Project 1 Progress Report
ROS 2 Jazzy + Gazebo + SLAM + Nav2
Setup, Mapping, Navigation, and Extensions

Kieu Son Tung

Hanoi University of Science and Technology (HUST)

February 3, 2026

Outline

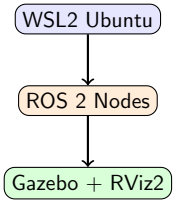
1. Environment Setup
2. Simulation and Teleoperation
3. SLAM Mapping
4. Autonomous Navigation (Nav2)
5. Programmatic Control
6. OS Resource Management (Extension)
7. Dynamic Obstacles (New Module)
8. Conclusion

1. System Environment

Platform

- Windows 11 + WSL2 (Ubuntu 24.04)
- ROS 2 Jazzy Jalisco
- Gazebo Harmonic (simulation)
- TurtleBot3 (Burger)

Goal: Build a reproducible robotics workflow for SLAM and navigation experiments.



Key Output

Working simulation + tooling for later research.

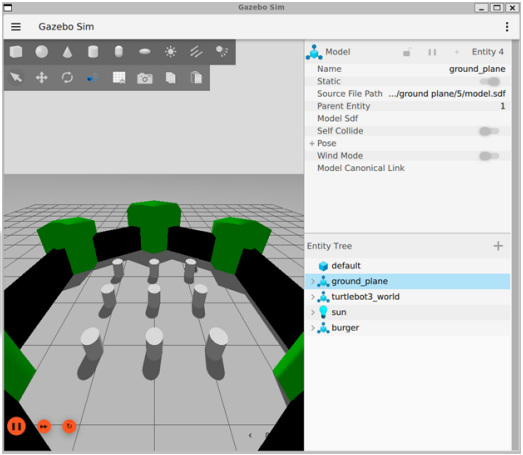
2. Gazebo Simulation

What we ran

- TurtleBot3 world in Gazebo
- Teleop keyboard to drive the robot
- Sensor topics available (e.g., `/scan`)

Result

Stable simulation environment ready for mapping and navigation.



Gazebo Harmonic simulation with TurtleBot3.

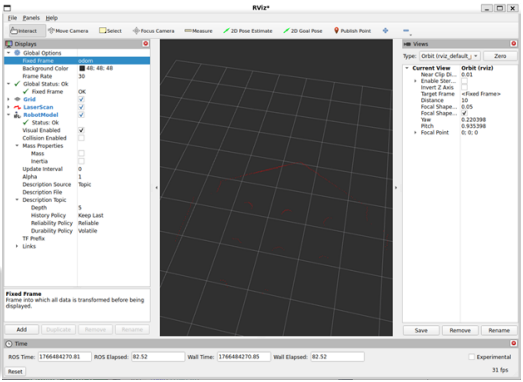
3. RViz2 Visualization

RViz configuration

- Fixed Frame: `odom` (teleop testing) / `map` (SLAM)
- Add `/scan` (LaserScan)
- Add RobotModel / TF for debugging

Why it matters

Quick feedback loop for sensor timing and frame consistency.



Laser scan and robot model in RViz2.

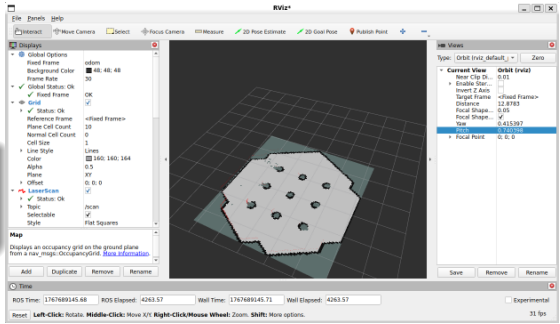
4. SLAM Toolbox (Online Async)

Mapping workflow

1. Launch SLAM Toolbox (online async) and drive the robot to explore.
2. SLAM performs scan matching and pose-graph updates.
3. Publish occupancy grid map (`/map`) for later use.

Deliverable

A complete map that can be saved and reused for Nav2 navigation.



Completed SLAM map.

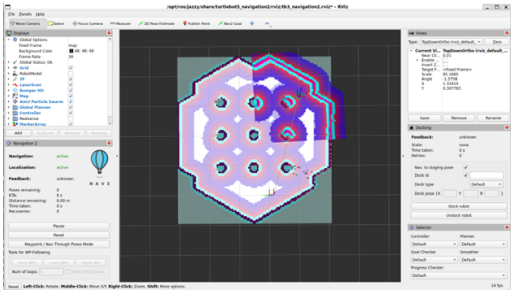
5. Nav2 Bringup on the Saved Map

Nav2 stack

- Global planner: Dijkstra/A* on costmap (e.g., NavFn)
- Local controller: DWB for reactive obstacle avoidance
- Behavior Trees for task orchestration

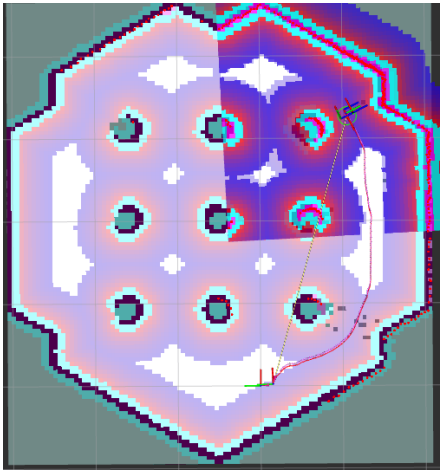
Operation in RViz2

Set 2D Pose Estimate, then send Nav2 Goal to navigate.



Nav2 initialized with the map loaded.

6. Navigation Result



The robot follows the planned path while the local planner avoids obstacles in real time.

7. Waypoint Navigation via Python

Motivation: Replace manual RViz testing with a repeatable program interface.

Approach:

- Use `nav2_simple_commander` (`BasicNavigator`)
- Send goals or follow a list of waypoints
- Monitor completion to build automation scripts

Result

The robot can patrol through predefined positions without manual intervention.

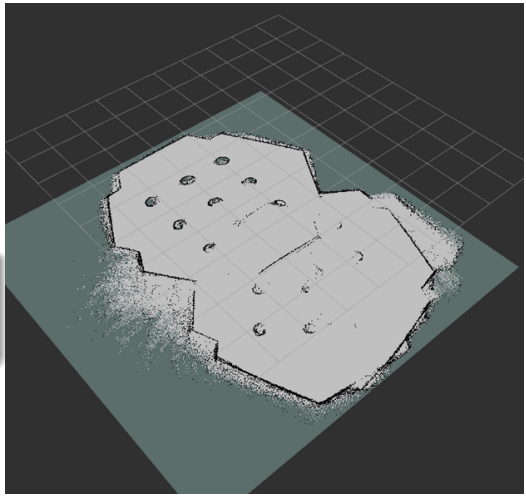
8. Why Resource Contention Matters

Problem

- ROS 2 nodes are OS processes competing for CPU time (CFS fairness).
- Under load, SLAM may process delayed scans $\rightarrow$ map drift.

Impact

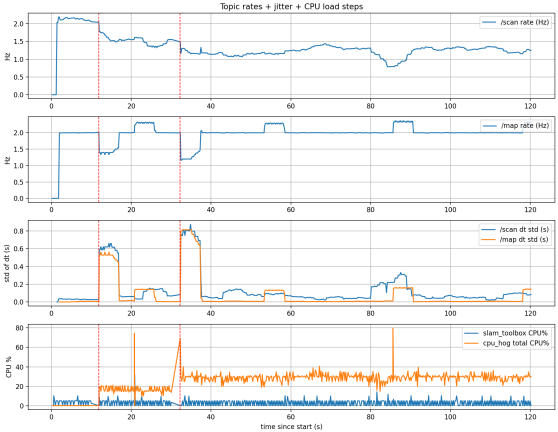
Once the map drifts, navigation quality degrades severely.



Example drift under contention.

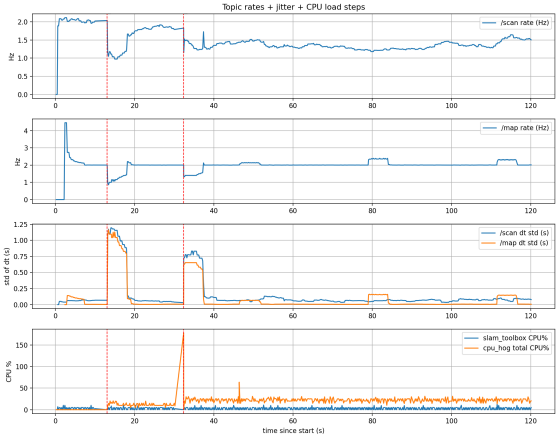
9. Bandit vs. Q-Learning (High Level)

Bandit (ϵ-greedy)



Stateless: learns a single best action on average.

Q-Learning (State-Aware)



Stateful: reacts earlier when jitter rises.

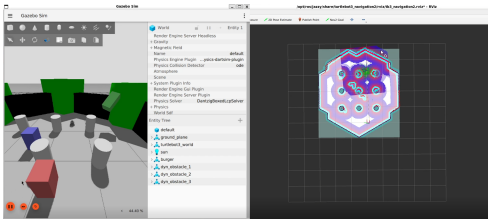
10. Dynamic Obstacles with Nav2

Objective

- Make the environment more realistic with moving obstacles.
- Test Nav2 re-planning behavior during execution.

Implementation

- Spawn and move box entities in Gazebo.
- Periodic pose updates (`ros_gz_sim set_entity_pose`).
- Integrated bringup: Gazebo + Nav2 + RViz + obstacles.



Moving obstacles during navigation.

11. Conclusion

- Completed a full simulation pipeline: Gazebo + RViz2 + SLAM mapping.
- Integrated Nav2 on the saved map for autonomous goal navigation.
- Added extensions: OS contention analysis and dynamic moving obstacles for more realistic testing.

Next Steps

Expand to more complex worlds, tune Nav2 costmaps, and evaluate policies under heavier dynamic scenes.

Thanks for listening!