Full Project Report

Autonomous Frontier Exploration with Semantic Hazard Mapping

Princess Colon · Manjunath Kondamu · Rohit Mane
RAS 598: Mobile Robotics · Arizona State University · Spring 2026

A comprehensive account of designing, building, debugging, and validating an autonomous ground robot that explores unknown environments, builds real-time occupancy maps, and detects semantic hazards — entirely without human teleoperation.

1. Project Overview

This project implements a fully autonomous ground robot system capable of exploring unknown indoor environments, building real-time occupancy maps, and detecting semantic hazards — all without any human teleoperation or pre-loaded maps.

The platform is a TurtleBot4 Lite running ROS 2 Jazzy on Ubuntu 24.04. The robot is equipped with an RPLIDAR A1M8 360-degree LiDAR for mapping and navigation, an OAK-D spatial AI stereo camera for depth sensing and hazard classification, a Create3 mobile base with onboard sensors (cliff, bump, IMU, wheel encoders), and a Raspberry Pi 4B as the onboard computer.

The complete system consists of five custom ROS 2 nodes communicating over topics and action servers, plus SLAM Toolbox for mapping. The project spans three course milestones over Spring 2026, from initial architecture design to validated empirical benchmarking.

> **Key Result:** A working autonomous exploration system that detects frontier cells, plans collision-free paths using A*, and classifies HAZMAT signs, fire, cliffs, potholes, and narrow corridors using a 5-layer sensor fusion pipeline. Running in both Gazebo simulation (maze world) and on real hardware.

2. Motivation & Goals

The core problem we wanted to solve: how can a robot safely enter an unknown dangerous environment and give first responders a complete picture of what is inside before they enter?

Real-world applications include post-disaster search and rescue (collapsed buildings, fire scenes), industrial inspection (chemical plants, confined spaces), and nuclear facility assessment.

Goals

- Autonomous map-building in unknown environments using SLAM (no pre-loaded map required)
- Systematic frontier-based exploration — explore the nearest unknown areas first
- Collision-free navigation without Nav2 (custom A* planner)

- Semantic hazard detection: HAZMAT signs (13 classes), fire, cliffs, potholes, narrow corridors, glass walls
- Real-time RViz2 visualization of map, robot pose, path, and hazard markers
- Unified launch for both simulation and real hardware via single flag

3. System Architecture

The system is built as five custom ROS 2 nodes communicating over a well-defined topic interface. The design principle: clear separation of concerns — each node has one job.

ROS 2 Topic Graph

- slam_toolbox → /map → frontier_explorer, navigation_planner
- frontier_explorer → /frontier_goals (PoseArray) → behavior_coordinator
- behavior_coordinator → /navigate_to_pose (action) → navigation_planner
- navigation_planner → /cmd_vel_unstamped (Twist) → Create3 base
- semantic_hazard → /hazard_alert (String), /exploration/stop (Bool), /hazard_markers (MarkerArray)
- Create3 → /hazard_detection (HazardDetectionVector) → semantic_hazard
- scan_relay → /scan_fixed (LaserScan) → slam_toolbox (sim only)

Hardware Setup

- Robot: TurtleBot4 Lite — Create3 base + Raspberry Pi 4B + OAK-D camera + RPLIDAR A1M8
- Development compute: NVIDIA RTX 5070 Ti, CUDA 13.1, Ubuntu 24.04
- Networking: ROS 2 Jazzy with Zenoh discovery server at 192.168.50.93:11811
- Simulation: Gazebo Sim 8 (Harmonic), turtlebot4_gz_bringup, maze world

4. SLAM & Mapping

We use slam_toolbox in async mapping mode. The robot builds a 2D occupancy grid at 0.05m/cell resolution as it moves. The SLAM node processes LiDAR scans and publishes a

map-to-odom TF transform that the entire navigation stack depends on.

Key Parameters: slam_sim.yaml: scan_topic: /scan_fixed (simulation), resolution: 0.05, map_update_interval: 2.0, transform_timeout: 0.2, tf_buffer_duration: 20.0.

Real vs Sim: On real hardware, SLAM starts and builds maps immediately — the RPLIDAR A1M8 publishes in rplidar_link frame which matches exactly what SLAM expects. In simulation, the LiDAR publishes in turtlebot4/rplidar_link/rplidar frame (a Gazebo naming quirk) which required a scan relay node to fix the frame_id.

5. Frontier Exploration

Frontier exploration is the core algorithm for autonomous navigation in unknown environments. A frontier is defined as a free cell adjacent to an unknown cell — the boundary between what the robot knows and does not know.

Algorithm: The frontier explorer runs every time a new map is received (approximately 1Hz). It scans all free cells, identifies those adjacent to unknown cells (and not adjacent to occupied cells), groups them into clusters via 4-connected flood fill, discards clusters with fewer than 10 cells (noise), and scores each cluster by inverse distance from the robot — nearest first.

Scoring Formula: score(c_k) = 1.0 / (dist(robot, centroid_k) + epsilon), epsilon = 1e-6

Scoring Evolution

- Original: score = (alpha * cluster_size + beta * unknown_nearby) / (dist + epsilon) — weighted combination. This caused the robot to chase large unknown areas rather than exploring systematically.
- Final (nearest-first wavefront): score = 1.0 / (dist + epsilon) — simply go to the closest frontier. Combined with the proximity lock in the behavior coordinator, this produces systematic nearest-first exploration.

Key Insight: Nearest-first scoring is simpler AND better in practice. The robot explores nearby areas completely before moving to distant frontiers.

6. Navigation Planner

We built a complete custom navigation planner as a ROS 2 action server implementing the /navigate_to_pose interface — the same interface Nav2 uses. This is a drop-in replacement for Nav2 without requiring any Nav2 infrastructure.

Why Not Nav2?

- Nav2 costmap inflation was very aggressive and kept marking valid frontier cells as obstacles
- Nav2 adds significant complexity with many parameters to tune
- Custom planner gives full control over exact behavior

What We Built

- Costmap builder: 3 tiers — free (0), soft inflation zone (10), hard inflation zone (80), wall (1000). Wall cells dilated by 3cm before inflation.
- A* path planner: 8-connected grid search with Euclidean heuristic. Costs from the costmap bias the planner away from walls.
- Waypoint pruning: Removes intermediate waypoints with line-of-sight to each other (swept-corridor test, robot half-width 0.20m).
- P-controller drive loop: 20Hz. Aligns to waypoint heading first (in-place rotation), then drives forward.

Key Tuned Parameters

- ALIGN_THRESHOLD = 0.20 rad (was 0.10 — too tight, caused oscillation)
- SNAP_SEARCH_RADIUS = 20 cells (was 10 — frontier goals landing in walls)
- V_MAX = 0.35 m/s, V_MIN = 0.10 m/s, W_MAX = 1.20 rad/s
- HARD_RADIUS = 0.15m, SOFT_RADIUS = 0.18m, WALL_DILATION = 0.03m

7. Semantic Hazard Detection

The hazard classifier fuses five sensor layers into a unified hazard alert published on /hazard_alert. A dedicated YOLO worker thread runs inference at most once per second to avoid blocking the main executor.

5-Layer Architecture

- L1 Hardware (Create3): Subscribes to /hazard_detection. Cliff sensors trigger immediate E-stop. Response time less than 10ms.
- L2a YOLO Fire (YOLOv5): Custom-trained fire_best.pt model. Confidence threshold 0.45. Detects fire from phone screen at 63% confidence.
- L2b HSV Vision: Fixed thresholds for fire (H in [0,15] or [165,179], S>150, V>200). Water detection disabled — caused false positives on blue walls.
- L3 Depth Analysis (OAK-D): NaN ratio >50% in floor region → GLASS. Depth drop >0.20m → POTHOLE (implemented, hardware testing pending).
- L4 LiDAR Sector: Minimum ranges in 3 sectors after correcting for the +90 degree RPLIDAR mounting offset. NARROW_CORRIDOR if any sector <0.30m.
- L5 DeepHAZMAT: YOLOv3-tiny model with 13 HAZMAT classes. 7M parameters, 15.8 GFLOPs. Approximately 30ms inference on RTX 5070 Ti.

LiDAR Angle Correction: The RPLIDAR A1M8 is mounted with a +90 degree rotation offset in the TurtleBot4 Lite URDF (rpy="0 0 pi/2"). Without correcting for this, the sector analysis computes wrong angles. Fix: theta_robot = theta_lidar - pi/2 before all sector computations.

YOLO Worker Thread: A critical design decision: YOLO inference runs in a dedicated background thread with a 1-second cooldown (YOLO_INTERVAL_SEC = 1.0). This prevents YOLO from blocking the main ROS 2 executor.

8. Behavior Coordinator

The behavior coordinator implements a 6-state FSM (IDLE → SELECTING → PENDING → NAVIGATING → ARRIVED → DONE) that manages the exploration lifecycle.

Proximity Lock: Without the proximity lock, the robot would constantly switch targets as the SLAM map updated and new frontiers appeared closer. The lock says: if the robot is within 2 x PROXIMITY_LOCK_RADIUS (3.0m) of its current goal anchor, ignore all frontier updates. Changed from 1.0m → 2.0m → 3.0m through testing.

Frontier Similarity Radius: Frontiers drift as the map updates. Rather than cancelling a goal every time the centroid drifts by a few cells, we check if any current frontier is within 0.50m of the original goal anchor. If yes, update coordinates silently.

Max Goal Failures: After 5 consecutive failures for the same frontier anchor, it is blacklisted and a new frontier is selected. This prevents infinite replanning loops.

9. Simulation Debugging

Bug 1: LiDAR all 0.164m: Root cause: Gazebo Sim 8 uses Ogre1 renderer for GPU ray sensors. TurtleBot4 self-collision geometry caused all 640 rays to hit the robots own mesh at minimum range. Fix: Change render_engine from ogre to ogre2 in create3.urdf.xacro (requires sudo). GitHub upstream issue #676.

Bug 2: SLAM frame_id mismatch: Root cause: Gazebo publishes LiDAR in frame turtlebot4/rplidar_link/rplidar but SLAM expects rplidar_link. Fix: Created scan_relay_node.py that republishes /scan → /scan_fixed with corrected frame_id.

Bug 3: SLAM scan_topic overwritten: Root cause: turtlebot4_navigation SLAM launch uses RewrittenYaml which programmatically overwrites scan_topic — ignoring any custom yaml passed in. Fix: Bypass turtlebot4 SLAM launch entirely. Run slam_toolbox directly with slam_sim.yaml. Manual lifecycle activation required.

Bug 4: diffdrive_controller rejecting commands: Root cause: Navigation planner uses wall clock time but Gazebo runs on sim time. The diffdrive_controller rejects commands older than 0.5 seconds — wall clock timestamps appeared stale from sim clocks perspective. Fix: ros2 param set /diffdrive_controller cmd_vel_timeout 2.0 plus use_sim_time:=true on all nodes.

Bug 5: SLAM stuck at unconfigured: Root cause: When slam_toolbox is launched directly, it initializes but does not auto-configure/activate. It waits in unconfigured state. Fix: Manual lifecycle trigger — ros2 lifecycle set /slam_toolbox configure then activate. The run_exploration.sh script handles this automatically.

Bug 6: Frontier goals in obstacle cells: Root cause: Frontier centroids land at the boundary between free and unknown space — which in the costmap often falls within the wall inflation zone. Fix: Increased SNAP_SEARCH_RADIUS from 10 to 20 cells.

10. Hardware Testing

What worked well

- SLAM builds maps immediately — real RPLIDAR publishes in correct frame, no relay node needed
- DeepHAZMAT detected Explosive 1.1 at 92% confidence and Radioactive I at 99% confidence at ~0.5m range
- Cliff sensor E-stop worked reliably — tested on table edge, triggered within 10ms
- Navigation planner navigated corridors without bumping walls with tuned parameters
- YOLOv5 fire detection worked at 63% confidence on phone screen video

Issues on hardware

- Camera processing backlog (~30 second delay) on Princess machine — fixed by pulling latest YOLO worker thread version
- HAZMAT markers placed at fixed 0.70m ahead of robot rather than actual depth-estimated distance
- WiFi latency causes occasional topic dropouts — Zenoh discovery server must be reachable

Run Order: source ~/mobile-robotics-frontier-exploration/open_ros_env.sh && source ~/mobile-robotics-frontier-exploration/ros2_ws/install/setup.bash && ros2 launch frontier_exploration_mapping exploration.launch.py use_sim_time:=false

11. Issues Found — Complete List

Every problem encountered and resolution:

- 1. LiDAR all 0.164m in sim — Ogre1 renderer self-collision — Change render_engine to ogre2
- 2. SLAM not building map (sim) — frame_id mismatch — scan_relay_node.py
- 3. SLAM ignoring custom scan_topic — RewrittenYaml override — Bypass turtlebot4 SLAM launch
- 4. Robot not moving (sim) — diffdrive_controller timestamp mismatch — cmd_vel_timeout 2.0
- 5. SLAM stuck unconfigured — Lifecycle not auto-activated — Manual lifecycle set configure + activate

• 6. All frontier goals surrounded by obstacles — Frontiers in wall inflation zone — SNAP_SEARCH_RADIUS = 20

• 7. Robot oscillating between frontiers — Score changes every map update — PROXIMITY_LOCK_RADIUS = 3.0m

• 8. ALIGN_THRESHOLD too tight causing oscillation — 0.10 rad → robot kept rotating — ALIGN_THRESHOLD = 0.20 rad

• 9. Camera 30 second backlog (Princess) — Old YOLO blocking main thread — Threaded YOLO worker + 1Hz cooldown

• 10. LiDAR sector angles wrong — +90 degree URDF mounting offset ignored — theta_robot = theta_lidar - pi/2

• 11. Water detection false positives — HSV water range matched blue walls — Disabled water HSV detection

• 12. Multiple scan_relay nodes running — Old inline Python scripts + new node — Kill all before fresh start

• 13. Root-level build/install folders — colcon build run from wrong directory — Always build from ros2_ws/

• 14. Hazard marker placement offset — Fixed 0.70m ahead, no depth projection — Known limitation

12. What We Learned

Simulation vs Hardware: Every bug we fixed in simulation was a different bug than what exists on hardware. Always test on real hardware as early as possible.

Simple beats complex: The original frontier scoring formula sounded more principled, but nearest-first with a proximity lock is simpler and works better.

Simulation time is subtle: Wall clock vs sim clock mismatches can manifest in unexpected ways. Always add use_sim_time to every node.

Threading matters: YOLO inference at 30ms/frame is fast enough, but blocking the main ROS executor with it caused a 30-second backlog.

Check your URDF: The LiDAR angle correction (+90 degrees) was buried in the TurtleBot4 URDF and took significant time to find.

What we would do differently

- Start with real hardware testing earlier
- Use Nav2 lifecycle manager properly or build our own
- Add depth-based marker placement from the start instead of fixed offset
- Implement a proper rosbag-based regression test suite

13. Future Work

- Depth-based hazard marker placement: Use OAK-D depth frame to project 2D bounding box center through camera intrinsics to 3D world position
- Semantic costmap integration: Mark detected hazard zones as high-cost in the navigation costmap
- Hardware pothole testing: Pothole depth analysis is implemented but not yet tested on real hardware
- Predictive hazard propagation: Mark adjacent cells as high-risk before the robot reaches a detected hazard
- Next-Best-View frontier scoring: Replace nearest-first with information-gain-per-meter-traveled scoring
- Multi-robot coordination: Partition the environment between multiple robots to reduce exploration time
- 3D mapping: Extend from 2D OccupancyGrid to 3D voxel map using OctoMap
- Multi-floor exploration: Add floor-change detection and staircase navigation

RAS 598: Mobile Robotics · Arizona State University · Spring 2026 · Group 1

Princess Colon (pcolon@asu.edu) · Manjunath Kondamu (mkondamu@asu.edu) · Rohit Mane (rmane2@asu.edu)

GitHub: github.com/PriColon/mobile-robotics-frontier-exploration