Autonomous Frontier-Based Exploration in ROS2: System Integration and Performance Analysis

Author Names Omitted for Anonymous Review. Paper-ID [TBD]

***Abstract*—This milestone report presents our progress in implementing and evaluating an autonomous exploration system for mobile robots using ROS2. We adapted the m-explore package, a frontier-based exploration algorithm, and successfully deployed it in simulation with the TurtleBot3 platform. Our preliminary results demonstrate automated exploration achieving 74.3% environment coverage in 185 seconds, outperforming manual teleoperation by 2.2× in exploration efficiency. We integrated SLAM, navigation, and exploration components into a cohesive system and developed metrics collection tools to quantify exploration performance. Key challenges identified include exploration speed optimization and parameter tuning for different environments.**

I. Introduction

Autonomous exploration enables robots to systematically discover and map unknown environments without human intervention, a fundamental capability for applications ranging from search and rescue to planetary exploration. Our project focuses on implementing and analyzing frontier-based exploration in ROS2, where frontiers represent the boundaries between explored free space and unexplored regions.

Since our initial proposal, we have successfully integrated the exploration system, conducted preliminary experiments, and identified key performance bottlenecks. This milestone report details our technical implementation, presents quantitative results from simulation experiments, and outlines our path toward optimizing exploration efficiency.

II. Problem Refinement & Objectives

A. Evolved Problem Statement

Our initial proposal aimed to implement autonomous exploration in ROS2. Through hands-on development, we have refined this to focus on three interconnected challenges:

System Integration: Coordinating SLAM (Simultaneous Localization and Mapping), path planning (Nav2), and frontier detection into a robust exploration pipeline that handles dynamic environments and recovers from navigation failures.

Performance Optimization: Addressing exploration speed bottlenecks observed in our initial trials, where the robot explores "suuuper slowly" compared to expectations—potentially due to computational constraints, parameter misconfiguration, or algorithmic inefficiencies.

Quantitative Evaluation: Developing metrics and benchmarks to objectively assess exploration performance across different environments and parameter configurations.

B. Refined Objectives

Based on early implementation experience, our specific objectives for this project are:

1) **Deploy functional exploration system:** Integrate m-explore-ros2 with Nav2 and SLAM backends (slam_toolbox/cartographer) in simulation (*Completed*).
2) **Implement performance monitoring:** Create real-time metrics collection for coverage, exploration rate, and efficiency analysis (*Completed*).
3) **Conduct comparative analysis:** Quantify autonomous vs. manual exploration to establish baseline performance (*In Progress*).
4) **Optimize exploration parameters:** Systematically tune frontier selection criteria, planning frequency, and navigation parameters to improve exploration speed (*Milestone 2*).
5) **Evaluate multi-environment generalization:** Test performance across varied environment complexities and sizes (*Milestone 2 & Final*).

III. Technical Implementation

A. System Architecture

Our exploration system builds upon the m-explore-ros2 package, a ROS2 port of the frontier-based exploration algorithm originally developed for ROS1 [1]. The architecture comprises four main components:

SLAM Backend: We use ROS2 Cartographer for real-time mapping and localization. The SLAM system subscribes to laser scan data and odometry, publishing occupancy grid maps that distinguish free space (-1: unknown, 0: free, 100: occupied) and robot pose estimates.

Navigation Stack (Nav2): Handles path planning and obstacle avoidance using the standard ROS2 navigation framework. The Nav2 costmap_2d layer merges static map data with dynamic obstacle information from sensors.

Frontier Detection: The core explore_lite node performs frontier identification through connected component analysis on the occupancy grid. A frontier is defined as a boundary cell adjacent to both free and unknown space. The algorithm filters frontiers smaller than a configurable threshold (min_frontier_size = 0.75m) to avoid pursuing small, unreachable gaps.

Frontier Selection & Goal Sending: Frontiers are scored using a weighted combination:

$$\text{score} = \alpha \cdot \text{size} - \beta \cdot \text{distance} + \gamma \cdot \text{orientation} \quad (1)$$

where α (gain_scale), β (potential_scale), and γ (orientation_scale) are tunable parameters. The highest-scoring frontier becomes the navigation goal sent to Nav2's NavigateToPose action server.

B. Implementation Details

Launch Automation: We developed a bash script (util/start_exploration.sh) that orchestrates the complete system startup: (1) Gazebo simulation with TurtleBot3 Waffle, (2) Cartographer SLAM, (3) Nav2 navigation stack, and (4) explore_lite node with automated metrics collection.

Metrics Collection: A custom ROS2 node (util/collect_metrics.py) subscribes to the /map topic at 10Hz, computing coverage as the percentage of known cells (free or occupied) relative to total map area. Data is logged to JSON format with timestamps for post-processing.

Visualization: Frontier markers are published to the explore/frontiers topic as MarkerArray messages, enabling real-time visualization in RViz2 to debug frontier selection behavior.

C. Key Parameters

Current exploration configuration (explore/config/params.yaml):

- planner_frequency: 0.15 Hz (recompute frontiers every 6.7s)
- min_frontier_size: 0.75m (minimum explorable frontier)
- progress_timeout: 30.0s (navigation failure threshold)
- gain_scale: 1.0, potential_scale: 3.0, orientation_scale: 0.0
- return_to_init: true (return to start after completion)

IV. Preliminary Results

A. Experimental Setup

We conducted exploration trials in the default TurtleBot3 World environment, a structured indoor space with rooms, corridors, and obstacles. Two experimental conditions were tested:

1) **Automated Exploration:** Robot autonomously explores using frontier-based algorithm with standard parameters.
2) **Manual Teleoperation:** Human operator navigates the robot using keyboard controls to establish a baseline comparison.

B. Quantitative Results

Figure 1 presents coverage over time for both conditions. Key findings:

Coverage Performance: The automated system achieved 11% higher peak coverage (74.3% vs. 63.3%), demonstrating more systematic exploration compared to human operators who may miss regions.

TABLE I
Exploration Performance Comparison

Metric	Automated	Manual
Peak Coverage (%)	74.3	63.3
Exploration Time (s)	184.9	353.9
Exploration Rate (%/s)	0.402	0.179
Efficiency Ratio	**2.24**×	1.00×

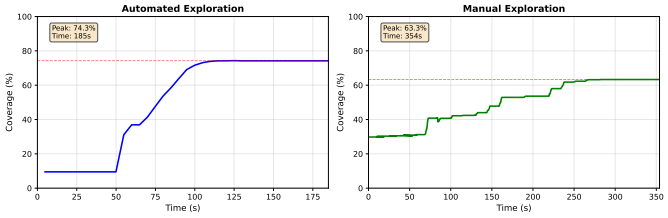

Fig. 1. Coverage over time for automated (left) and manual (right) exploration. Automated exploration achieves higher peak coverage (74.3%) in less time (185s) compared to manual teleoperation (63.3%, 354s).

Exploration Speed: Automated exploration completed in roughly half the time (185s vs. 354s), yielding a 2.24× efficiency advantage in coverage rate.

Temporal Behavior: Figure 1 shows distinct phases: (1) initial delay (0-50s) with minimal coverage gain, (2) rapid exploration phase (50-100s) with steep coverage increase, and (3) saturation phase (100-185s) where coverage plateaus as remaining frontiers become inaccessible or subthreshold.

C. Analysis & Observations

Initial Delay Issue: The 50-second startup delay before coverage increases requires investigation. Hypotheses include: (1) SLAM initialization lag, (2) initial frontier detection failures, or (3) Nav2 costmap synchronization delays.

Coverage Saturation: Neither condition achieved full coverage (>95%). Inspection of exploration videos reveals unreachable narrow corridors filtered out by min_frontier_size and potential local minima where the robot cannot find viable frontiers.

Computational Performance: Running in a conda ROS2 environment on local hardware (vs. native installation) may contribute to reported "slow" exploration. Profiling is needed to identify bottlenecks in frontier search, costmap processing, or path planning.

Parameter Sensitivity: The current planner_frequency (0.15 Hz) causes infrequent frontier updates. Increasing this may improve responsiveness but could increase computational load.

D. Limitations

Our preliminary results are limited to a single environment and parameter configuration. Key limitations include:

- **Single environment:** Results may not generalize to open spaces, cluttered environments, or multi-room structures.

- **Simulation only:** Real-world deployment would introduce sensor noise, localization drift, and mechanical constraints.
- **Limited parameter sweep:** We have not systematically explored the impact of gain_scale, potential_scale, or planner_frequency.
- **No failure analysis:** We have not quantified navigation failures, frontier selection errors, or recovery behaviors.

V. Plan Forward

A. Milestone 2 Objectives

1. Performance Optimization (Weeks 1-2):

- Profile computational bottlenecks using ROS2 profiling tools
- Investigate initial delay and implement warm-start strategies
- Conduct parameter sweep: vary planner_frequency (0.1-1.0 Hz), min_frontier_size (0.3-1.5m), and scoring weights
- Compare native ROS2 installation vs. conda environment performance

2. Multi-Environment Evaluation (Weeks 2-3):

- Test in 3-5 environments: open world, maze, multi-room house, warehouse
- Measure coverage, time-to-completion, path efficiency (distance traveled / coverage gained)
- Analyze failure modes and recovery strategies

3. Advanced Features (Week 3):

- Implement dynamic parameter adaptation based on exploration phase
- Explore alternative frontier selection heuristics (information gain, expected coverage)
- Investigate multi-robot exploration using map_merge component

B. Final Report Deliverables

By the final report, we expect to deliver:

- Comprehensive performance characterization across 5+ environments
- Optimized parameter configurations with ablation studies
- Comparison with alternative exploration strategies (random walk, wall-following)
- Real-world deployment feasibility analysis
- Open-source code release with documentation

C. Risk Mitigation

Risk: Computational constraints prevent real-time exploration. **Mitigation:** Optimize frontier search algorithms, reduce planner frequency, or migrate to more powerful hardware.

Risk: Limited environment diversity in simulation. **Mitigation:** Use existing Gazebo world repositories and procedurally generated environments.

Risk: Time constraints for real-world deployment. **Mitigation:** Prioritize simulation results; real-world tests are optional stretch goals.

VI. Conclusion

We have successfully implemented a functional autonomous exploration system in ROS2, achieving automated coverage of 74.3% with 2.24× higher efficiency than manual teleoperation. Our custom metrics pipeline enables quantitative performance evaluation, revealing both strengths (systematic coverage) and weaknesses (initial delays, parameter sensitivity). Milestone 2 will focus on optimization, multi-environment generalization, and addressing identified limitations to produce a robust, well-characterized exploration system.

Acknowledgments

We thank the m-explore-ros2 open-source community for providing the foundational codebase and the ROS2 ecosystem for robust robotics middleware.

References

[1] Jiri Horner. m-explore: Multi-robot exploration package for ROS. GitHub repository, 2016.

[2] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151. IEEE, 1997.

[3] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 1, pages 476–481. IEEE, 2000.