Assignment 2: ROS Tutorial

IITB Mars Rover Team - Software Subsystem

Apratim Anand

November 28, 2025

Abstract

Complete assignment report and the logic for solving it.

1 Problem Statement

Exercise 1: Rover Status

We need to create a ROS 2 package that simulates the rover's status. There are two publishers—one sends random battery level (0-100) and temperature (-20 to 80°C) data, and another sends health status (like "Healthy", "Critical", or "Warning") based on battery. A subscriber receives both and prints the data nicely. We also need to visualize everything using rqt graph and make launch files to start all nodes together

Exercise 2: Rover Odometry

We have to create another ROS 2 package for rover odometry. This involves defining a custom message with rover ID, orientation, linear velocity, and angular velocity. A publisher sends random odometry data every second, and a subscriber receives it, updates the rover's position coordinates, and prints them. We also need to warn if linear velocity goes above 3 m/s. Again, use rqt graph for visualization.

2 Solution Approach

Exercise 1: Rover Status - Solution Approach

Sensor Node:

I created a sensor node that simulates the rover's battery and temperature data. This node runs a callback function every 5 seconds that generates random values for battery level and temperature, then publishes this data to the topic "/rover/battertemp" using the Float32MultiArray message type.

Health Status Node:

Next, I made a healthstatus node that subscribes to the "/rover/battertemp" topic. When it receives data from the sensor node through the callback function, it calls another function called healthstatus() which checks if the battery level is above a certain threshold. Based on this check, it publishes a health warning (like "Healthy", "Warning", or "Critical") to the topic "/healthstatus" with the data type String.

Info Node:

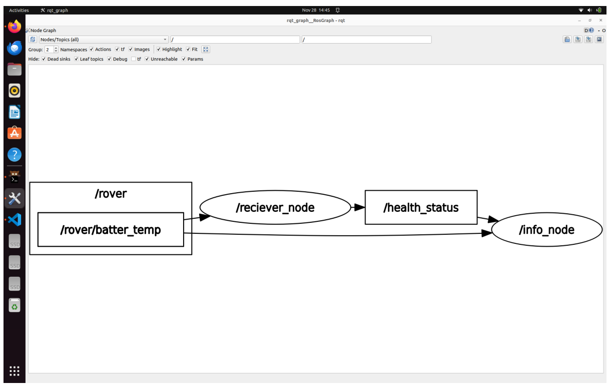

Figure 1: RQT graph

Then I created an info node that subscribes to both topics—"/rover/battertemp" and "/healthstatus". This node receives all the data and prints everything in a well-formatted way so we can see the battery, temperature, and health status all together.

Launch File:

Finally, I made a launch file that starts all three nodes (sensor, healthstatus, and info) simultaneously instead of running them manually one by one.

Exercise 2: Rover Odometry - Solution Approach

Custom Message Definition

First, I created a custom message package to define the RoverOdometry.msg file with the required data types:

- int32 roverid — unique identifier for the rover
- float32 orientation — rover's orientation in radians
- geometrymsgs/Twist linearvelocity — rover's linear speed in m/s
- float32 angularvelocity — rover's angular speed in rad/s

Velocity Node

I then made a velocity node in the "roverodometry" package that uses this custom message type. This node publishes random odometry data to the topic "/odometry data" every second. The random values are generated for linear velocity and angular velocity, while orientation can be fixed or incremented slightly.

Coordinate Node

Next, I created a coord node that subscribes to the "/odometry data" topic. In its callback function, when it receives odometry data, it updates the rover's x, y coordinates and orientation based on the linear velocity, angular velocity, and current orientation.

Additionally, the callback function checks if the linear velocity magnitude exceeds a specified threshold (like 3 m/s). If it does, it prints a warning message. Finally, it prints all the updated coordinate data along with any appropriate warning notes. Result

Both nodes work together to simulate how a rover moves and tracks its position based on velocity and orientation data, while monitoring for unsafe speeds

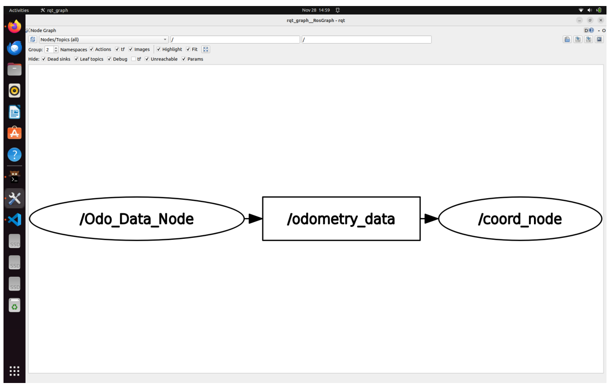

Figure 2: RQT Graph

3 Challenges and Bugs

Launch File Issues

I faced significant trouble with creating the launch file initially. The main problem was that it wasn't detecting the package properly. After spending some time debugging, I realized it was a silly mistake—I had written the package name incorrectly in the launch file. Once I fixed the spelling and naming, everything worked fine. Also faced a problem while making changes in setup file for the launch files as there were some cache files which incurred errors.

Custom Message File Errors

While setting up the custom message file RoverOdometry.msg, I ran into issues when editing the CMakeLists.txt file. Initially, I added all the custom message configuration code below the ament package() line, which caused build errors. The issue was resolved once I moved the custom message setup to the correct location in the file, above the ament package() call.

4 Key Learnings

ROS 2 File Structure

I gained a solid understanding of how ROS 2 projects are organized. Learning the proper directory structure for packages, nodes, and configuration files made it much easier to navigate and create new components without confusion.

Building Nodes from Scratch

This assignment taught me how to create ROS 2 nodes independently, from writing the node code to setting up publishers and subscribers. I learned how nodes communicate with each other through topics and how to implement callback functions to handle incoming data.

Workspace and Package Management

I understood how to set up and manage ROS 2 workspaces, create packages within them, and properly build everything using the build system.

Custom Messages and Data Types

Creating custom message types like RoverOdometry.msg showed me how to define specific data structures. I learned how to properly configure the CMakeLists.txt file to use them across different nodes.

Launch Files

I learned how to create launch files that start multiple nodes simultaneously instead of opening a new terminal everytime.

Learning Resources

Following ROS 2 documentation and YouTube tutorials significantly helped in understanding concepts and implementation. These resources provided clear examples and explanations that made the learning process smoother and faster.