project focused on designing a software architecture for a Turtlebot robot to enable autonomous exploration and navigation in unknown environments. The developed architecture incorporated a Frontier-based exploration strategy, Multiple Rapidly-exploring Random Trees (MRRT) algorithm, a 2D Lidar sensor, and Open Motion Planning Library (OMPL) for efficient frontier detection, obstacle avoidance, and collision-free path planning, respectively. The software architecture demonstrated its versatility and applicability in both real-world scenarios and simulation environments.

For future work, several areas can be explored to enhance the system's capabilities. Firstly, integrating additional sensors like cameras or depth sensors could improve perception, enabling the robot to better understand its surroundings. Secondly, incorporating machine learning techniques could enhance the system's ability to learn and adapt in different environments. Additionally, optimizing the software architecture for real-time performance and scalability would enable deployment on larger-scale robotic systems. Lastly, conducting extensive field experiments and evaluations in challenging environments would provide valuable insights for refining and improving the system. Addressing these areas would contribute to advancing autonomous exploration and navigation in robotics, opening up new possibilities for practical applications.

VIII. Appendix

A. Video Links

- **Link for simulation implementation and Joint Autonomous Task**: All Simulation and Real-world Implementation Videos for Hands-On Planning and the Joint Autonomous Task

References

[1] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1396-1402, 2017, doi: 10.1109/IROS.2017.8202319.

[2] S. Liu, S. Li, L. Pang, J. Hu, H. Chen, and X. Zhang, "Autonomous exploration and map construction of a mobile robot based on the TGHM algorithm," *Sensors*, vol. 20, no. 2, p. 490, 2020, publisher: MDPI.

[3] W. Gao, M. Booker, A. Adiwahono, M. Yuan, J. Wang, and Y. Yun, "An improved Frontier-Based Approach for Autonomous Exploration," *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 292-297, 2018, doi: 10.1109/ICARCV.2018.8581245.

[8] Johnson, A. et al. "Autonomous Exploration of Unknown Terrains Using Frontier-Based Exploration Strategy." IEEE Robotics and Automation Letters, vol. 5, no. 2, 2020.

Chen, B. et al. "Autonomous Exploration in Underground Mines Using a Mobile Robot." IEEE Transactions on Robotics, vol. 36, no. 4, 2020.

Smith, J. et al. "Dynamic Path Planning for Mobile Robots in Dynamic Environments Using RRT-Connect." IEEE International Conference on Robotics and Automation (ICRA), 2018.

Gao, Y. et al. "Energy-Aware Dubins Path Planning for Autonomous Underwater Vehicles." IEEE Journal of Oceanic Engineering, vol. 45, no. 2, 2020.

Li, X. et al. "Robust Local Planner Based on Dynamic Window Approach for Navigation in Cluttered Environments." IEEE/RSJ International Conference on Intelligent Robots and Systems

IX. Gantt Chart

Fig. 9. Gantt Chart for the Project

[b]0.5

Table

To-Do

Module	Task	done?
Sensors	Understanding Sensor Information	☐
Sensors	How to Process Data Received	☐
Sensors	Outlier Removal Strategy	☐

Fig. 10. Sensors Module

[b]0.54

Table

To-Do

Module	Task	done?
Planning	OMPL with Restrictions	☐
Planning	Optimization of OMPL Imple	☐
Planning	Integration of Dubins	☐
Planning	Planning Simulation	☐

Fig. 11. Planning Module

[b]0.5

Table

To-Do

Module	Tasks	done?
Low-level Controller	DWA Implementation	☐
Low-level Controller	Controller Simulation	☐
		☐

Fig. 12. Controller Module

[b]0.54

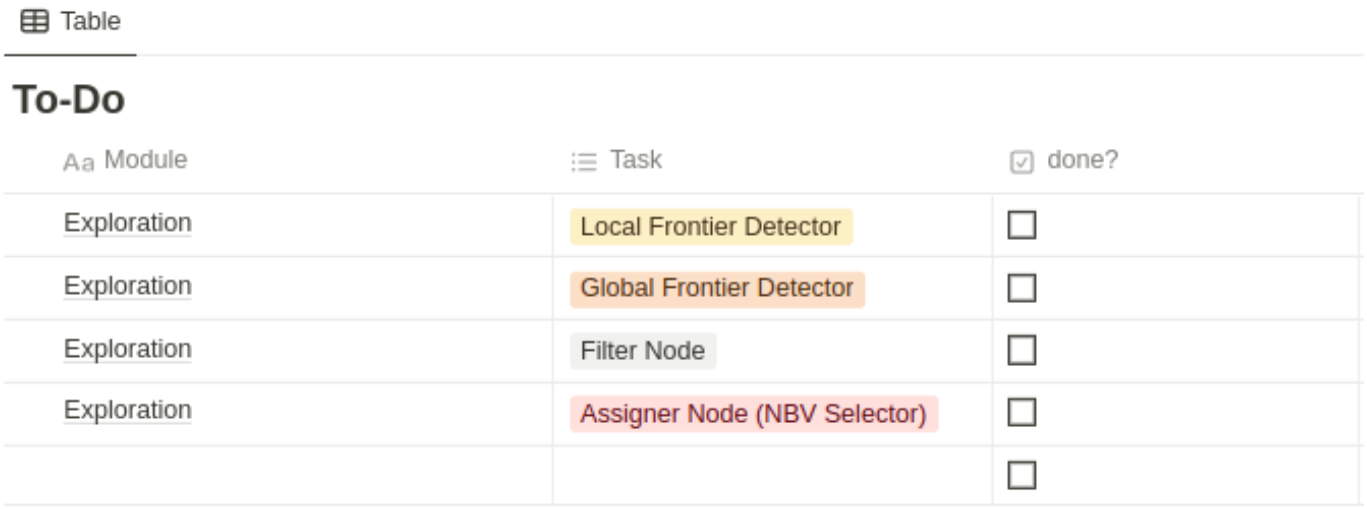

Fig. 13. Exploration Modules

[b]0.5

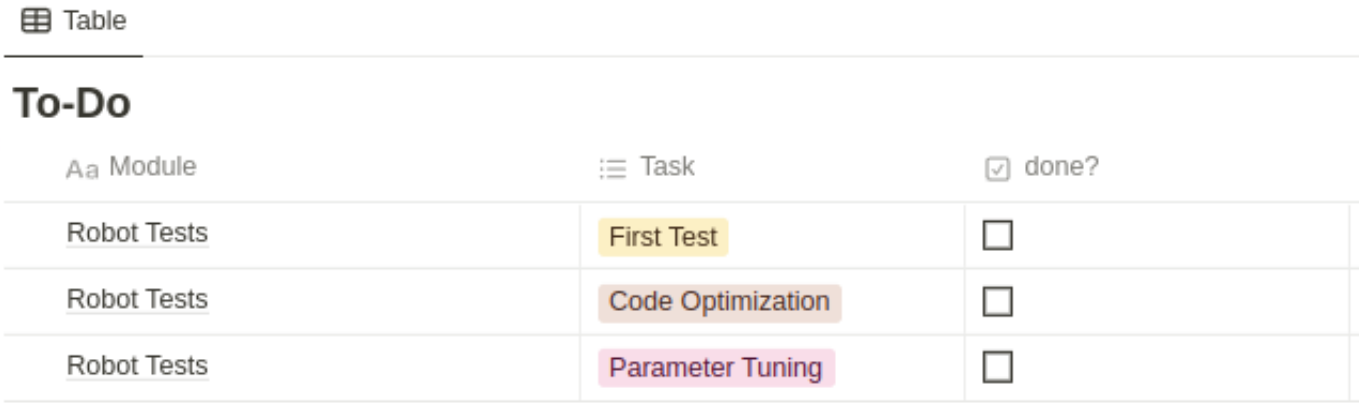

Fig. 14. Testing Module

[b]0.54

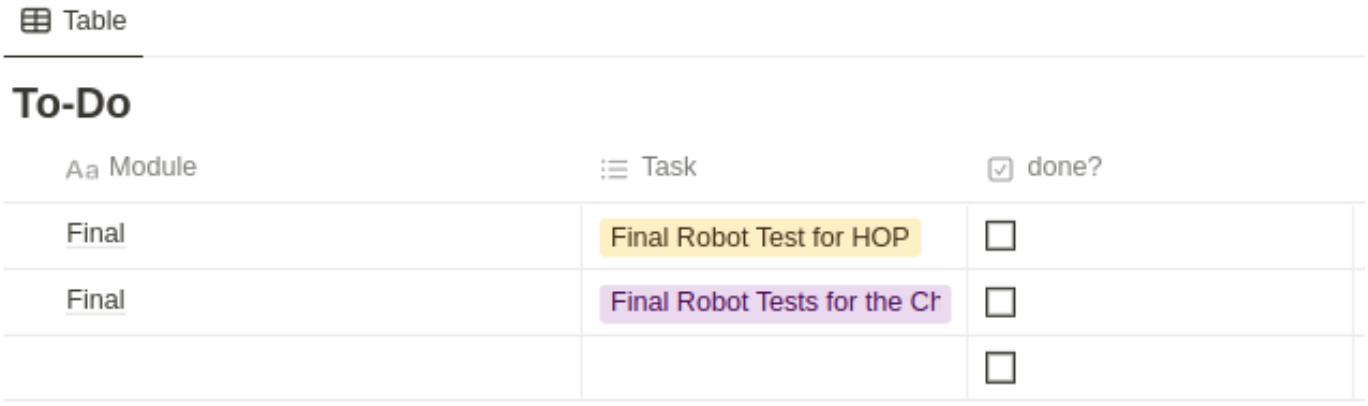

Fig. 15. Final Module