ROBOTICS CLUB
IIT DHARWAD

ROBOMAPPER ODESSEY TECHNICAL REPORT

Autonomous Exploration & Mapping with Differential Drive Robot

Soumya Basuli

📝 Executive Summary

The RoboMapper Odyssey project was initiated as a modular, ROS2-based robotic platform designed to tackle autonomous mapping and navigation in simulated 3D environments. The project aimed to progress through three core milestones—starting with teleoperation-based control, advancing to manual SLAM, and culminating in full autonomous navigation.

Our focus for the current phase was the successful completion of Milestone 1, where we developed a custom mobile robot, modeled and simulated it in ROS2 and RViz2, and implemented real-time manual control. Although unforeseen technical limitations prevented us from completing Milestone 2, the foundational systems we built are scalable and directly extensible to support SLAM and autonomous navigation.

📌 Objectives

- **Design and simulate a 4-wheeled differential drive robot.**
- **Integrate ROS2 nodes for hardware abstraction, control, and visualization.**
- **Implement keyboard-based teleoperation using teleop_twist_keyboard.**
- **Validate the system in a simulated RViz2 environment.**
- **Prepare for SLAM integration by ensuring modularity and sensor placeholders.**

📐 System Architecture

Software Stack

- **Operating System:** Ubuntu 22.04 (Live Environment)
- **Middleware:** ROS2 Humble Hawksbill
- **Simulation & Visualization:** RViz2
- **Teleoperation:** teleop_twist_keyboard

- **Robot Description:** URDF via xacro
- **Version Control:** Git & GitHub

Robot Configuration

- **Chassis:** 4-wheeled mobile base
- **Drive Type:** Differential drive (simplified as 2-wheel model)
- **Control Interface:** ROS2 cmd_vel topic
- **Fixed Frame:** base_link
- **Sensors (Planned):** Placeholder for LiDAR (for SLAM and navigation)

⚙ Implementation Details

URDF Modeling

We defined a modular robot using **URDF** and **Xacro macros**, including physical links (base_link, wheel_links) and joints. Inertial and collision properties were approximated for realistic visualization in RViz2. The robot description was compiled using xacro and published through a launch file using:

- **robot_state_publisher**
- **joint_state_publisher_gui**

Teleoperation Interface

The robot was manually controlled via the **teleop_twist_keyboard** ROS2 node, which publishes velocity commands to the **/cmd_vel** topic. The following nodes were active in the launch sequence:

- **robot_state_publisher**
- **joint_state_publisher_gui**
- **rviz2**
- **teleop_twist_keyboard**

This setup allowed real-time feedback between keypress actions and simulated robot motion, providing a reactive interface to test motion control.

Launch System

The **view_robot.launch.py** file was created using the **launch** and **launch_ros** libraries. It dynamically loads the robot description and initializes all essential nodes.

🏆 Achievements in Milestone 1

1. Successfully launched and visualized the custom robot in RViz2
2. Enabled reactive motion control using keyboard inputs
3. Set up modular launch and description files using ROS2 best practices
4. Created a base architecture extensible for SLAM and navigation
5. Fully containerized ROS2 workspace with scoped packages **(my_basic_bot_description, my_basic_bot_bringup, teleop_control)**

⚔️ Challenges Faced & Solutions

Challenge	Root Cause	Resolution
xacro path error	Incorrect string concatenation using os.path	Replaced with FindPackageShare and proper substitution methods
ROS2 package not found	Workspace not sourced	Ensured correct source install/setup.bash after every build
joint_state_publisher_gui missing	Missing dependency	Installed via sudo apt install ros-humble-joint-state-publisher-gui
RViz2 missing robot model	Faulty URDF path or topic	Fixed robot_description topic and checked TF tree
System storage constraints	Live Ubuntu environment with 8 GB overlay	Attempted cleanup with apt clean, journalctl, and ROS logs but eventually blocked Milestone 2 due to lack of space

❓ Challenges Faced & Solutions

Despite completing foundational work and initiating Milestone 2 by installing **slam_toolbox**, launching **online_async_launch.py**, and preparing URDF integration for SLAM, our environment—**a live Ubuntu session with a 7.7 GB overlay filesystem (/cow)**—reached its capacity.

As Gazebo and SLAM packages such as **libprotobuf-dev**, **libignition**, and others required substantial disk space, multiple **.deb** installation operations failed due to "No space left on device" errors. Even aggressive cleanup could not free enough space due to the read-only or temporary nature of the environment.

🧗 Preparedness for Future Milestones

While Milestone 2 (manual SLAM with real-time map building) and Milestone 3 (autonomous navigation using Nav2) could not be implemented fully, the following systems are already in place:

- Fully modular ROS2 workspace architecture
- SLAM Toolbox installed and integrated (launch-ready)
- Teleoperation node and **/cmd_vel** control interface validated
- LiDAR placeholder support in URDF for future sensor plugins
- Visual confirmation of robot model in RViz2 with correct TF configuration

📜 Conclusion

Though our work was limited to Milestone 1 due to infrastructure constraints, the RoboMapper Odyssey project laid a strong technical foundation for SLAM and navigation in ROS2. The modularity of the codebase, adherence to best practices, and clear vision for scaling up ensure that, given an adequate environment, we could progress into Milestones 2 and 3 with minimal rework. Our project stands as a well-structured ROS2 implementation of a real-world robotic control system and can be rapidly extended to incorporate SLAM and autonomous behavior.