MARS-EYE

(Mars Autonomous Rover System)

JYOTHISHMATH BYRAVARUPU[1], JOSIA VARGHEESE THOMAS[2]

1. Student, Robotics And Autonomous Systems, Arizona State University
2. Student, Robotics And Autonomous Systems, Arizona State University

ABSTRACT

This project presents Mars-Eye, project focused on the development and iterative refinement of a simulation environment for a Mars rover utilizing the Robot Operating System 2 (ROS 2) Humble, Gazebo for physics-based simulation, and RViz for 3D visualization. The primary objective was to establish a stable platform capable of simulating rover kinematics and sensor feedback in a representative Mars-like terrain. The methodology involved creating a URDF model for the rover, configuring a Gazebo world, and developing ROS 2 launch files to integrate these components. Key outcomes include the successful deployment of the rover model in the Gazebo environment and the visualization of its state and transform (TF) data in RViz. Manual rover control via teleoperation was also established. Significant challenges were encountered and addressed relating to TF tree synchronization between Gazebo and RViz, ensuring consistent visualization of the rover's motion. While initial explorations into SLAM integration with RTAB-Map were made, the core effort centered on achieving a reliable foundational simulation. This work provides a basis for future development in autonomous navigation and advanced mapping.

I. INTRODUCTION

Autonomous exploration of extraterrestrial environments is a key focus in the field of planetary robotics. With challenging terrains, limited communication, and the need for real-time decision-making, robotic systems designed for planets like Mars must possess advanced capabilities in perception, mapping, and navigation. To simulate and study these systems effectively, virtual testing environments offer a safe and scalable approach.

Mars-Eye is a simulation project that emulates a Mars rover navigating and mapping Martian-like terrain using advanced robotics software tools. Built on the ROS 2 framework and simulated within Gazebo, the rover utilizes LiDAR sensors and a camera to perform Simultaneous Localization and Mapping (SLAM) using the SLAM Toolbox. The project also integrates the Nav2 stack for autonomous navigation and Teleop for manual control during testing phases. Real-time visualization is handled via RViz, allowing users to monitor the rover's position, map generation, and sensor feeds.

The primary goal of Mars-Eye is to simulate a realistic planetary exploration mission, showcasing how sensor fusion and SLAM can enable autonomous navigation in unknown environments. This project serves as a testbed for developing and validating robotic algorithms relevant to future Mars exploration missions.

PROJECT OVERVIEW

The Mars-Eye project aims to develop a simulated Mars rover capable of autonomous exploration, real-time mapping, and live visual feedback within a Mars-like terrain. The entire system is built using the Robot Operating System 2 (ROS 2) and deployed in a Gazebo simulation environment, which provides realistic physics and terrain modelling.

At the core of the project is a virtual rover equipped with a LiDAR sensor for spatial awareness and a camera for real-time visual streaming. The rover utilizes the SLAM Toolbox for Simultaneous Localization and Mapping, enabling it to build a 2D map of its surroundings as it moves through uncharted terrain. The Nav2 stack is used to facilitate autonomous navigation, including path planning, obstacle avoidance, and goal tracking. For manual testing and override, Teleop controls are integrated, allowing keyboard-based movement. Visualization and monitoring of the mapping and navigation processes are handled through RViz.

The system is designed to simulate realistic robotic behaviour 4under constraints typical of planetary exploration missions, such as limited sensor views and complex, uneven landscapes. By combining SLAM, navigation, and vision systems within a unified ROS 2 framework, Mars-Eye provides a robust platform for experimenting with autonomous exploration in space-like environments.

II. PROJECT OBJECTIVES

The central aim of this project was to construct a functional simulation environment for a Mars rover using contemporary robotics tools. The specific objectives were:

- To configure a stable simulation environment utilizing ROS 2 Humble as the middleware and Gazebo as the physics simulator.
- To define and implement a basic rover model using URDF, incorporating essential elements for simulated motion and perception (e.g., camera).
- To achieve accurate visualization of the rover's model, its coordinate frames (TF), and sensor data within RViz.
- To enable basic teleoperation of the rover within the simulated Gazebo world.
- To investigate and resolve common integration issues, particularly concerning the synchronization of state information between the simulator (Gazebo) and the visualization tools (RViz).

III. PROJECT DESCRIPTION

This report details the setup of the rover_terrain project, focusing on the simple_rover ROS 2 package. It covers the design of the rover URDF, the creation of the Gazebo simulation world, the launch configurations, and the troubleshooting steps undertaken to achieve a working simulation. The subsequent sections describe the System Architecture, Implementation Details, observed Results and Discussion including challenges, and a Conclusion with Future Work.

IV. SYSTEM ARCHITECTURE

A. Software Framework

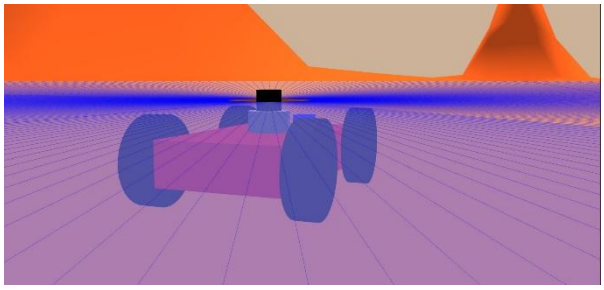
The simulation environment is built upon several key software components:

- ROS 2 Humble: Serves as the foundational middleware, providing inter-process communication, node management, and a suite of development tools. The colcon build system was used for compiling the project's ROS 2 packages.
- Gazebo (Version 11): Utilized as the 3D rigid-body physics simulator. Gazebo renders the Mars-like terrain, simulates the rover's dynamics based on its URDF, and provides interfaces for sensor data generation and actuator control via ROS 2 plugins.
- RViz2: Employed as the primary 3D visualization tool. RViz subscribes to ROS 2 topics to display the rover model, its coordinate frames (TF tree), sensor data streams (e.g., camera images), and other relevant information..

B. Rover Model

The rover was defined using the Unified Robot Description Format (URDF), likely within a file such as src/simple_rover/urdf/your_rover_model.urdf.xacro.

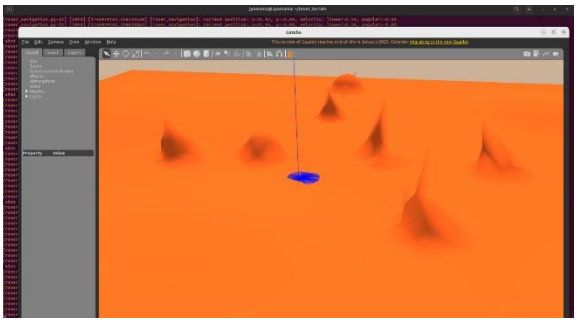
- Structure: The URDF describes the rover's physical components (links) and their connections (joints), including their dimensions, mass, and inertial properties.
- Sensors: A camera was included in the URDF, configured to provide image data within the simulation.
- Actuators/Controllers: A differential drive controller plugin (e.g., gazebo_ros_diff_drive) was specified within the URDF to interface with Gazebo, allowing the rover's wheels to be driven based on velocity commands.



(a.) Model of the Rover

C. Simulation Environment

- Gazebo World: A custom world file (e.g., located in src/simple_rover/worlds/) was used to define the Mars-like terrain, including its geometry and surface properties.
- Gazebo Models: The simulation utilized models from the global Gazebo model database and potentially custom models located in the models/ directory of the workspace or src/simple_rover/models/.



(b.) Simulation Environment

D. Key ROS2 Packages and Nodes

- simple_rover Package: The core custom package containing the rover's URDF, launch files, configuration files, and any custom scripts.
- robot_state_publisher: This node reads the rover's joint states (published by Gazebo plugins) and the URDF to compute and publish the 3D poses of the rover's links via TF transforms.
- joint_state_publisher: This node (or its GUI variant) can be used to publish joint states, primarily for visualizing the robot model in RViz when Gazebo is not running or to manually set joint positions.
- Gazebo ROS Plugins: Essential plugins like gazebo_ros_diff_drive (for rover motion) and gazebo_ros_camera (for simulated camera data) were configured in the URDF.
- teleop_twist_keyboard: This standard ROS 2 package was used to provide manual control inputs to the rover by publishing Twist messages on the /cmd_vel topic.

V. IMPLEMENTATION DETAILS

A. Workspace and Package Setup

The project was developed within a ROS 2 workspace named rover_terrain, located at ~/rover_terrain/. The primary development occurred within the simple_rover ROS 2 package, housed in ~/rover_terrain/src/simple_rover/. This package followed standard ROS 2 conventions, including CMakeLists.txt and package.xml files, and subdirectories for launch, urdf, worlds, and config.

B. Launch Configuration

The simulation was initiated using a primary launch file, for example, rover_sim.launch.py, located in src/simple_rover/launch/. This Python-based launch file was responsible for:

- Starting the Gazebo server and optionally its GUI client, loading the specified world file.
- Launching RViz2 with a predefined configuration (if available).
- Starting the robot_state_publisher node to broadcast TF transforms.
- Spawning the rover model (defined in the URDF) into the Gazebo environment using a spawner node.

C. Rover Integration in Gazebo

The rover, described by its URDF, was loaded into Gazebo at runtime. Gazebo plugins, referenced in the URDF, handled the simulation of wheel motor controllers (responding to /cmd_vel) and sensor data generation (e.g., camera image topics).

D. RViz Visualization Setup

- RobotModel Display: Showed the rover's 3D model based on the URDF and TF data.
- TF Display: Visualized the hierarchy of coordinate frames, crucial for debugging pose and orientation issues.
- Camera Display: Showed the image feed from the rover's simulated camera.

E. Control Implementation

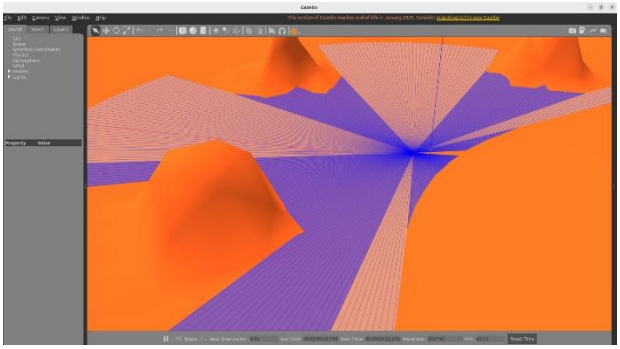
Manual control was achieved using the teleop_twist_keyboard node. This node translates keyboard presses into geometry_msgs/Twist messages, which were published to the /cmd_vel topic. The Gazebo differential drive plugin subscribed to /cmd_vel to drive the rover's wheels.

VI. RESULTS AND DISCUSSION

A. Simulation Environment Functionality

The project successfully established a simulation environment where:

- The custom Mars-like terrain and the rover model loaded correctly within the Gazebo simulator.
- The rover model responded to /cmd_vel commands issued via the teleop_twist_keyboard node, allowing for manual navigation within the Gazebo environment.



(c.) Simulation Environment function

B. Visualization in RViz

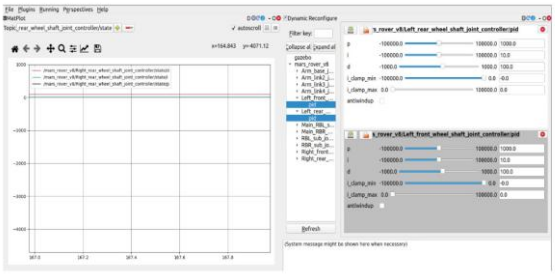
RViz was successfully configured to:

- Display the 3D model of the rover.
- Show the TF tree, representing the geometric relationships between different parts of the rover and its relation to the world/odom frame.
- Render the output from the simulated camera.

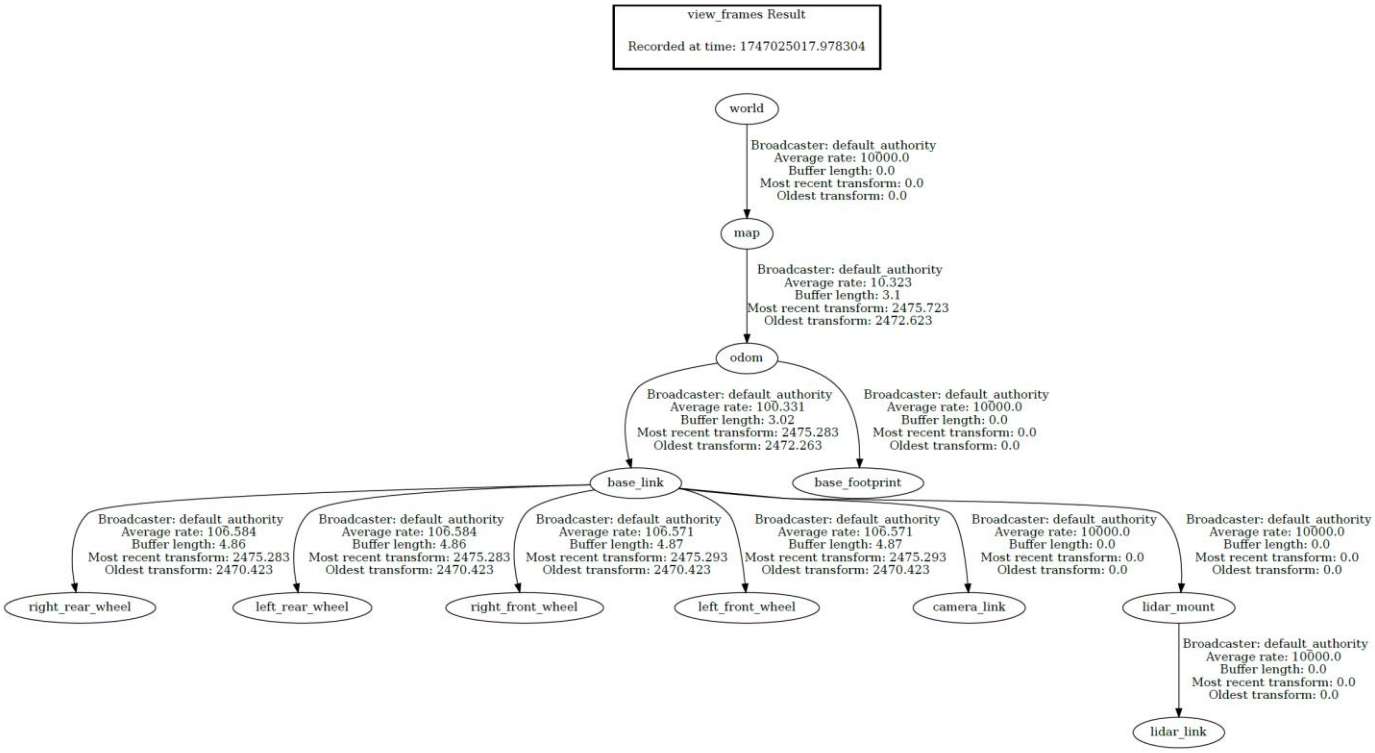
C. Challenges Encountered

Troubleshooting Several significant challenges were addressed during the project:

- Initially, we designed our own Mars rover, which was controlled using PID controllers. The wheels of the rover were jittering badly after the addition of controllers to the ROS package. Assumed PID values were causing the wheel shaft attached to the wheels to vibrate badly after spawning the model in the Gazebo world. This resulted in the model not being stationary when no command was given and the rover was at rest. This motion caused the model to move in random directions. Tuning the PID values using the **rqt** offered by ROS allowed for a live view of how the wheels reacted to changes in the Proportional, Integral, and Derivative components of the controllers. Adjusting these values decreased the vibration of the wheels and resulted in the expected behaviour of the wheels remaining stationary when no command was given.



(d)PID Controller unit



URDF Framework

- Initial Gazebo/RViz Launch Issues: Early attempts faced problems with Gazebo or RViz failing to start correctly or displaying incorrect information, necessitating checks of environment variables, ROS_DOMAIN_ID, and resource paths.

- TF Tree Integrity and Synchronization: A primary challenge was ensuring the TF tree was correctly published and synchronized between Gazebo's state and RViz's visualization. Discrepancies led to the rover appearing static or incorrectly positioned in RViz while moving in Gazebo. This involved:

    1. Ensuring robot_state_publisher was correctly configured and running.
    2. Verifying use_sim_time was consistently set to true across all relevant nodes.
    3. Debugging TF frames using ros2 run tf2_tools view_frames.py and analyzing the generated frames.pdf.
    4. Ensuring odometry information, if used as a TF source, was being published correctly by Gazebo plugins.
- Model Path and Resource Location: Ensuring all model files (meshes, worlds) were correctly pathed and accessible by both Gazebo and ROS 2 nodes was crucial and required careful organization of the models/ directory and URDF paths.

- SLAM Integration (if RTAB-Map was attempted): Initial attempts to integrate RTAB-Map highlighted further complexities related to ensuring consistent and timely data (camera images, odometry, TF) for the SLAM algorithm. Issues with data synchronization and parameter tuning were noted as areas requiring more in-depth work.

VII. CONCLUSION AND FUTURE WORK

A. Summary of Achievements

This project successfully developed a foundational Mars rover simulation environment using ROS 2 Humble, Gazebo, and RViz. The key achievements include the ability to simulate rover motion in a custom terrain, visualize the rover and its sensor data, and control the rover manually. Critical troubleshooting steps, particularly concerning TFsynchronization, were undertaken to achieve a stable simulation baseline.

B. Lessons Learned

The development process underscored the intricacies of integrating multiple complex software systems like ROS 2, Gazebo, and RViz. The paramount importance of a correctly configured and synchronized TF tree for accurate simulation and visualization was a key takeaway. Careful management of simulation time (use_sim_time) and resource paths is also essential for reliable operation.

C. Future Work

The current simulation provides a solid platform for several future enhancements:

- Autonomous Navigation: Implement the ROS 2 Navigation Stack (Nav2) for autonomous point-to-point navigation and obstacle avoidance.

- Advanced Sensing and Mapping: Integrate more sophisticated sensors like LiDAR and fully develop the SLAM capabilities (e.g., with RTAB-Map) to generate robust 3D maps of the simulated environment.