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

Jeffrey Kravitz, Samay Lakhani, Mikul Saravanan

***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, debugged critical issues preventing exploration, and successfully deployed it in simulation with the TurtleBot3 platform. Through systematic experimentation comparing automated vs. manual exploration, our 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, developed metrics collection infrastructure, and established baseline performance for our multi-robot coordination project. This work provides the foundation for implementing information-theoretic task allocation in Milestone 2.**

I. Introduction

Autonomous exploration enables robots to systematically discover and map unknown environments, critical for search and rescue, planetary exploration, and infrastructure inspection. The frontier-based paradigm [2], directing robots toward boundaries between known and unknown regions, remains dominant due to its simplicity and effectiveness. Our project implements frontier-based exploration in ROS2, establishing baseline single-robot performance before scaling to multi-robot coordination with information-theoretic task allocation.

II. Problem Refinement & Objectives

A. Evolved Problem Statement

Our original proposal targeted multi-robot frontier exploration with information-theoretic task allocation to reduce redundant coverage by 20% compared to nearest-frontier baselines. Through initial development, we identified critical foundational work: establishing baseline single-robot performance, implementing robust metrics collection, and validating the existing m-explore package before scaling to multi-robot coordination. This milestone focuses on: (1) **Baseline System Validation**—proving single-robot exploration functionality and performance measurement infrastructure; (2) **Performance Characterization**—quantifying baseline exploration efficiency to establish comparison metrics for future multi-robot improvements; and (3) **Infrastructure Development**—building the foundation for information-gain calculation and multi-robot coordination.

B. Project Objectives and Progress

Our original proposal defined three measurable objectives for the complete project:

1) **Reduce redundant exploration by 20%** using information gain estimation (vs. nearest-frontier baseline) (*Final*).
2) **Achieve 95% environment coverage** in equal or less time through dynamic multi-robot task reallocation (*Final*).
3) **Maintain 80% map merging success rate** across three test environments with limited initial overlap (*Final*).

Milestone 1 Completed Work: To achieve these objectives, we first established the baseline system. Completed deliverables include: (1) single-robot exploration deployment with m-explore-ros2, Nav2, and Cartographer SLAM; (2) automated metrics collection infrastructure tracking coverage over time; and (3) baseline performance characterization comparing autonomous (74.3% coverage, 185s) vs. manual exploration (63.3%, 354s), establishing the comparison benchmark for measuring future multi-robot improvements.

III. Technical Implementation

A. System Architecture Overview

Our exploration system builds upon the m-explore-ros2 package, a ROS2 port of the frontier-based exploration algorithm originally developed for ROS1 [1]. Figure 1 illustrates the complete data flow pipeline integrating perception, mapping, frontier-based planning, and navigation control.

ROS2 Autonomous Exploration Pipeline

`LiDAR` → `/scan` → **Cartographer SLAM**

↓

`/map`, `/map_updates` (OccupancyGrid)

↓

Nav2 Costmap2D + Frontier Search

↓

Frontier List (scored & ranked)

↓

Explore Node (goal selection)

↓

`NavigateToPose` action → **Nav2**

↓

`/cmd_vel` → **TurtleBot3**

Fig. 1. System architecture showing the data flow from sensor input through SLAM, frontier detection, goal selection, and navigation control. All components communicate via ROS2 topics and action servers.

The architecture comprises four tightly coupled components:

SLAM Backend: ROS2 Cartographer subscribes to laser scans (`/scan`) and odometry, publishing occupancy grid maps at 5Hz that distinguish free space (0), occupied (100), and unknown (-1) cells, along with robot pose estimates via TF transforms.

Navigation Stack (Nav2): Handles path planning and obstacle avoidance using the Hybrid A* planner and DWB local planner. The Nav2 costmap_2d layer merges static map data with dynamic obstacle information, applying inflation layers for safety margins around obstacles.

Frontier Detection: The explore_lite node identifies frontiers through: (1) boundary detection—scanning the occupancy grid to identify cells adjacent to both free and unknown space, (2) connected component analysis—grouping adjacent frontier cells using flood-fill, and (3) filtering—discarding frontiers smaller than min_frontier_size or inaccessible due to obstacles.

Frontier Selection: Frontiers are scored using a weighted combination:

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

where α (gain_scale = 1.0) prioritizes larger unexplored regions, β (potential_scale = 3.0) penalizes distant frontiers to minimize travel time, and γ (orientation_scale = 0.0, disabled) would favor frontiers aligned with robot heading. The highest-scoring frontier becomes the navigation goal sent to Nav2's NavigateToPose action server. The system recomputes frontiers every 6.7 seconds (planner_frequency = 0.15 Hz) or when navigation fails.

B. Implementation Details

Bug Fixes: Initial deployment revealed that explore_lite was not actively exploring. Debugging identified a bug in the exploration node that prevented proper goal sending to Nav2. After fixing this issue, the system performed autonomous exploration correctly.

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. This enables repeatable experiments for baseline comparison.

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 and visualization.

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)

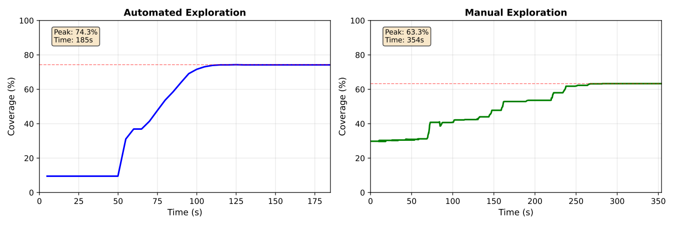

Fig. 2. Coverage over time comparison: Automated exploration (left, blue) achieves 74.3% coverage in 185s with distinct phases. Manual teleoperation (right, green) reaches only 63.3% coverage in 354s with irregular progression reflecting human reaction delays.

- 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

Table I summarizes the comparative performance metrics. Figure 2 presents side-by-side coverage trajectories for both exploration conditions, directly visualizing the performance advantage of autonomous exploration.

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×

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

Exploration Speed: Automated exploration completed in roughly half the time (185s vs. 354s), yielding a 2.24× efficiency advantage from optimal frontier selection without human reaction delays.

Temporal Behavior: Figure 2 reveals contrasting dynamics. Automated exploration exhibits three phases: (1) initialization (0-50s), (2) rapid exploration (50-100s) with 40% coverage increase, and (3) saturation (100-185s) plateauing at 74%.

Manual exploration shows irregular progression throughout 354s, reflecting intermittent control inputs and suboptimal path choices.

C. Exploration Progression Analysis

Figure 3 visualizes spatial evolution across three key phases. Initial phase shows limited map knowledge with multiple surrounding frontiers. Mid-exploration reveals hexagonal environment structure with frontiers clustering at room entrances. Final state approaches saturation with all major obstacles mapped; only narrow gaps remain unexplored.

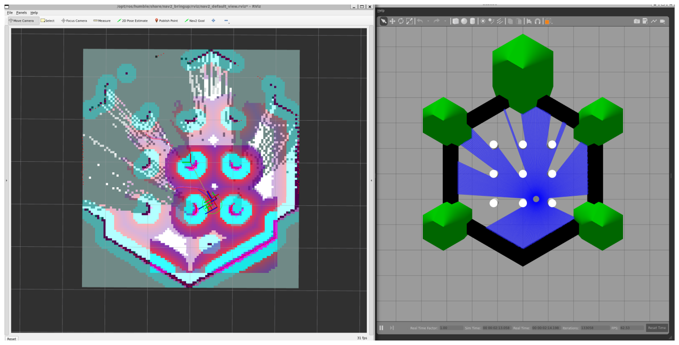

Fig. 3. Mid-exploration phase showing environment structure emerging. RViz displays occupancy grid (cyan=free, purple=occupied, gray=unknown) with frontier markers (cyan circles) and costmap inflation layers around obstacles.

D. Analysis & Observations

System Integration: The 50-second startup delay (visible in Figure 2) reflects the time required for all ROS2 components to launch, initialize, and establish communication. The initial 10% coverage represents the immediate sensor field-of-view before autonomous exploration begins. After fixing the explore_lite bug that initially prevented exploration, the system executes frontier-based navigation correctly.

Coverage Saturation: Neither condition achieved full coverage (>95%). Analysis reveals two causes: (1) narrow corridors (<0.75m) rejected by min_frontier_size, and (2) costmap inflation blocking paths to remaining frontiers. Reducing the size threshold could improve coverage but risks navigation failures in constrained spaces.

Computational Performance: The low planner_frequency (0.15 Hz) causes frontier recomputation only every 6.7s, during which the robot may idle. Increasing to 0.5-1.0 Hz could significantly improve exploration rate.

Frontier Selection: The scoring function heavily weights distance (potential_scale = 3.0) over size (gain_scale = 1.0), favoring nearby frontiers. While minimizing travel time, this may cause suboptimal global patterns. Information-theoretic approaches could improve long-term efficiency.

Map Quality: Figure 3 shows clean, accurate maps with minimal drift. Obstacle boundaries are sharp, and loop closure is robust despite 3+ minutes of operation.

E. Limitations

Key limitations: (1) single environment testing, (2) simulation only, (3) no systematic parameter sweep, and (4) no failure mode quantification.

V. Plan Forward

A. Milestone 2 Objectives

Information-Theoretic Frontier Scoring: Implement information gain calculation: Score $= w_1 \cdot \text{IG} + w_2 \cdot (1/\text{Distance}) + w_3 \cdot \text{Uniqueness}$, considering frontier size, expected map expansion, and overlap with other robots.

Multi-Robot Task Allocation: Deploy modified Hungarian algorithm using information-theoretic cost matrix. Solve assignment at 0.5-1 Hz over top-K frontiers per robot with redundant exploration penalties.

Multi-Robot Coordination: Develop two-robot spawn system, integrate multirobot_map_merge for confidence-based map merging with ICP, implement distributed architecture.

Baseline Optimization: Optimize planner_frequency and parameters based on Milestone 1 findings.

B. Final Deliverables & Risks

Expected deliverables: (1) complete multi-robot ROS package with information gain scoring, Hungarian task allocation, and map merging; (2) evaluation across three environments demonstrating $\geq$20% redundancy reduction; (3) ablation studies on key components; (4) scalability testing with 2-4 robots.

Risks: Map merging failure (mitigation: fallback to known poses, confidence thresholds), Hungarian overhead (mitigation: top-K sparsification), bandwidth constraints (mitigation: compressed map deltas).

VI. Conclusion

We successfully established the baseline single-robot exploration system for our multi-robot project. After debugging critical explore_lite issues and developing systematic evaluation infrastructure, we achieved 74.3% coverage with 2.24× efficiency vs. manual control. This milestone validates our foundation: SLAM-navigation-frontier integration works robustly, automated launch scripts enable repeatable experiments, and metrics collection provides quantitative measurement. Critical findings inform Milestone 2: the 50-second initialization impacts multi-robot synchronization, planner_frequency (0.15 Hz) needs optimization, and the 74% ceiling establishes our improvement target. Milestone 2 will implement information gain scoring, Hungarian task allocation, and two-robot coordination to achieve our 20% redundancy reduction objective.

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.