

Development of a Dual Robotic System with a SLAM-like Autonomous Exploration System

Aditya Pillai, Yash Kakade

December 30, 2025

1 Overview

Within this project, our goal was to implement a SLAM-like auto-exploration system in a specific mapped out environment. The motivation for this project takes inspiration from missions to explore uncharted areas, specifically those in extraterrestrial environments like Mars or the moon. In this project specifically, our goal was to simulate two robots mapping out a given environment given they have perfect ability to localize themselves. In this case SLAM is not exactly happening due to the perfect localization, but our main goal was for the robots to map out the area effectively.

1.1 Goals

For this project, we were a bit ambitious with our approach, and, as a result, ended up with quite a few goals that we wanted to achieve as our final product.

- **Autonomous Exploration:** We wanted to implement our SLAM-like autonomous exploration algorithm, which involved efficiently mapping out a given space as best as possible while avoiding obstacles. We wanted to effectively recreate the idea of robots on extraterrestrial worlds being able to take map out a space without human intervention. This is a rather significant limitation due to the distance between the operators and the robots.
- **Realistic Software and Visualization Implementation** We wanted to develop and simulate this system in a realistic software environment that is on par with what is expected in the robotics industry and research. As such, we aimed to implement our project in `ROS2` with support for `RViz` and `Gazebo` running on Ubuntu. Though we are unfamiliar, this software framework seems to be the most viable for developing complex robotic systems.
- **Multi-Robot System** Though one robot seems to be feasible for missions in real life (Curiosity Rover, Perseverance Rover, etc.) probably due to cost constraints, we aimed to implement 2 robots into our mapping system for an extra challenge. These robots would individually act as sensors transmitting data to the same shared map, which allows for a faster mapping process.

2 Approach

The task that we had designed for ourselves was, thus, quite loaded and a bit overbearing. Here is how we took care of each portion.

2.1 Autonomous Exploration Algorithm

This algorithmic design was quite hefty and implemented a few steps, which are listed below. Correct implementation of all of the results and data transfer between different nodes in `ROS2` should yield the same result.

- **Mapping Algorithm:** The first part of this process involves implementing the correct mapping and localization algorithm. In the case of this project, since we were building off of our work in homework 6, where we'd already set up an environment in ROS2 with visualization in RViz. LiDAR sensing and localization was already implemented, which allowed for data to be read and the map to be created via log-odds scheme.
- **Path Planning:** Planning paths to far-away nodes for exploration purposes was completed via implementation of a Rapidly-exploring Random Tree (RRT). This RRT grew out in the robot's immediate detected free space and was implemented with natural obstacle avoidance along the path that it picked. However, the path planning was customized a bit to form a buffer region set a perpendicular distance around the paths to ensure that the robot didn't collide with obstacles as it navigated to the target node. Additionally, nodes were given a buffer region to ensure nodes didn't get too close to detected obstacles and were thus not explorable.
- **Exploration Cost Strategy:** The final crucial step for this exploration algorithm involved the cost assigned to each of the nodes that the RRT grew to in the robot's free space. This function picked the node of those discovered by the tree that was, on average, the **farthest away from all other nodes that the robot had traveled to and thus explored already**. This biased the robot to pick nodes in new areas that it had not been in previously and led to good results in exploration of the environment.

2.2 System Architecture with Dual-Robot System

- **Main SLAM Node:** The Main SLAM Node combines LIDAR data from both robots into a single, coherent map. It uses a grid-based approach where each cell represents the probability of occupancy. This node is designed to handle instances where robots may perceive the same area differently, ensuring that the final map remains consistent even when data is gathered from varying positions. Essentially, both robots LIDAR act as individual sensors that contribute to one map that is shared between the two.
- **Sensor System:** We are using LIDAR as our sensors to scan the surrounding environment comprehensively. It processes the raw scan data to extract useful information and accurately tracks the position of each robot by combining wheel movement data with scan matching techniques. This integration of data ensures that the robots always know their precise locations in the environment.
- **Mapping:** For mapping, we implemented a log-odds-based occupancy grid representation, which was chosen over feature-based SLAM methods after careful consideration of the trade-offs. Grid-based methods provide computational efficiency and work particularly well in structured environments with defined walls and obstacles. The mathematical foundation of the log-odds approach enables straightforward integration of multiple sensor readings by simply updating the log-odds values of the corresponding grid cells. Furthermore, the occupancy grid representation is compact and requires minimal bandwidth for sharing between robots, which is crucial for real-world deployments where communication constraints exist.
- **Display and Simulation System:** The Display and Simulation System leverages Gazebo for physics simulation and robot visualization, while RViz provides real-time displays of SLAM and sensor data processing. This system merges the sensor data from both robots into a single unified map, avoiding the complexities of maintaining separate SLAM instances. It visually presents the current map, sensor outputs, robot positions, and planned paths, and supports both autonomous exploration and manual control modes, allowing seamless transitions without the need for a system restart.

3 Technical Details

Each of the subsections below discusses the technical implementation for each unique portion of our project.

3.1 Random Navigation and Path Planning

The navigation system employs a modified RRT algorithm that has been enhanced for exploration scenarios. The core algorithm is presented below:

3.1.1 Pseudocode for Specialized RRT Algorithm

Algorithm 1: Specialized RRT-Based Random Navigation Algorithm

Input: Occupancy grid map from `slam.py`, robot's current pose from TF
Output: Velocity commands guiding the robot towards unexplored areas

```
1: while task not completed do
2:   Receive the current occupancy map and transform to get robot pose;
3:   Initialize RRT with root node at the current robot position;
4:   for iteration = 1 to MAX_ITERATIONS do
5:     Sample a random free point ( $x_{sample}, y_{sample}$ ) from map;
6:     Find nearest node  $n_{near}$  in RRT to ( $x_{sample}, y_{sample}$ );
7:     Steer towards ( $x_{sample}, y_{sample}$ ) from  $n_{near}$  to get ( $x_{new}, y_{new}$ );
8:     if path from  $n_{near}$  to ( $x_{new}, y_{new}$ ) is collision-free with safety buffer then
9:       Add new node  $n_{new}$  to the tree with parent  $n_{near}$ ;
10:    Find the furthest valid node  $n_{goal}$  in the tree with safe clearance;
11:    Backtrack from  $n_{goal}$  to root to create the planned path;
12:    while not at  $n_{goal}$  do
13:      Calculate distance and heading to the next node in path;
14:      Compute angular error relative to current orientation;
15:      Publish cmd_vel for smooth navigation towards the node;
16:      if reached node threshold then
17:        Move to the next node in the path;
18:    Store  $n_{goal}$  as visited;
19:    Repeat RRT growing from new position, targeting nodes furthest on average from visited
nodes;
20: return Task Completed
```

Summary of Pseudocode

- This RRT is grown from the robot's current position, using the occupancy grid from that is imported from `slam.py` node.
- After constructing the initial RRT and choosing a “furthest visible node” as a goal, the robot navigates to that node by following the path given by backtracking from the goal to the start.
- Because each iteration (we currently implemented 10 iterations in `random_navigation.py` as this produced a developed map of the area) re-invokes an RRT expansion from the *new* position, the robot is able to effectively explore new areas over multiple RRTs.
- The occupancy grid's `data` array is used to check collisions and define valid vs. invalid points. Cells with values ≥ 50 (on a 0–100 scale) are considered obstacles for the robot to avoid.
- A buffer distance (`SAFETY_BUFFER`) is checked along the path segments to ensure clearance from obstacles, as the robot goes over some area around the path, as it has a 2 dimensional projection on the occupancy grid.

3.1.2 Occupancy Grid Mapping Mathematics

World to Grid Conversion: The following equations describe the transformation between the robot's position in the world-frame position (x, y) and the occupancy grid indices (u, v) given some occupancy grid

with resolution r :

$$u = \left\lfloor \frac{x - x_0}{r} \right\rfloor, \quad v = \left\lfloor \frac{y - y_0}{r} \right\rfloor.$$

Algebraically, we are able to find the inverse transformation back to the global position as such:

$$x = u \cdot r + x_0, \quad y = v \cdot r + y_0.$$

Occupancy Probability: In our `slam.py` node (ironically named SLAM though it really only handles mapping), all mapping is approached by implementation of a log-odds representation, which is given for each cell (i, j) as follows:

$$\ell_{ij} = \log \left(\frac{P(\text{occupied} | i, j)}{1 - P(\text{occupied} | i, j)} \right),$$

This can then be converted to a probability of occupancy given the following formula that was covered in class:

$$P(\text{occupied} | i, j) = \frac{1}{1 + e^{-\ell_{ij}}}.$$

However, in our actual implementation of `random_navigation.py` and `slam.py`, the probabilities on the map for each given cell were scaled up to integer values going from 0 (free) to 100 (occupied). Cells ≥ 50 are considered obstacles in the path planning methods of `random_navigation.py`.

3.1.3 RRT States and Expansion

Nearest Node: The distance metric used to find the nearest node n_{nearest} to a sample $(x_{\text{rand}}, y_{\text{rand}})$ is the Euclidean distance:

$$d(n_1, n_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

State Transition and Steering: When we want to connect n_{nearest} to the sample point $(x_{\text{rand}}, y_{\text{rand}})$, we only step up to `MAX_STEP_LENGTH` (a global field in `random_navigation.py` defined as 0.5 m in our implementation). If $d(n_{\text{nearest}}, \text{sample}) \leq \text{MAX_STEP_LENGTH}$, we connect them directly. Otherwise, we place a new node:

$$x_{\text{new}} = x_{\text{nearest}} + \text{MAX_STEP_LENGTH} \cos(\theta), \quad y_{\text{new}} = y_{\text{nearest}} + \text{MAX_STEP_LENGTH} \sin(\theta),$$

where

$$\theta = \arctan(y_{\text{rand}} - y_{\text{nearest}}, x_{\text{rand}} - x_{\text{nearest}}).$$

Ensuring Robot Safety Via Buffer Zone: For a path segment from (x_1, y_1) to (x_2, y_2) , we check 20 intermediate points at equal increments between the two points. At each increment, we also offset points by $\pm \text{SAFETY_BUFFER}$ perpendicular to the path, which ensures that the robot can safely navigate through the path without collision. All those grid cells from the path to the buffer edge must be valid, i.e. not marked as an occupied cell.

3.1.4 Path Following and Transformations

Once a valid goal node is selected, we reconstruct the path by backtracking through the pointers of each of the parent nodes (This is shown in line 11 of the pseudocode). The robot then follows this path by:

$$\Delta x = x_{\text{next node}} - x_{\text{robot}}, \quad \Delta y = y_{\text{next node}} - y_{\text{robot}},$$

Given the robot's current heading θ_{robot} , we compute the error on the steering angle:

$$\Delta\theta = \theta_{\text{goal}} - \theta_{\text{robot}},$$

and control the robot accordingly with linear velocity v and angular velocity ω .

3.1.5 Cost Function and Exploration

The node performs up to 10 iterations, each time finding a RRT expanding from the node that the robot is currently at. This allows exploration of different parts of the map. Once the robot reaches the goal it selects in the RRT, the path is cleared, a new RRT is grown, and the next goal is selected based on the cost function we have implemented:

$$C(x) = -\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \|x - v\|_2$$

Where \mathcal{V} is the set of all nodes that the robot has visited and $\|x - v\|_2$ is the Euclidean distance between the candidate node x and a visited node v .

Launching and Integration. Both the `random_navigation` node and the `slam` node were launched together in the base ROS 2 launch file for HW6. The navigation node listens for:

- `/map` (the occupancy grid)
- `/scan` (the laser scan for current robot pose transforms)

and publishes velocity commands to `/cmd_vel` and visualization markers to `/visualization_marker`.

3.2 System Architecture with Dual-Robot System

Our solution to the multi-robot SLAM and exploration problem uses a mixed central-distributed design that handles the main challenges of coordinating multiple robots. Instead of making everything centralized or completely separate, we combined the best parts of both approaches to make the system work well and be easy to expand.

The system is built using ROS2, where each part runs as its own node. This makes it easier to find and fix problems, and we can add more robots later if needed. The main components of our system are:

3.2.1 Main SLAM Node

The central SLAM node performs several critical functions:

- Combines LiDAR data from both robots to create one unified map
- Uses a grid-based map where each cell has a probability of being occupied
- Handles cases where robots see the same area differently
- Maintains map consistency even with data from different positions
- Shares the updated map with all other parts of the system

3.2.2 Sensor System

The sensor integration system provides comprehensive environmental awareness:

- Utilizes 360-degree LiDAR sensors for environmental scanning
- Processes raw scan data into usable information
- Maintains continuous robot position tracking
- Combines odometry and scan matching for precise localization

3.2.3 Display and Simulation System

Our system leverages both Gazebo and RViz for different aspects of the simulation. Gazebo handles the physics simulation and robot visualization, while RViz is where the actual SLAM and data processing visualization occurs. In RViz, we create a single unified map by combining sensor data from both robots, rather than maintaining separate SLAM instances. This allows both robots to contribute their sensor data to the same map, creating a more complete and accurate representation of the environment.

The integration of our multi-robot SLAM and navigation system within the ROS 2 framework presented significant technical challenges due to the distributed nature of the system and the complexities of coordination. The system architecture consists of a network of interconnected nodes communicating through a well-defined topic and service interface hierarchy.

The SLAM node subscribes to the laser scan topics from both robots (`/robot1/scan` and `/robot2/scan`) using a quality-of-service profile configured for reliable transport to ensure no scan data is lost. It also subscribes to the transform messages (`/tf` and `/tf_static`) to maintain the spatial relationships between the robot frames. The node publishes the unified occupancy grid map on the `/map` topic using the `nav_msgs/OccupancyGrid` message type.

The transform tree is particularly complex in our multi-robot setup. Each robot has its own transform tree with the following key frames: `base_footprint` → `base_link` → `lidar_link`, with additional transforms for the wheels and other components. The SLAM node handles the transforms between the global map frame and each robot's `base_footprint` frame, effectively localizing both robots within the shared map.

The navigation nodes for each robot subscribe to the shared map topic and publish velocity commands on the respective robot's command velocity topic (`/robot1/cmd_vel` and `/robot2/cmd_vel`). These nodes also subscribe to their robot's laser scan topic to perform local obstacle avoidance.

The parameter handling in ROS 2 was utilized extensively to make the system configurable at runtime. Critical parameters such as map resolution, update frequency, RRT biasing weights, and collision avoidance parameters were exposed through the ROS 2 parameter server, allowing for dynamic reconfiguration during operation.

Inter-node communication was implemented using both standard publish-subscribe messaging for periodic data and service calls for request-response interactions. For high-frequency data such as laser scans, we utilized the DDS reliability and durability QoS settings to optimize for real-time performance while ensuring data integrity.

RViz was configured as the primary visualization tool, displaying the shared occupancy grid map, robot positions, laser scan data, and planned paths. The RViz configuration was carefully tuned to handle the visualization of multiple robots and their sensor data without performance degradation. Key display elements included the occupancy grid display for the map, TF displays for the robot frames, and marker arrays for visualizing the RRT trees and frontier points.

Gazebo served as the simulation environment, providing a realistic Mars-like terrain for the robots to explore. While Gazebo handled the physical simulation aspects, including robot dynamics, sensor simulation, and environmental modeling, all the data processing, mapping, and path planning were performed by our ROS 2 nodes with results visualized in RViz. This separation of concerns allowed us to focus on algorithm development while relying on Gazebo's robust physics engine for realistic simulation.

The system architecture was designed with scalability in mind, allowing for the addition of more robots with minimal changes to the core components. Each new robot would simply require adding the appropriate subscription topics to the SLAM node and creating an additional navigation node instance.

3.2.4 Simulated Mars Environment and Robot Models

The Mars terrain used in our simulation was created using Gazebo's heightmap functionality with texture overlays derived from actual Mars imagery. Terrain roughness was modeled through procedural generation with parameters tuned to match Martian surface characteristics based on NASA's publicly available data.

For our robotic platforms, we utilized two instances of the wheeled vehicle model from the Gazebo Model Editor tutorial . These models feature differential drive control systems with LIDAR sensors mounted on their chassis. We implemented teleop for both robots using standard ROS 2 keyboard control interfaces, allowing us to manually position the robots or intervene during autonomous operation when necessary. The

robot models were particularly suitable for our Mars exploration scenario due to their rugged chassis design and sensor placements optimized for environment scanning.

4 Contributions and Lessons Learned

4.1 Technical Contributions

The project contributed some technical advancements to the field of multi-robotic autonomous exploration. Our approach allows for one robot to communicate with a central node that transforms all robots' sensory input at once, maintaining a global unified map.

The second contribution was the RRT algorithm to better explore unknown spaces. The (we think to be) original awareness cost function that makes nodes in seemingly explored regions costly while those in "less-seen" areas more beneficial is able to bias the single robot towards mapping out an unknown space rather efficiently (close to 10 steps from repeated trials with the algorithm). This allows the robot to really search the entire map and spread itself to all corners of the area effectively and safely.

Finally, our ROS 2 integration scheme provides a foundation for multi-robot systems, addressing the challenges of namespace assignment and robot-to-robot messaging. In addition, the scheme is modular for somewhat simple implementation for added robots or minor variations in sensor configurations. This combined with the custom RRT algorithm provides an almost-complete system to effectively map out a space.

4.2 System Trade-offs

The development of our multi-robot exploration system revealed several important trade-offs that must be carefully considered in such systems. Our hybrid approach centralizes processing to ensure map consistency. While this centralization simplifies map maintenance and ensures consistency, our testing (in task manager) showed that the CPU utilization with two robots to be consistently above 80-90%, proving to be very computationally costly.

In our implementation, we struggled with exploration efficiency vs. mapping quality. For instance, many exploration algorithms depend on fast exploration of a space, not necessarily high-detail mapping. Yet if a robot moves too fast, its sensors may become compromised due to motion blur and sensor noise. We tried to alleviate this concern by allowing us to adjust robot velocity based on local environmental characteristics and uncertainty. The more uncertain or complicated a space, the slower the agents would move, providing better input from their sensors.

There's a balance of efficiency versus optimality with certain sampling-based planning algorithms — and RRT is one of them. The more quickly that RRT finds a viable path, the less optimal that path will be when compared to other planning algorithms that take the time to ensure completion. Yet for exploration, this isn't necessarily a problem. When exploring new areas, being in the moment and creating on the fly works better than switching to paths of optimality that may not accommodate subsequent discoveries down the line.

4.3 Lessons Learned

The biggest lesson we learned is to start earlier with ROS as there are many bugs and small things that there is not much documentation on that caused issues and unmotivated us slightly. However, the biggest lesson learned was building off separate laptops/builds and then attempting to combine everything, especially last second due to other difficult courses, was not as simple as just moving files over. We got some implementation to carry over but fell short with the multi-robot RRT implementation and thus it was only demonstrated on one robot. We are still proud of our final project as we were able to show working knowledge of ROS and Gazebo which we believe will be useful to use pursuing a career in robotics.

The development and testing of our multi-robot system taught us many lessons to remember when implementing multi-robot systems in the future. First, if using ROS 2, namespace control is very important for any multi-robot system. During the early testing of our system, we experienced lots of independent failures

related to topic naming collisions and transform frames, resulting in many minutes each of cumulative debugging per failed occurrence. The failures were resolved upon achieving namespace control and consistency. Each topic/frame was given a robot-specific prefix, e.g., the laser scan was /robot1/scan instead of merely /scan, complete with proper alignment across all topic names and frames. This caused us halfway through the project to redefine all the pathing.

5 Conclusions and Future Work

This project has demonstrated the effectiveness yet complexity of a unified autonomous exploration system for multi-robot applications. The approach we developed balances the trade-offs between centralized and distributed processing, exploration efficiency and map quality, and computational load versus path optimality. While we weren't able to fully integrate both ends of the system (the autonomous exploration algorithm being implemented on the two robots to map out the custom Gazebo environment). Theoretically, though, these strategies should significantly outperform single-robot approaches in terms of mapping speed, accuracy, and coverage efficiency.

For others attempting to implement a similar system, several recommendations emerge from our experience. Starting with proper namespace organization from the beginning is essential to avoid complex refactoring later in the development process. Every topic, service, transform frame, and parameter should follow a consistent naming convention that clearly associates it with the relevant robot or subsystem. Implementing comprehensive visualization tools early in the development process greatly accelerates debugging and parameter tuning. RViz configurations should be created for different debugging scenarios, such as examining transform relationships, monitoring exploration progress, and analyzing mapping accuracy. Testing should begin with simplified environments before moving to complex scenarios. Message queue sizes should be carefully tuned based on the observed communication patterns in the specific deployment scenario. Finally, critical high-frequency topics like laser scans and transform broadcasts may require larger queues.

In conclusion, our multi-robot simulation paired with the autonomous SLAM-like exploration system demonstrates the significant potential of collaborative robotics for mapping and navigating unknown environments - though we could never truly see this result :(.

Video Link Here!