GENG5511 Engineering Research Project Part 2

Mars Rover (Autonomous Software)

Luan Swart
22777124
School of Engineering, University of Western Australia

Supervisor: Prof. Thomas Bräunl
School of Engineering, University of Western Australia

Co-Supervisor: Kieran Quirke-Brown
School of Engineering, University of Western Australia

Word count: ***7966***

School of Engineering
University of Western Australia

Submitted: 14 Oct 2024

Project Summary

This build project focussed on the development of an autonomous navigation system for a prototype Mars rover which was constructed in 2023 at the University of Western Australia (UWA). The initial design of the prototype rover incorporated a basic teleoperation (remote control) mode for driving, implementing turn on the spot and side driving behaviour. This project aimed to extend that functionality by implementing additional steering controllers such as Ackermann-like controls and developing autonomous driving capabilities. The purpose of the autonomous driving capabilities is to solve a key issue faced by planetary rovers, namely the communication delay between earth and the rover which makes teleoperation a difficult and delicate operation. Modern planetary rovers all take advantage of autonomous navigation systems utilizing a variety of sensors for guidance. This project will further demonstrate UWA's growing capabilities in the areas of autonomous navigation in different environments and applications.

The aim of this project was to develop relevant software for the rover to autonomously navigate between two locations on a self-generated map whilst avoiding obstacles. The initial map of the area is created through teleoperation of the rover utilizing a simultaneous localisation and mapping package (SLAM).

This build project utilized the open-source Robot Operating System 2 (ROS2) variant called Humble Hawksbill as the core communication and control framework. ROS2 provides a number of software libraries and tools which allows for easy integration of sensors and standardised communication between software packages to implement the necessary control. This project predominantly utilizes sensors such as LIDAR, a stereo camera and an inertial measurement unit (IMU) which will be used with the ROS2 Navigation 2(Nav2) stack to achieve the autonomous navigation functionality.

The Nav2 stack is an open-source framework developed to support autonomous navigation for robotics applications. It consists of a large number of interfaces, plugins, controllers, planners and various packages which can be used to assist with autonomous navigation objectives. For the stated autonomous navigational goals of the rover, the Nav2 stack implementation utilizes visual SLAM, odometry data, LIDAR point clouds, robot transforms, various planners, controllers and behaviour servers along with rover specific base controllers within the ROS2 Nav2 ecosystem to navigate autonomously between locations.

The rover has successfully demonstrated autonomous navigation between points in a simulation environment and limited autonomous navigation in the physical setup due to both hardware and software limitations.

Acknowledgements

I would like to extend my gratitude to a number of individuals whose guidance and assistance during this project was instrumental to its success

I would like to thank my supervisors, Professor Thomas Bräunl and Kieran Quirke-Brown for their guidance and support during this project.

I would also like to thank my brother Andries Swart and friend Nico Myburgh who assisted me numerous times during experimental testing of this project. Your help was greatly appreciated, and without your assistance experimental testing would have been exceptionally difficult.

Table of Contents

List of Figures

List of Tables

Nomenclature

Acronym	Definition
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
MPPI	Model Predictive Path Integral
NASA	The National Aeronautics and Space Administration
Nav2	Navigation 2 Stack
QoS	Quality of Service
RGB-D	Red Green Blue – Depth (Camera)
ROS	Robot Operating System
RTAB-Map	Real Time Appearance Based Mapping
RViz	ROS Visualization
SLAM	Simultaneous Localisation and Mapping
URDF	Unified Robotics Description Format
UWA	The University of Western Australia
VSLAM	Visual Simultaneous Localisation and Mapping

1. Introduction and Literature Review

1.1. Introduction

Planetary rovers have been an instrumental part in human exploration efforts on Mars, specifically related to the characterization of the climate, geological properties and in the search for extra-terrestrial life [1]. Modern advances in computing technologies have allowed these exploration efforts to rapidly accelerate as it solves one of the key issues faced by these rovers which is communication lag between earth and Mars making teleoperation difficult.

In particular, advances in computing power allows new planetary rovers to be more autonomous which allows them to travel further and faster with less manual intervention. The current most advanced rover in terms of autonomous navigation is NASA's Perseverance rover which is shown in Figure 1.1. Perseverance holds the record for the longest distance covered during unassisted navigation at 759 meters [2].

Figure 1.1: NASA's Perseverance Rover in the Jet Propulsion Laboratory [3]

In 2023, as part of a final year thesis, a proof-of-concept planetary rover consisting of a 6-wheel rocker-bogie suspension and a solar panel sized chassis has been constructed at the University of Western Australia. The design of the planetary rover took inspiration from the Perseverance rover, specifically for the wheels and suspension. The design intent behind the rover was to demonstrate driving functionality of the rocker-bogie suspension over uneven terrain along with basic navigation tasks. The prototype rover as constructed in 2023, is showcased in Figure 1.2 below.

Figure 1.2: Prototype Planetary Rover [4]

At the end of 2023, only basic driving functionality of the rover has been implemented using a Raspberry Pi 4B as the control system for the rover. The original intent was to incorporate additional navigation abilities utilizing a stereo camera in conjunction with wheel encoders and a LIDAR sensor, however this was not attempted. Upon the start of this design project, the rover consisted of 6 independently adjustable steering servo's, 6 motors for driving which are controlled by 4 channels, along with 6 motor encoders. All 6 wheels can be turned independently however, the driving speed of the 6 wheels can only be controlled across 4 channels, meaning that the speed of all wheels cannot be independently adjusted as some wheels share a motor controller channel.

This project aimed to develop autonomous navigation software of the prototype rover for terrains similar to the Martian surface. The rover had major limitations in regard to its drive train, and as such, testing and developing of the autonomous driving software focussed on flat and smooth surfaces. This project's main focus was to demonstrate a real-world application of this autonomous navigation between points on a map however, a simulation environment was also be set up in order to test and demonstrate the proposed functionality of the developed code due to significant issues with reliability related to the prototype rover's drivetrain. A detailed description of the objectives of this project is given in section 1.3. This project aimed to further showcase UWA's capabilities in autonomous driving albeit in a different planetary environment, which could open new areas of research and funding for UWA.

1.2. Literature Review

The section will focus on common tools, software packages and libraries that are used in robotics applications related to autonomous navigation specifically focussing on the Robot Operating System 2 and the Navigation 2 stack along with its associated components. Early research indicated that the Navigation 2 stack was likely to be the only feasible implementation of the autonomous navigation system without expanding the intended scope multiple times over.

1.2.1. Robot Operating System 2

Robot Operating System 2 (ROS 2) is an open-source set of software libraries and tools available to assist developers with robotics applications. It acts as robotics middleware, which maintains platform independence and is especially useful to simplify communication between devices in a distributed system [5]. It possesses a number of fundamental concepts, including nodes, topics, services and messages each of which serves a different purpose.

A node is a process which performs some form of computation and can pass the results to another node via a message. Messages are strictly typed data structures and nodes can only send messages to other nodes via a topic. The messages are received or transmitted by the nodes by either publishing or subscribing to a topic. Topics are simply a string value used for reference [6]. In addition to this, a ROS service implements a request-response communication structure whereby the communication consists of two messages, one for requesting the data and the other for receiving [7]. A simple overview of the communication between publisher and subscriber nodes via a topic is presented in Figure 1.3 below. Nodes can be both publishers and subscribers at the same time and can subscribe or publish to a number of different topics simultaneously.

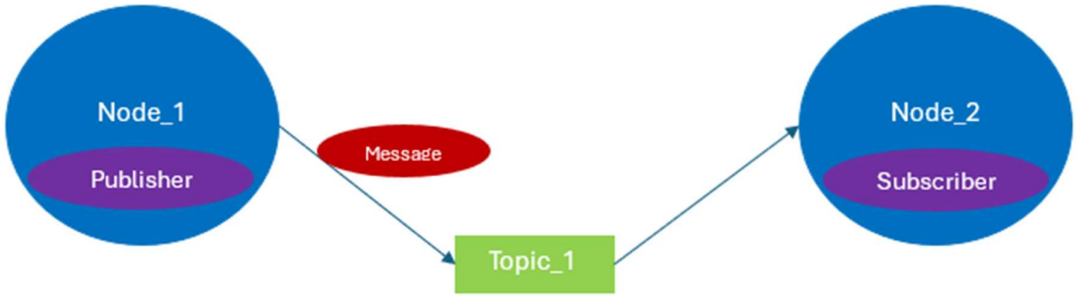


Figure 1.3: Communication between a ROS Publisher and Subscriber Node

ROS has a large and active community of loosely affiliated developers and hobbyists however the source code is maintained by Open Robotics which makes it an attractive choice for robotics applications [8]. In particular, the use of ROS for the Mars Rover project can help centralise the communication system of the rover however, running ROS increases the overhead on the controller's processing resources.

1.2.2. LIDAR

LIDAR is a commonly used method for measuring distances to objects. It is a remote sensing method which utilizes laser light pulses to measure distances from objects [9]. The distance measurement is done by measuring the two-way travel time of the emitted laser pulse also called the time of flight (ToF). Given that the laser is a form of light, the distance can be calculated in a straightforward manner as given in equation 1 below [10],

$$Distance = \frac{ToF \times Speed\ of\ light}{2} \quad (1)$$

LIDAR modules can either consist of a single layer laser or multiple layers, with a key trade-off being increasing cost as the number of layers increase. Costly, three-dimensional LIDAR sensors can be used to create high-resolution three-dimensional representations of the sensor's surrounding environment, which can be utilized for many different applications including object detection, geospatial mapping, surveying etc. According to Li et al. [11], a two-dimensional line-scan LIDAR mounted on a robotic servo can imitate the functionality of a much more expensive three-dimensional LIDAR module albeit at a reduced frequency of scan and provided that the necessary transforms and control structures are set up.

1.2.3. Stereo Camera's

A stereo camera is a camera which consists of 2 or more image sensors which imitate human vision giving it the ability to perceive depth [12]. The basic idea behind a stereo camera is to have an offset between the two or more cameras in its physical configuration with the cameras pointing in the same direction. This will result in the two images provided by the camera's having an offset between the object of interest in the frame [13]. This offset or disparity can then be used to calculate the depth of the object of interest.

Figure 1.4 below shows the basic idea behind stereo disparity. The image from the left camera detects the object of interest towards the right of the frame as shown by the triangle and the right camera detects the object of interest towards the left of the frame. As such, a

disparity in pixel overlap occurs which is presented in the combined frame of the left and right images from which the distance can then be calculated.

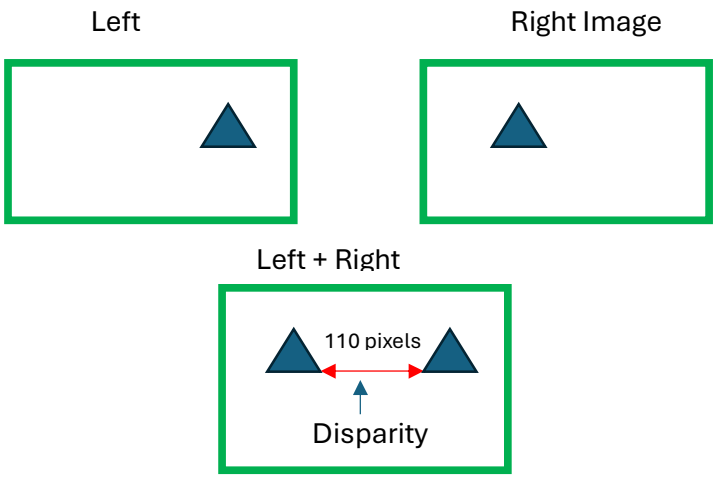


Figure 1.4: Stereo Image Disparity Example

1.2.4. Simultaneous Localisation and Mapping

Simultaneous Localisation and Mapping (SLAM) is a common method used in the area of autonomous navigation to develop a map of a robot's environment and to establish the robot's pose withing that map. SLAM allows a vehicle to build a map and simultaneously localise the vehicle within that map utilizing the robot's odometry [14]. Localisation of a robot or vehicle describes the estimation of the changes in the robot's position while the vehicle is moving. Since the pose of a vehicle cannot be directly measured, it has to be acquired and then calculated from sensor data. As such any and all errors in the original sensor data will result in the accumulation of errors in the resulting pose which is known as drift [15]. The output of a SLAM algorithm generally consists of a point cloud map, either in 2D or 3D.

Since its introduction, SLAM based approaches have evolved to substantially mitigate the effect of cumulative odometry errors resulting in drift. A popular process for correcting these errors is called loop closure. Loop closure works by returning to a known point at which the SLAM algorithm will recognize overlapping points. This allows the process to calculate any drift or tracking errors that could influence the accuracy of the localisation and make the necessary adjustments [15], [16]. Typically, these loop closure detection algorithms improve localisation by publishing rectified odometry, which in turn corrects the robot's pose estimate [17].

“Appearance- and pose-based SLAM methods offer a radically new paradigm for mapping and location estimation without the need for strong geometric landmark descriptions” [17, p. 116]. As such SLAM is of central importance to modern day autonomous driving in unknown environments, especially in areas with no GPS availability.

Modern SLAM packages can utilize a variety of sensors such as LIDAR, stereo cameras, RGB-Depth cameras and IMU sensors. An IMU sensor is commonly used in combination with a camera or LIDAR sensor. Advances in computing technology has allowed SLAM packages that uses a combination of LIDAR sensors, stereo or RGB cameras, IMU sensors and wheel odometry.

1.2.5. Visual SLAM

Visual SLAM(VSLAM) refers to SLAM algorithms which can utilize a variety of different cameras as source for pose estimation and depth perception. Common types of cameras supported by VSLAM packages include stereo, multi, wide angle, fisheye and time of flight cameras (RGB-D) [14]. One of the major drawbacks is that stereo depth perception and hence VSLAM can be compute intensive if not hardware accelerated through the use of graphics processing units [18]. In addition, VSLAM requires the surrounding environment to consists of a number of distinct and discriminative features in order to perform loop closure to accurately localise and map the surrounding environment. VSLAM approaches work well in areas with a number of obstacles, but localisation and mapping become much harder in monotonous and indistinct or repetitive areas [19], [20].

1.2.6. Real Time Appearance Based Mapping(RTAB-Map)

RTAB-Map is open-source library which initially started as an appearance-based loop closure detection method, which then grew to implement a graph-based approach to simultaneous localization and mapping and is now commonly used in robotics applications [20], [21].

This SLAM package is particularly useful for robotics applications utilizing a stereo or depth camera in ROS2 as it is one of the few packages which can implement VSLAM within ROS2. In addition to this, RTAB-Map has the ability to fuse multiple cameras and LIDAR sensors in order to generate a more accurate map and implement a more superior form of loop closure detection and benefitting from algorithmic effectiveness multimodal SLAM [22], [23]. Loop closure is the process of identifying already visited points in order to correct drift [24]. Loop closure detection in VSLAM approaches is often far superior to loop closure

detection in LIDAR based SLAM approaches in environments with rich visual imagery due to the distinct nature of reference points [25].

1.2.7. Ros2 Navigation 2 Stack (Nav2)

The ROS2 Navigation 2 stack is a highly configurable comprehensive framework which is designed to enable autonomous navigation of a robot. It consists of a large number of tools, libraries and algorithms for robots to navigate their surroundings autonomously whilst avoiding obstacles and driving to a goal pose. The Nav2 stack provides perception, control, planning, localisation and visualization in order to achieve autonomous navigation [26]. A simplified breakdown of the Nav2 stack is given in Figure 1.5 below.

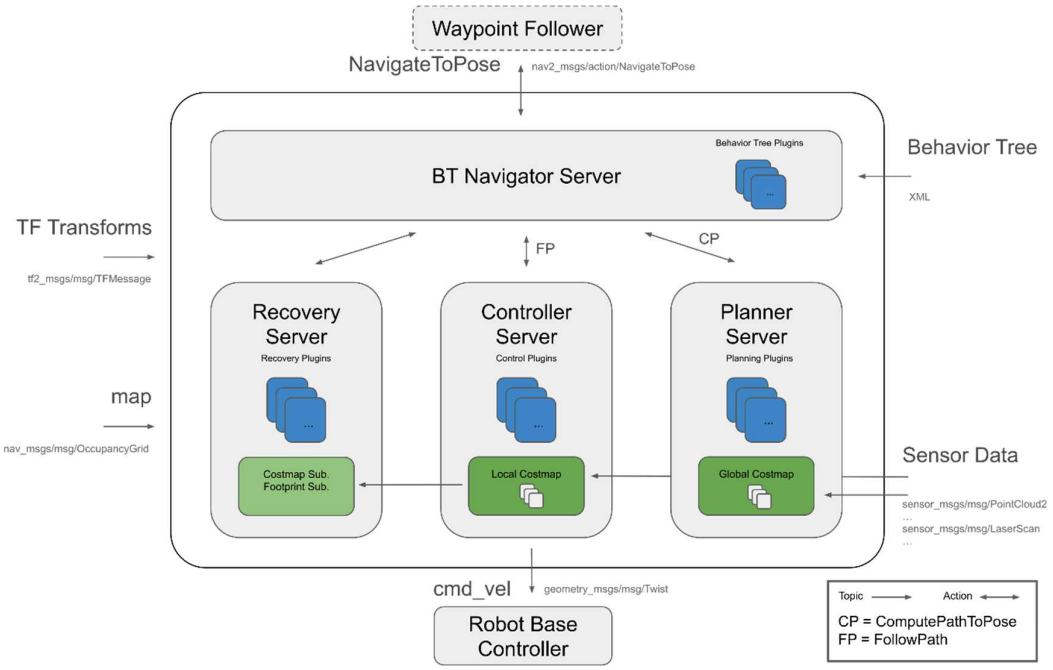


Figure 1.5: Nav2 Stack [27]

The Nav2 stack uses a number of inputs to allow for path planning and autonomous control including but not limited to,

- Point clouds from either LIDAR or stereo cameras.
- An initial map provided by a SLAM package.
- Sensor odometry data such as IMU's and encoder feedback.
- Coordinate frame transforms of the robot
- A customisable behaviour tree.
- Waypoint goal pose

It then uses a behaviour tree decision making structure in order to calculate desired routes and hence outputs a command velocity to the robot's base controller which communicates with the physical hardware to drive motors and steering [26].

1.2.7.1. Odometry

Odometry is the utilization of sensor data to estimate the position change over time of an object of interest. Accurate estimation with error correction is essential for autonomous navigation tasks, including generating a map utilizing SLAM or localising the robot within that map [28]. In the Navigation 2 stack, the odometry frame or "odom" frame is associated with the robot's odometry system and can take various sensor inputs including, wheel encoders, inertial measurement units, visual inertial odometry and LIDAR odometry. Section 1.2.7.2, below will discuss the odom frame setup in relation to the rest of the robot. Odometry is used to provide a locally accurate estimate of the robot's pose, however globally accurate information is required to correct drift and this will be provided by the map frame [29].

1.2.7.2. Transforms and Frames

In order to set up Nav2 correctly, related packages and software requires a standard coordinate frame setup for different components in the robotics system. In Nav2, the standard is REP105, which defines the relationship between the earth, map, odom and base_link frames, which is required for successful integration. The linking between frames is shown in Figure 1.6 below. The top frame is earth and then progresses down to map, odom and then base_link. The earth frame is not required for successful setup. The frame links for physical components of a robot is attached to the base_link as indicated by the ellipsis in Figure 1.6 [30].

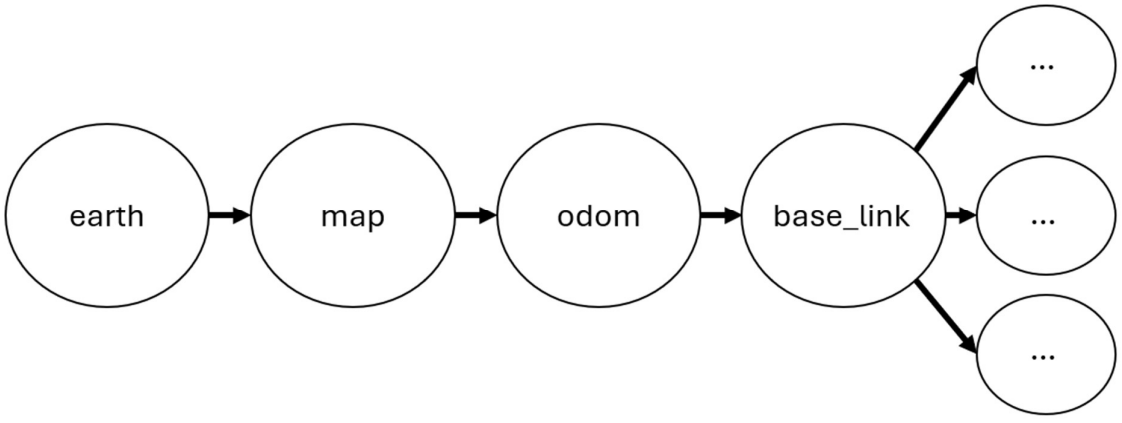


Figure 1.6: REP105 Frame Standard

A common way of providing the relationship between frames is via a unified robot description format (URDF) file which gets converted to joint states through the use of a joint state publisher [31].

1.2.7.3. Planners and controllers

A major part of the Nav2 integration relates to the setup of various planners and controllers. Of particular interest is the configuration of the Planner and Controller server which is used to specify which planner and controller plugins to use for handling path planning requests along with the method of computation for path planning. The planner and controller servers work in conjunction with one another.

Specifically, the planner server is responsible for developing a global path from the robot's current pose to the specified goal pose utilizing the known environment from the global costmap which is used to identify known static obstacles [32]. A number of different algorithms can be implemented in the planner server including the NavFn, SMAC and Theta Star planner approaches all of which are optimized to plan routes for robots of different configurations [33]. The output of the planner server is the global path.

Subsequently, the controller server takes the global path from the planner server as input and then applies the relevant velocity commands to the base controller to ensure the robot follows the planned path accurately and smoothly [34]. In addition, the controller server is responsible for implementing the local costmap and for avoiding dynamic obstacles. Specific controller servers available for use include the DWB, Regulated Pure Pursuit, Model Predictive Path Integral (MPPI) and the Rotation Shim controller, all of which is optimised for different robot configurations [35].

1.2.7.4. Costmaps

In Nav2, costmaps are developed by utilizing sensor data from the world whereby an occupancy grid map is produced based on perceived obstacles in the surrounding environment. Costmaps provide information about obstacles and free space as well as the intermediary zones which is crucial for path planning along with efficient and safe navigation of the robot. In Nav2 the package utilized to implement the costmaps is the costmap_2d package [36].

True to its name, the costmaps utilize a "cost value" approach to mapping whereby each cell of the produced occupancy grid map has a value ranging from 0-255 whereby 0 represents

free space and it gradually increases to 255 which represents a fully occupied or lethal obstacle [37]. Costmaps can be constructed from a number of layers including the static, obstacle, inflation and voxel layers. Each of the layers are responsible for a different aspect of costmap generation including aspects like static and dynamic obstacle detection, buffer zone generation and 3-dimensional voxel grid information.

1.2.7.5. Nav2 Complete Overview

An example output of a typical navigation 2 stack implementation is visualised in Figure 1.7 below. This figure visualises the global costmap, local costmap, global path and the robot's footprint. Additionally, the robot is fully localised on the map and a LIDAR sensor is visualised to show the robots field of view in its current environment.

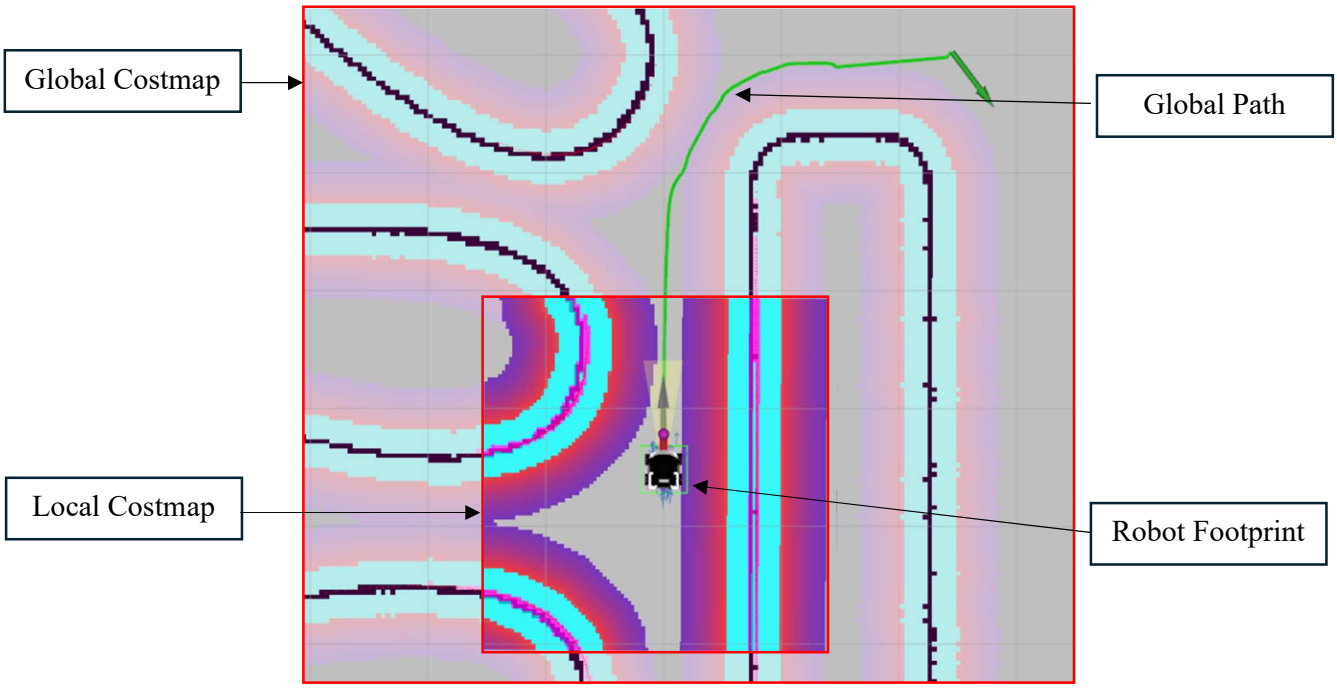


Figure 1.7: Navigation 2 Stack Example Output(Adapted From Rafal [37])

1.3. Project Objectives

Building upon the mechanical work done by the previous student, the overall goal of this project was to develop the autonomous navigation software for the prototype rover whereby the rover autonomously traverse terrain between points on a continuously self-generating map. Specifically, the rover needed to have the ability to autonomously navigate and traverse terrain by avoiding obstacles through accurate obstacle detection, localisation and path planning.

The aim for this project was to develop and incorporate the necessary sensors, controllers, control structures and software to implement the autonomous navigation as described, such that the software is fully self-contained on the rover along with an easy-to-use interface.

Consideration was given to the prototype rover's physical limitations when designing and developing the driving software as the drivetrain is plagued with mechanical issues, resulting in tolerances exceeding the initial design constraints, specifically in relation to the steering axis, drive belts and wheel alignment making it difficult to drive.

The physical inadequacies of the current rover design meant that testing was conducted in both a simulation and then limited physical environment. The step-by-step objectives of the project is summarised in Table 1.1 below.

Table 1.1: Key Aims of the Project

Aim	Description
1	Incorporate the necessary sensors for autonomous navigation
2	Experiment and develop driving controllers to suit the rover's physical construction.
3	Implement teleoperation of the rover along with an easy-to-use user interface
4	Implement a simultaneous localization and mapping package utilizing visual and/or LIDAR based technologies.
5	Implement an autonomous navigation package and set up the necessary inputs.
6	Demonstrate functionality of autonomous navigation between two points on a teleoperated generated map.

This project will lay the groundwork for continuing development of autonomous navigation systems to be utilized in non-standard environments such as off-road or extra-terrestrial environments. In recent literature, autonomous navigation has heavily focussed on robots or vehicles driving on smooth surfaces due to their applicability to modern transportation. This project will demonstrate growing feasibility of adapting autonomous navigation systems for approaches to different environments and ultimately demonstrate feasibility of increasingly autonomous navigation on extra-terrestrial environments.

2. Design Approach

The first task for the design approach was to develop and define the requirements of the system to ensure the system will be feasible for the proposed goal of autonomous navigation as described in section 1.3. Based on early research, some key requirements and constraints of the proposed autonomous navigation system were identified, and these are listed below in section 2.1 to 2.2. Some of the criteria is dependent on the initial decision to implement the autonomous navigation system in the ROS2 Nav2 ecosystem due to the lack of viable alternatives without significantly expanding the scope of this project.

2.1. Hardware Requirements/Constraints

In terms of hardware requirements, the system will need to have the ability to incorporate or maintain the following functionality.

1. Maintain the existing configuration of 6 independently steered wheels (6 steering channels and 4 motor speed control channels).
2. Have the ability to incorporate a LIDAR sensor and stereo camera.
3. Have the necessary compute capacity to implement SLAM utilizing a stereo camera and/or a LIDAR sensor.
4. Have the ability to run ROS2 Humble with no detrimental effects on performance of the controller due to computing power.
5. Have the ability to incorporate additional hardware such as a robotic arm and/or additional LIDAR sensors or stereo cameras for future development.
6. Remain within budgetary constraints of $500 total.

2.2. Software Requirements/Constraints

The software requirements and constraints of the design consists of the following items:

1. Utilize open-source software.
2. Incorporate a method to drive the rover using teleoperation.
3. Dedicated robot base controller(s) which communicates with the motor and servo controllers of the rover.
4. Implement an easy-to-use user interface including a wireless interface for control
5. Develop a URDF to model the physical characteristics of the rover which provides joint states.

6. Utilize a 2D LIDAR and/or depth camera to detect obstacles and generate a map via SLAM.
7. Localise the rover within the generated map
8. Plan a path to a received goal while avoiding obstacles on the developed map.
9. Autonomously drive the rover to the received goal on the planned path

2.3. Design Approach

The design process is divided into hardware and software development phases. Since the software development is contingent on hardware selection, the required hardware components had to be identified and integrated before software for the physical rover could be configured. A simulated environment was set up to assist with development of the configuration files and for Nav2. It was also identified that the simulation setup could be used as a mitigation against mechanical issues experienced by the physical rover.

The breakdown of the steps of the project's design and development is presented in Figure 2.1 below. The development of the complete system first started with the hardware component selection and integration, followed by the development of the hardware/software interfaces between controllers and then the autonomous driving and navigation code as well as the user interface setup.

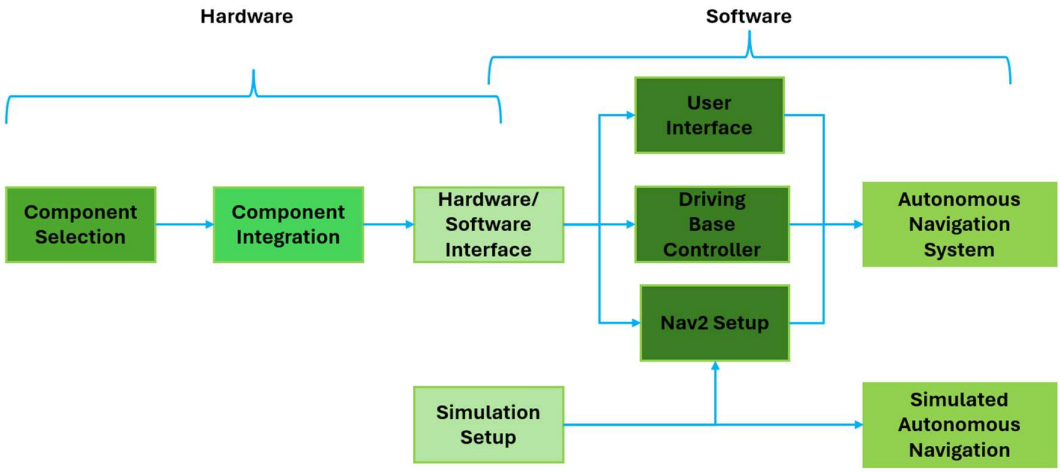


Figure 2.1: Overview of Design Steps

2.4. Relevant Standards

Key design constraints and standards identified and employed during this project all relates to the ROS2 framework. ROS2 utilizes a variety of standards in order for the system to

effectively communicate between its nodes, topics and services. For this project, the key standards were as follows:

REP103 - Standard Units of Measure and Coordinate Conventions

This standard describes the base units for measurable quantities such as length, mass, time, current, velocity etc. In addition to this, it also specifies the coordinate frames of commonly used topics such as cmd_vel. For example, the cmd_vel topic requires the axis orientation in relation to a body to be set up such that [38]:

- x represents forwards
- y represents left
- and z represents upwards.

The axis tree is shown in Figure 2.2 below and the design will adhere to the base units of measure and base directions.

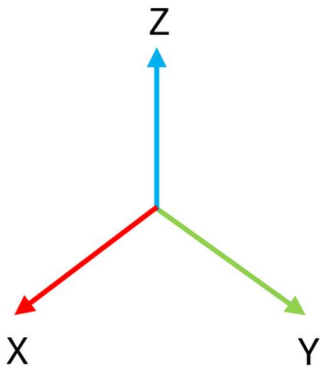


Figure 2.2: Axis Tree of REP103

REP105 - Coordinate Frames for Mobile Platforms

This standard is applicable to the coordinate frames and linking of frames and joints of the model of the physical rover within ROS2. Key elements from the standard were described in section 1.2.7.2 and the key component which is useful for this specific design is the linking of the model's frame from,

earth->map->odom->base_link->…

It should be noted that the base_footprint link can also exist in between the odom and base_link. Both REP103 and REP105 was identified as being key to ensure correct setup of interfaces between different components of the system such that it functions as intended.

2.5. Criteria for Evaluation and Experimental Test

Based on the mentioned requirements and constraints along with the project objectives, the successful demonstration of a number of key elements via experimental tests will validate the success of this project as presented in Table 2.1 below.

Table 2.1: Criteria for Evaluation and Experimental Tests

No	Criteria for Evaluation	Experimental Test
1	Teleoperation of the Rover in ROS2	Drive the rover using a wireless joystick on a flat surface either indoor or outdoor, demonstrating linear and angular motion
2	Automatic startup of all components including the controller after a pushbutton press.	Demonstrate the startup of the controller including booting to the home screen when the power switch is pressed
3	Implementation of a touchscreen user interface for control and visualization	Test the launch application and live visualisation on the touchscreen after running the controllers.
4	A successful SLAM generated map of the rover's environment with the rover localised on the map.	Teleoperate the rover whilst running the necessary sensor interfaces and SLAM package in both simulation and a smooth physical environment. The output should be an occupancy grid map and a localised rover.
5	Demonstrated capability of path planning avoiding obstacles.	After a SLAM map has been generated, visualise the global path in RViz and send a goal pose waypoint to NAV2. The calculated path should show in the visualisation and avoid obstacles.
6	Autonomously navigate between two points on a map	Once a SLAM map with the rover localised has been generated and the global path has been calculated, observe the commands to the robot base controller along with visually confirming the rover navigates between points.

3. Results

The results will first cover the final hardware and software design, followed by simulation and then physical testing results. Although simulation was not part of the initial scope and objectives, it was introduced due to early mechanical failures to allow for safer and easier code development and testing.

3.1. Hardware Design

The majority of the hardware for this rover, including the LIDAR and Stereo/Depth camera were selected before the commencement of the project. The only relevant component selection was the central controller, however a brief overview of the critical components are provided in section 3.1.1 below.

3.1.1. Component Selection

Central Controller

Based on the design requirements, it was determined that the initial controller setup utilizing a Raspberry Pi 4B as the main controller will not have sufficient compute power to run ROS2 Humble and compute intensive VSLAM without significant drawbacks [18], [39]. To eliminate this constraint and ensure future expandability, the Nvidia Jetson AGX Xavier development kit, was utilized due to its availability at no cost and increased compute capacity. The Jetson boasts 32TeraOps of computing power with an 8 core ARM CPU and a 512 core Volta GPU and 16GB of RAM, far exceeding the compute power of the Raspberry Pi 4B [40]. The controller is shown in Figure 3.1 below.

Figure 3.1: Nvidia Jetson AGX Xavier Development Kit [40]

Depth Perception

Both the LIDAR sensor and stereo camera were preselected for this project and include the 2D Richbeam Lakibeam 1s LIDAR and the Luxonis Oak-D S2 camera, which is shown in Figure 3.2 below.

Figure 3.2: Richbeam Lakibeam 1S Lidar (left) [41], and the Luxonis Oak-D S2 Camera(right) [42]

The LIDAR sensor has a range of 0-15m at a field of view of 270° and sampling rate of 10-20Hz [41]. The Oak-D S2 camera has both a 1MP, WXGA stereo camera pair and a 4K 12MP RGB Depth(RGB-D) Camera operating at refresh rates of up to 60FPS for the 4K RGB sensor and 120FPS for the stereo camera pair.

The remaining components remain unchanged from the original design and is presented in Appendix A.

3.1.2. Component Layout and Integration

A simplified layout of the electronic components is presented in Figure 3.3 below. The lines represent control connections between modules however, power connections are omitted. Some devices receive power over USB from the Jetson, but the majority have dedicated power connections which is fed from an onboard fuse box which is supplied by a large 12.8V 2.5Ah lithium battery. The layout was redesigned to accommodate the addition of the LIDAR sensors and stereo camera's as well as the integration of the Jetson controller.

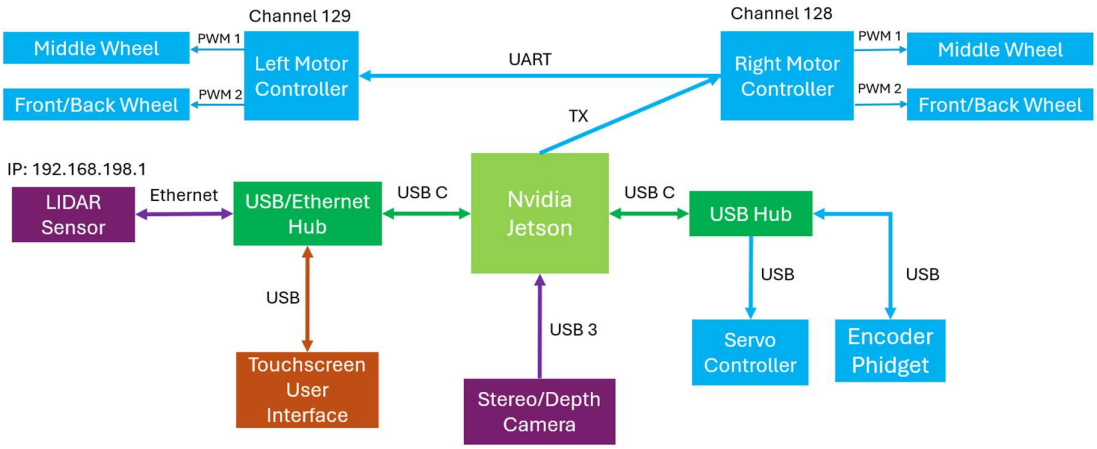


Figure 3.3: Controller and Sensor Integration

The left and right motor controllers are sharing a single TX line over UART with separate channels for each side of the rover. Each motor controller contains 2 PWM output channels resulting in a total of 4 linear velocity variables for the 6 wheels of the rover. As such, the front and back wheels of the rover on each side, share a channel and the middle wheels have their own channel for control. The motor controllers are responsible for the linear velocity of the rover. In contrast to this, there are 6 independently controllable servo's which are used for steering. The 6 servos communicate to a single servo controller which in turn communicates over USB to the Jetson.

The Depth Camera communicates over USB and auto negotiates a port. The LIDAR sensor connects over ethernet to a USB hub which in turn connects to the Jetson controller via USB. A static IP address for the LIDAR is set for it to interface correctly with its driver. The ports and software interfaces are discussed in section 3.1.2 below. Both the LIDAR and Oak-D S2 camera are mounted on the top front of the rover and the exact location is specified within the URDF of the rover to ensure the joint states are set up correctly.

3.1.3. Hardware/Software Interface

A breakdown of the hardware and software interfaces along with relevant ports for the final design is presented in Table 3.1 below. Some interfaces required permission changes within the udev rules of the operating system.

Table 3.1: Hardware and Software Interfaces

Component	Interface	Port/Adress
Left Motor Controller	Serial over UART	Chanell 128 on/ dev/ttyTHS0
Right Motor Controller	Serial over UART	Chanell 129 on/ dev/ttyTHS0
Servo Controller	USB	/dev/ttyACM0
LIDAR	Ethernet	192.168.198.1
Stereo/Depth Camera	USB	Auto Negotiate
Encoder Phidget	USB	Auto Negotiate

3.2. Software Design

The software design is built on the ROS2 Nav2 ecosystem which has been extensively explained in section 1.2. This was implemented on the Jetson controller running the Ubuntu based Nvidia Jetpack 5.12 operating system. The general design of the autonomous navigation software consists of a SLAM package, the Navigation 2 stack, Robot Base Controllers, component drivers and various smaller elements to assist with correct setup of the Navigation 2 stack

3.2.1. Autonomous Navigation Software Overview

The developed autonomous navigation code integrates pre-built packages and drivers with the ROS2 Nav2 ecosystem along with developed base controllers and configuration files. The design involved configuring correct topics and nodes for input into the Nav2 stack based on the sensor configurations and data from the physical rover. The LIDAR and Depth Camera outputs are used by linking manufacturer's ROS2 drivers with the necessary SLAM and Nav2 packages. Other sensors and controllers such as the joystick and robot base controllers are also set up to interface with Nav2. The robot base controller responsible for the motor and servo controllers were developed from scratch for ROS2 compatibility.

A number of software packages and manufacturer drivers were identified and incorporated into the final control structure. The software packages and usage are presented in Table 3.2 below.

Table 3.2: Software packages

Software Package	Use Case
Lakibeam1-ROS2	2D LIDAR driver for ROS2 which is used to assist with costmap generation.
Luxonis DephtAI-ROS-Driver	OAK-D S2 camera driver for ROS2 which provides depth cloud data and configurations for VSLAM
RTAB-Map	ROS2 SLAM Package which implements VSLAM based on the inputs from the Depth/Stereo camera.
Joy	ROS2 Joystick Driver for Linux being used for teleportation
Nav2	Core package for autonomous navigation responsible for implementing the planners, controllers, costmaps, lifecycle managers etc, based on the inputs from other packages and topics.
Joint_state_publisher	Parses the URDF rover file to set up the joint states for Nav2

A high-level overview of the interconnection of the abovementioned packages and drivers are shown in Figure 3.4 below. This simplified diagram showcases the connections between the main components of the developed autonomous navigation code within ROS2. A more detailed discussion and interconnections and specific packages will be presented for critical components.

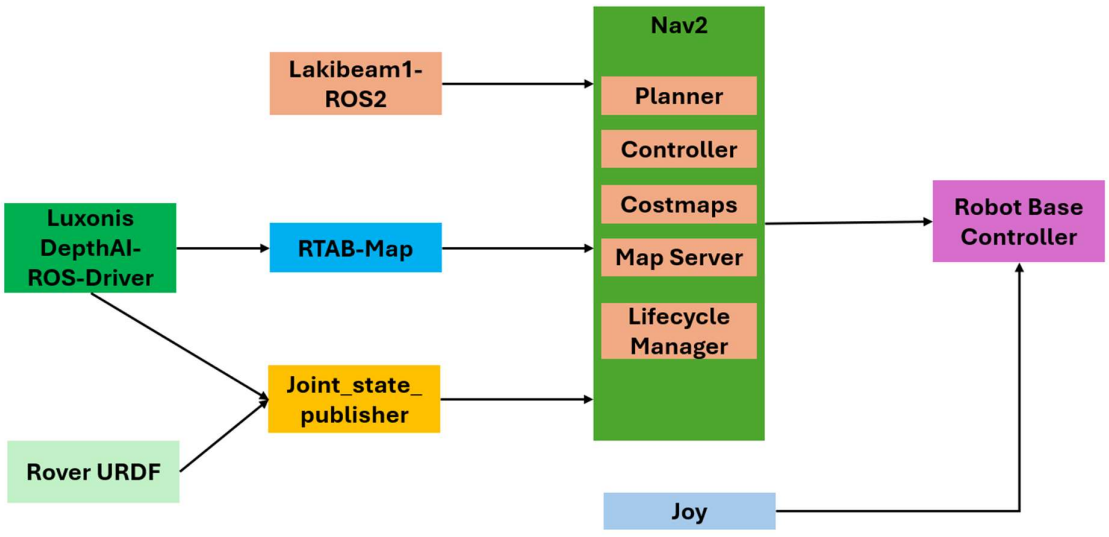


Figure 3.4: High level Interconnection Between Packages and Drivers

3.2.1.1. Robot Base Controller

The robot base controller drives the rover by sending commands to the wheel motors and steering servos. It takes linear and angular command velocities as inputs and implements appropriate controls based on the kinematics of the rover. The steering implementation is an adapted Ackermann-like, swerve driving controller as indicated in Figure 3.5 below.

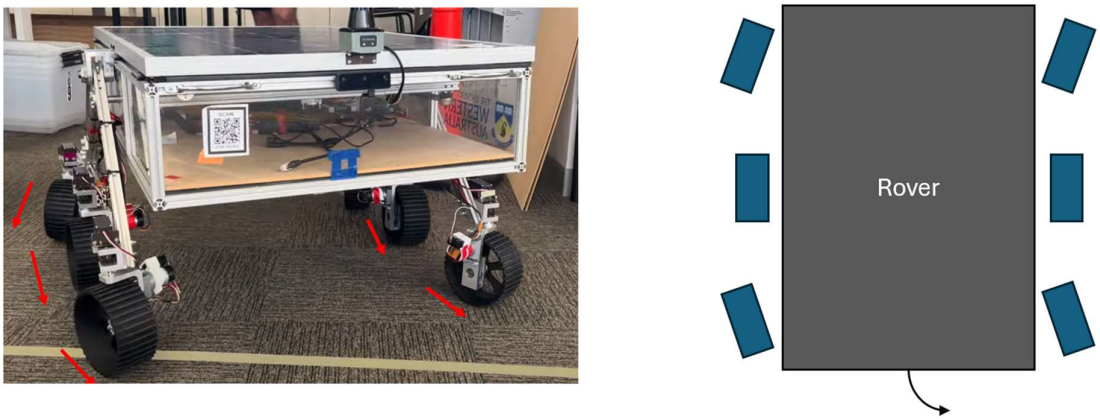


Figure 3.5: Ackermann-like Swerve Driving Implementation

Evidently, the front and back wheel servo's steer the wheels in opposite directions albeit at the same absolute angle to implement the turning behaviour. In addition to this, the outer and inner wheels travel at different speeds to implement a velocity differential whilst cornering. The speed for the inner and outer wheels during turning are fixed according to the relationship presented in equations 2 and 3 below,

$$v_{inner} = v \times \left(1 - \frac{L_{wheelbase} \times W_{wheelbase}}{2}\right) \quad (2)$$

$$v_{outer} = v \times \left(1 - \frac{L_{wheelbase} \times W_{wheelbase}}{2}\right) \quad (3)$$

Where $L_{wheelbase}$ and $W_{wheelbase}$ is the length of the rover's wheelbase and v is fixed at approximately 20% of the PWM maximum for the motors during turning to avoid mechanical failures and to allow for VSLAM computations to complete.

This steering setup is implemented using the motor_controller and servo_controller nodes as indicated in the Figure 3.6 below (nodes are indicated as ovals and topics as rectangles). The controllers receive linear and angular velocities based on the REP103 standard and converts it to the necessary PWM or position signals. The velocity commands can come from either

Nav2 (cmd_vel) or the joystick (cmd_vel_joy) and is first processed by cmd_vel_switch node before being sent to the motor and servo controllers. The cmd_vel_switch prioritises the joystick input and will override the Nav2 commands when manual control is taken via the joystick, acting as a safety measure.

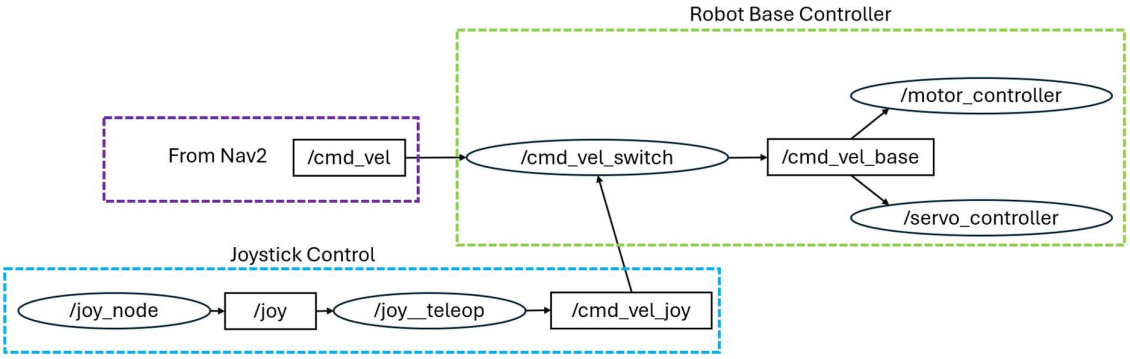


Figure 3.6: Robot Base Controller and Command Velocities

3.2.1.2. SLAM Setup

SLAM for the rover is provided by the RTAB-Map package, utilizing the depth and stereo camera on the rover. A simplified overview of topic and node connections are showcased in Figure 3.7 below with the nodes indicated as ovals and the topics as rectangles. The camera driver creates the oak node and based on the configuration and launch files, publishes the necessary time synced image and camera information topics. These topics include the camera info, rectified depth image and the raw stereo images. Once these topics are published, the RTAB-Map package subscribes to the /oak camera container, the odometry and the IMU data and computes the occupancy grid map.

In addition to the produced map, another node within the RTAB-Map package called, rgbd_odometry produces rectified odometry based on visual inertial odometry and loop closure detection. Lastly, the rtabmap_viz node is purely used for visualization and to adjust settings during experimentation.

The output of the RTAB-Map VSLAM setup is the occupancy grid map and the rectified odometry.

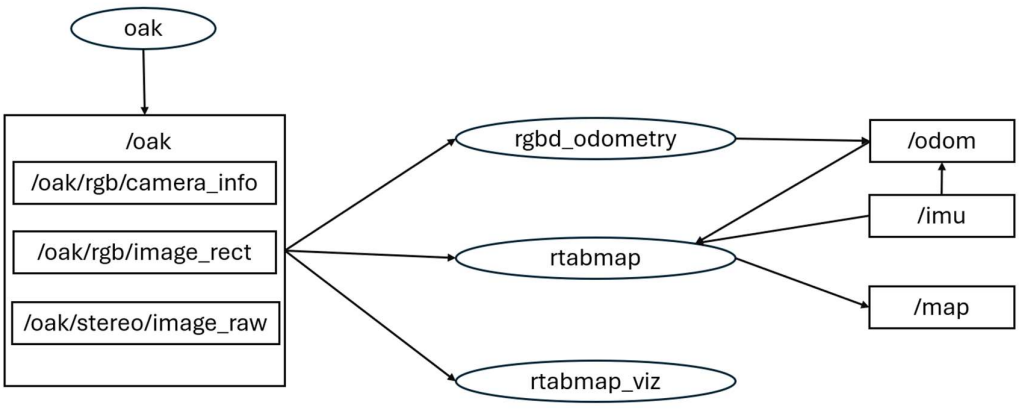


Figure 3.7: Simplified overview of Node-Topic connections for SLAM

3.2.1.3. Navigation 2 Setup

The Navigation 2 stack was launched utilizing the nav2_bringup package, which launches all the necessary controllers, planners, servers and lifecycle managers associated with a standard Nav2 setup. The bringup package utilizes the navigation.yaml parameter file, which sets characteristics for the controllers, planners, servers, smoothers, managers and costmaps for successful path planning and navigation in an environment, whilst taking into account the rover's geometry, move base and input topics. The key components of Nav2 and their associated plugins and subscriptions are indicated in Table 3.3 below.

Table 3.3: Nav2 Component Plugins

Component	Plugins	Subscribes to
Map Server	map_server	/map from RTAB-Map
Planner Server	Grid_based: NavFn_Planner	-
Controller Server	FollowPath: DWBLocalPlanner	-
Local Costmap	InflationLayer, VoxelLayer	/lidar from the Lakibeam package
Global Costmap	ObstacleLayer, StaticLayer, InflationLayer	/map from RTAB-Map /lidar from Lakibeam

A simplified black hole schematic is shown in Figure 3.8 below to showcase the relevant inputs and outputs of the complete Nav2 setup as implemented. It receives input data from

the map, odometry, lidar, tf and goal_pose topics via the packages shown in Figure 3.4. Nav 2 then computes the necessary path and generates the command velocities on the cmd_vel topic which is sent to the robot base controllers.

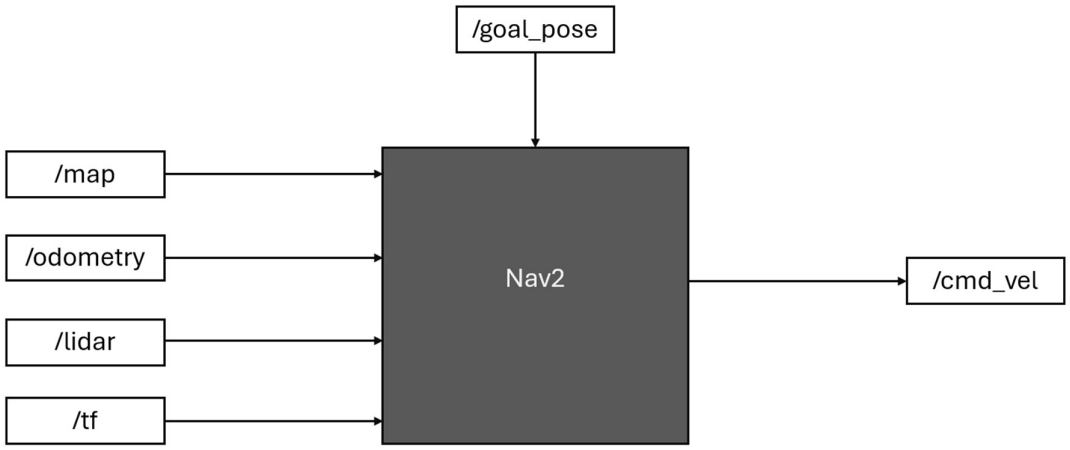


Figure 3.8: Nav2 Inputs and Outputs

3.2.1.4. URDF Design

A URDF model was developed to assist with the creation of joint states of the rover and for use in simulation. The URDF was created in XML format following the REP105 standard, defining frame links and plugins for the simulation environment. The created URDF model consists of 46 linked frames to replicate the rover's physical design however a highly simplified model of the linked frames is shown in Figure 3.9 below due to its size. The full transform tree is shown in Appendix B. Direct links are indicated in a solid black line and indirect links (i.e. links with excluded frames) are indicated in a dotted line.

The Gazebo plugins specified within the URDF provide the necessary sensors interfaces for the simulation environment which will be discussed as part of the simulation in section 3.4. When the URDF is used on the physical rover, these additional plugins are ignored during the parsing stage of the URDF.

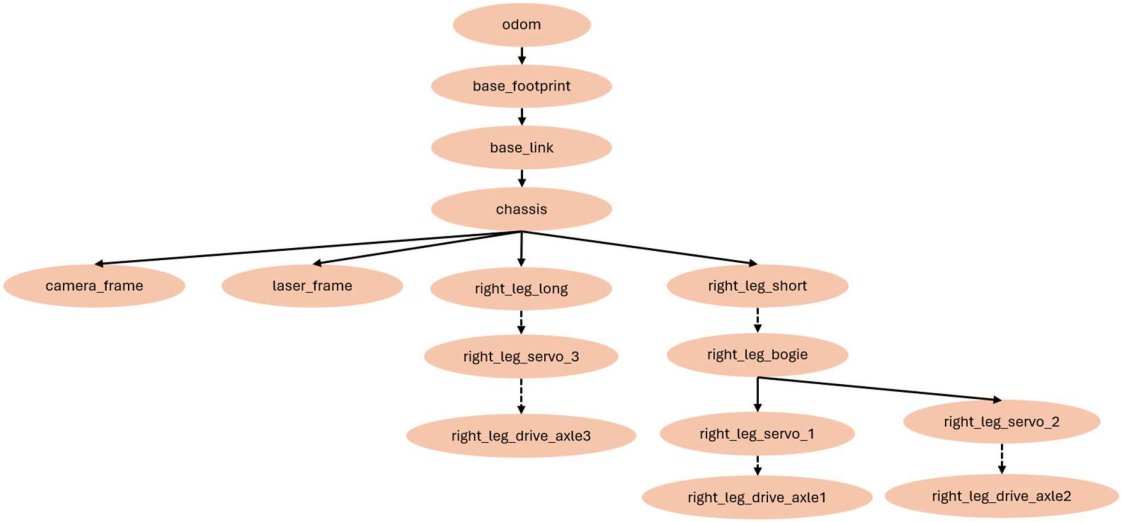


Figure 3.9: Simplified Coordinate Frames of the Rover

The resultant model of the rover can then be visualized in both Ignition Gazebo and RViz. The created model including the simulated sensors such as the camera(shown in green) and the LIDAR sensor (shown in black on the top of the rover), is shown Figure 3.10 below.

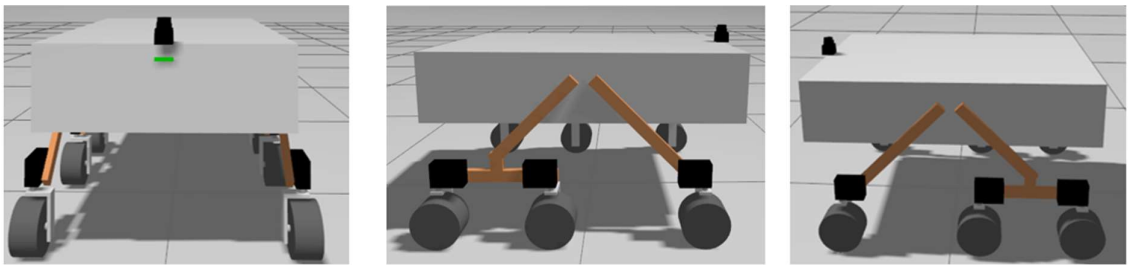


Figure 3.10: Front, Left and Right View of the Rover Model in Gazebo

3.3. User Interface

Part of the requirements was to incorporate an easy-to-use user interface to allow for remote access, control and visualisation of data and is shown in Figure 3.11 below. A VNC server was set up on the Jetson to allow for remote control and live visualisation during testing over Wi-Fi. In addition, a touchscreen was added which visualises the data on-board and is also used to start the controllers. Once the rover is turned on, a simple double tap of the icon indicated in red on the touchscreen will launch a bash script which launches the relevant robot base controller for teleoperation.

Figure 3.11: User Interface and Visualisation

3.4. Simulation

Utilizing Ignition Gazebo, a simulation environment and model of the code and rover has been developed for testing and refining of configuration files.

Basic simulation worlds with a variety of obstacles were created. The simulation setup incorporated sensor plugins for the rover including the gpu_lidar, the camera and the imu, all of which are set up to approximately match the real rover. In addition to the sensors, the Gazebo AckermannSteering plugin was utilized for driving due to the limited availability of driving plugins. This AckermannSteering controller can only steer 2 wheels instead of 4 on the physical rover [43].

The sensor data had to be bridged to ROS2 and as such a ROS-Gazebo bridge node was set up to publish the sensor data on the correct topics which is shown in Figure 3.12.

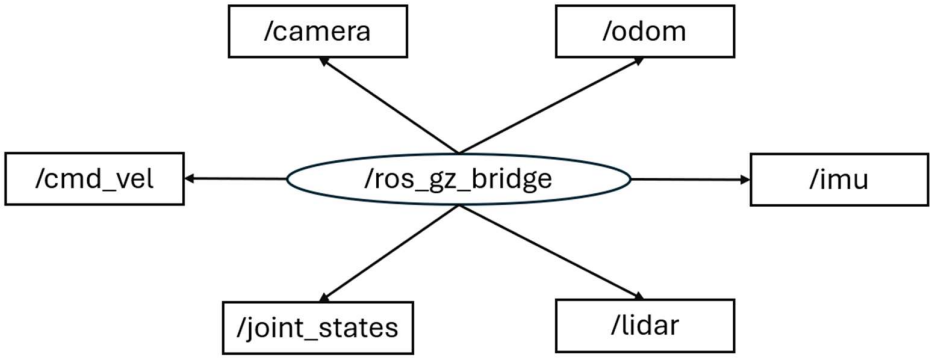


Figure 3.12: ROS-Gazebo bridge

3.4.1. Simulation Results

Several different tests were conducted in the simulation environment during the development of the complete code. First and foremost, the driving plugin setup was tested to ensure the rover is manoeuvrable followed by SLAM tests and then testing of the autonomous navigation using Nav2.

3.4.1.1. SLAM Results

The purpose of the SLAM setup and testing is to provide the map for the system with robot localized within. RTAB-Map was used in the simulation environment to generate maps using teleoperation of the rover.

Results of a successful SLAM test is shown in Figure 3.13 below. The simulated environment is shown on the left, the unexplored environment upon startup in the middle and the fully explored environment on the right. Light grey represents open space, green represents unexplored regions and black represents obstacles.

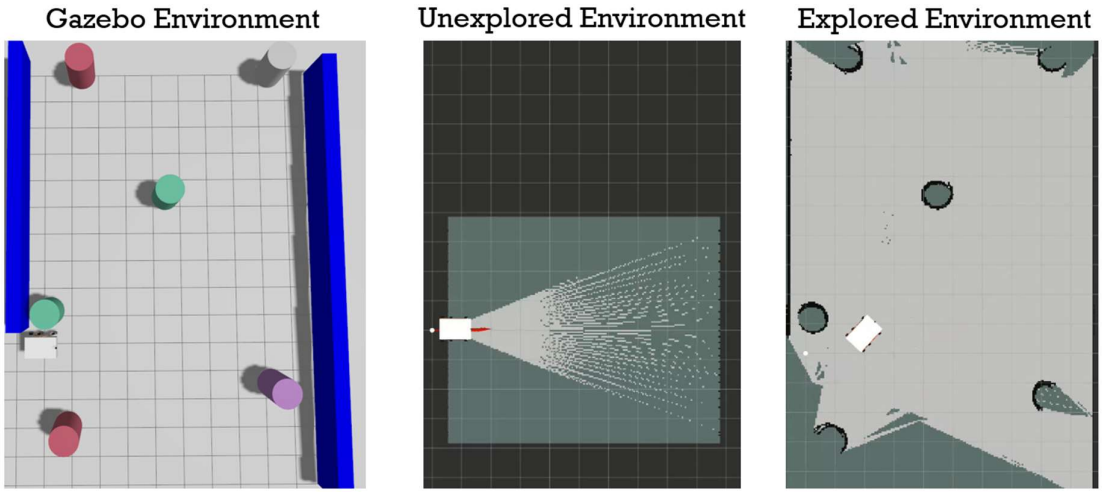


Figure 3.13: Simulated SLAM map

It should be noted that the SLAM packages only work well in object dense environments whereby the loop closure implementation can assist with correcting odometry. If the odometry drift is significant before loop closure can correct it, there is a possibility of breaking localisation of the rover within the map.

An example of excessive drift, causing a break in localisation is shown in Figure 3.14 below. To mitigate against significant drift and inadequate localisation, an EKF localiser was added

to assist with localisation based on the /lidar topic scans and the /imu topic data. This showed an improvement in overall localisation.

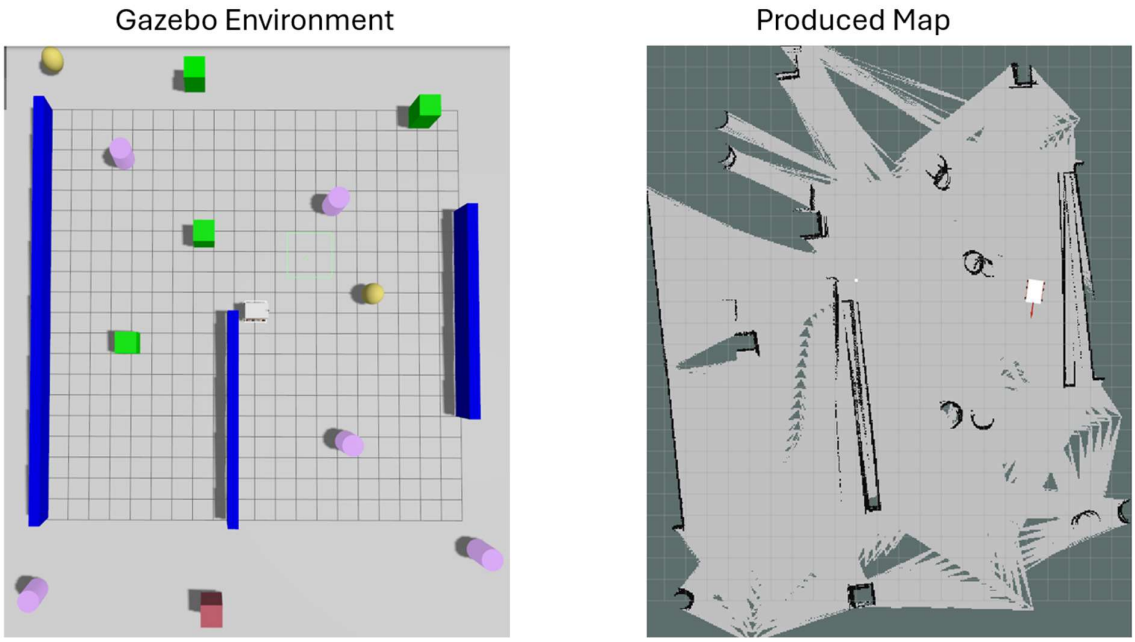


Figure 3.14: Broken Localisation During Mapping in Simulation Environment

3.4.1.2. Autonomous Navigation Results

Once the produced map from the SLAM package was considered viable, tests were undertaken to demonstrate the functionality of the autonomous navigation system. This was done by sending a goal pose to the Nav2 stack via RViz. The autonomous navigation is demonstrated in Figure 3.15 below. It shows the simulation world, the produced SLAM map, the Goal Pose location as indicated by the red flag, and the planned global path as indicated via the green line. The rover then autonomously traversed to the goal pose whilst avoiding the obstacle in the middle of the map

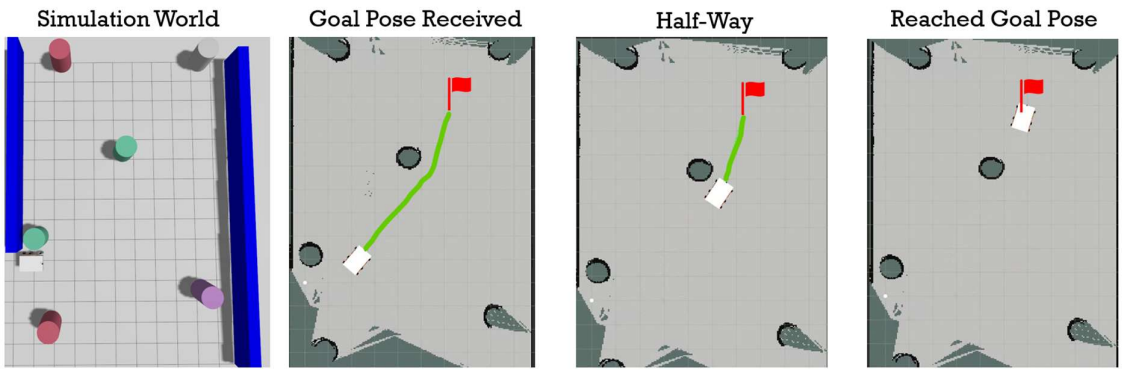


Figure 3.15: Simulated Autonomous Navigation 1

Another example of the autonomous navigation is presented in Figure 3.16 below.

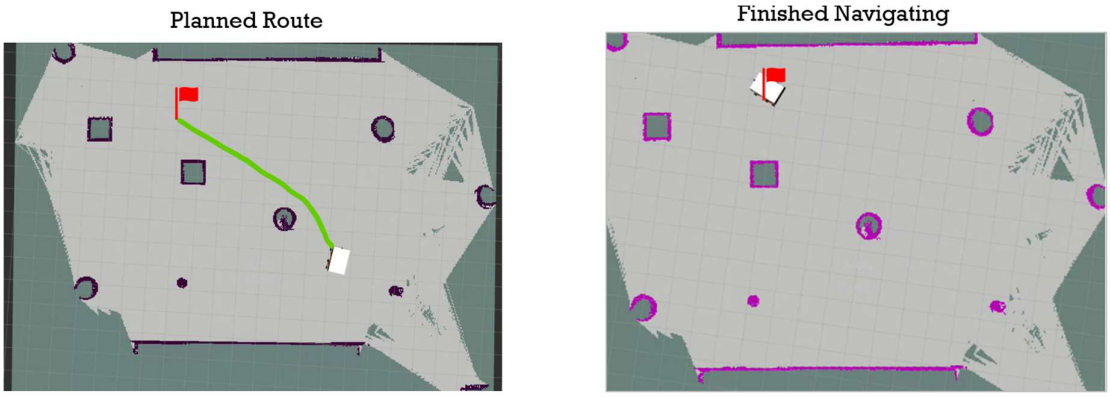


Figure 3.16: Simulated Autonomous Navigation 2

Evidently, in the simulation setup, the rover can autonomously traverse terrain with obstacles in between to navigate to a goal pose.

3.5. Physical Results

All physical tests conducted, resulted in some form of mechanical failure which hampered the efficacy of the results. Nevertheless, multiple attempts were made to test the validity of the software design. After first confirming adequate driving performance on flat surfaces through indoor tests based on its Ackermann-like steering setup using teleoperation, tests were undertaken outdoors to assess the RTAB-Map VSLAM and corresponding Nav2 setup.

3.5.1. SLAM Results

The first VSLAM test took place on a residential driveway with obstacles such as cars and walls around. The physical environment with obstacles in black and clear space in blue is presented in Figure 3.17 below, along with the produced map from the RTAB-Map VSLAM setup.

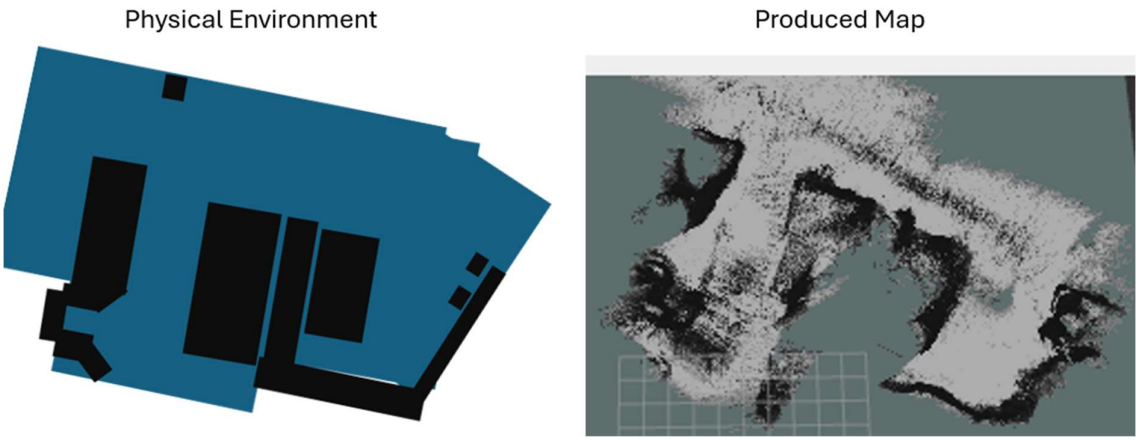


Figure 3.17: Physical SLAM Test 1

Evidently, many erroneous obstacles were being detected, which was likely due to the environmental factors. The test was conducted in the afternoon with direct sunlight affecting the camera. In addition to this, the surface on which it drove was a highly reflective exacerbating the issue. The RGB-D VSLAM approach can be influenced by errors due to sunlight as experienced in this case [44]. Before being able to remap the area and rely on loop closure detection to correct the map, one of the wheels broke off resulting in the emergency stop being pressed, meaning the onboard controller experienced a hard shutdown before additional data could be observed.

Additional VSLAM tests were undertaken in a more natural environment. The produced map from this test is shown in Figure 3.18 below along with the raw image the camera sees.

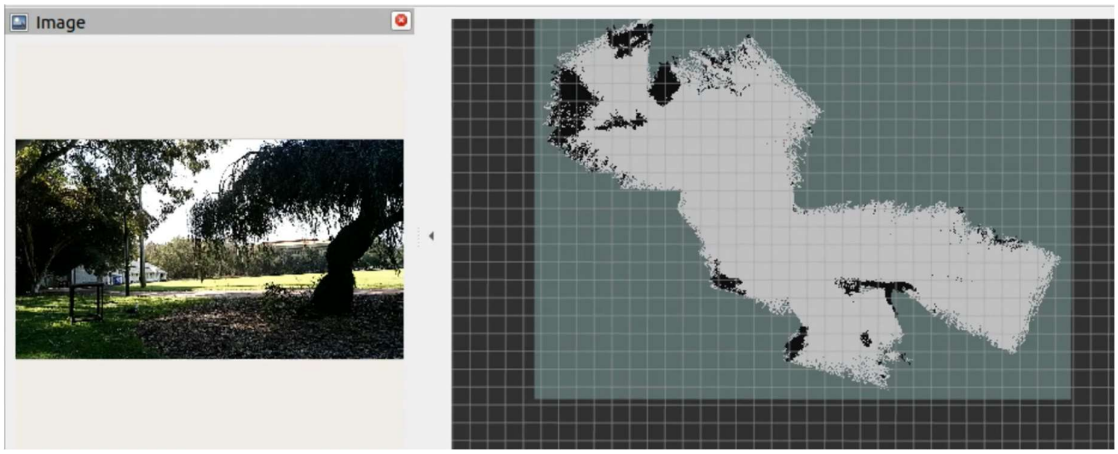


Figure 3.18: Physical SLAM Test 2

This test was successful as the map accurately depicted the surrounding environment and the effects of loop closure correction was observed in real time. This test was conducted at the

front of the Clough building at UWA. It was determined that maps of this quality will be sufficient for autonomous navigation testing. Another example of a produced map is shown in Figure 3.19 below. This map is approximately 28x22m wide and long and was conducted in the courtyard behind the UWA Robotics laboratory.

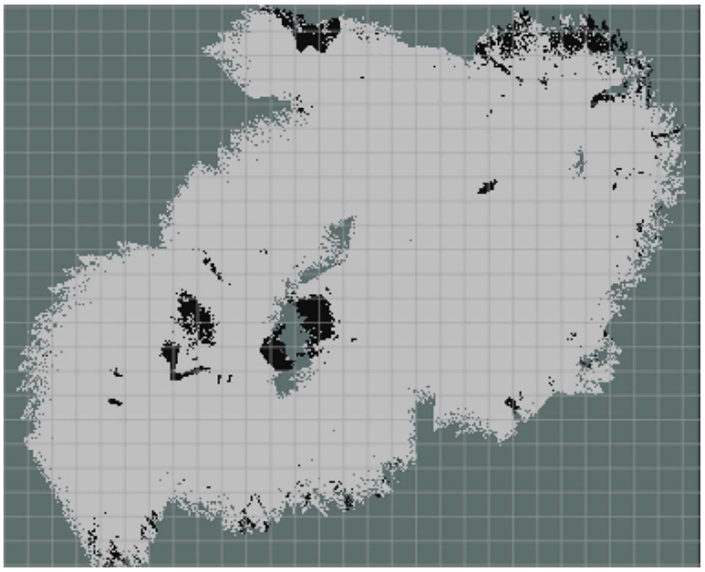


Figure 3.19: Physical SLAM Test 3

3.5.2. Autonomous Navigation Results

The Nav2 setup was tested to confirm that the software design on the physical rover can navigate to a received goal pose. This test was undertaken in the courtyard at the back of the of the UWA Robotics laboratory. The results are presented in Figure 3.20 below. The map was first produced using VSLAM while teleoperating the rover. Once a sufficiently large map was produced, a goal pose was sent via RViz to Nav2. It was expected that the rover would navigate from its current location to the goal pose, however it was discovered that the localisation of the rover body for Nav2 planning was not in the correct location and different from the localisation of the camera doing the VSLAM.

The planned path shown in green in Figure 3.20, was developed from the initial pose of the rover before mapping commenced and did not reflect the rover's current pose. Evidently the path planning and obstacle avoidance worked from a technical perspective, however there was a disjoint between the camera frames used for localisation in RTAB-Map and the rest of the rover, resulting in improper localisation in Nav2. This resulted in the path being planned from the wrong location. It was also visually observed and filmed that the Nav2 setup does

send command velocities to the rover based on its perceived location and proposed direction of travel.

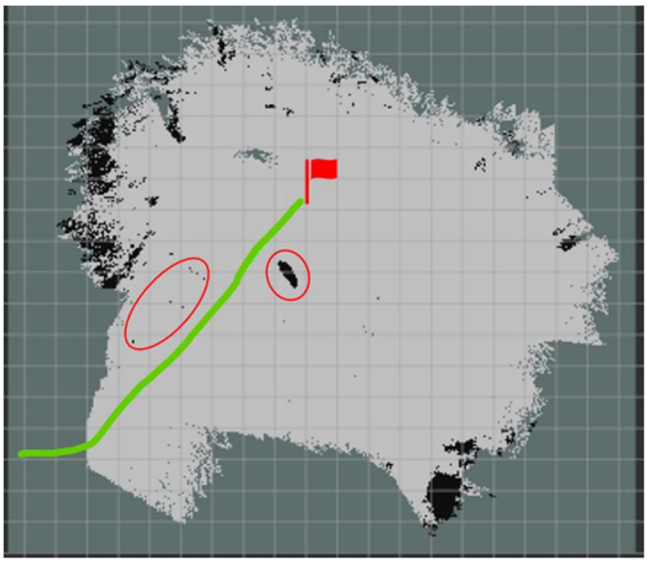


Figure 3.20: Physical Autonomous Navigation Test

3.6. Implications of Results and Limitations

The rover successfully demonstrated mapping and autonomous navigation between points in simulation but did not demonstrate the complete functionality in the physical setup based on the criteria outlined in sections 1.3 and 2.5. The physical setup successfully implemented the robot base controllers, VSLAM and the Nav2 components such as path planning and obstacle avoidance, however the localisation disjoint between the camera and the rover body meant successful navigation between two points on the map was not observed.

The improper localisation in Nav2 is related to the improper frame linking of the Luxonis-DepthAI-ROS-Driver package and the rover's URDF. Futile attempts have been made to rectify the issue however, once this has been fixed, the rover is expected to demonstrate its full autonomous driving capability such that it drives autonomously between points on a map.

The simulation setup was limited by Ignition Gazebo's driving plugins and only reflected an approximation of the physical rover. Additionally, in both simulation and physical setups, the costmaps were being created however trying to visualise them in Rviz proved futile which is thought to be related to a ROS QoS issue. Lastly, the mechanical inadequacies of the rover resulted in mechanical failure during all physical tests, leading to loss of data at times.

4. Conclusion and Future Works

This project aimed to develop an easy-to-use autonomous navigation system for the prototype Mars rover whereby the rover autonomously traverses flat, object laden terrain and navigate between points on a generated map which it has mostly demonstrated.

The successful development and integration of component interfaces into the Jetson controller in ROS2 laid the foundation for the autonomous navigation setup and was demonstrated using joystick teleoperation. The autonomous navigation setup was built using the Nav2 framework utilizing VSLAM, manufacturer drivers, robot base controllers and a variety of sensor inputs and configuration files.

The developed autonomous navigation software implemented the full functionality of the project objectives as outlined in section 1.3 within the simulation environment and mostly implemented this on the physical rover with one minor limitation.

The simulation environment successfully demonstrated the generation of SLAM maps and autonomous navigation between points on the map whilst avoiding obstacles, including incorporating items such as path planning, controlling and smoothing.

The physical rover demonstrated the development of visual SLAM maps, along with its planning and controlling capabilities, however an incorrect frame link between the onboard camera and the rover's URDF breaks localisation within Nav2. The physical setup can plan a feasible route whilst avoiding obstacles and then control the robot base controller, however the planned route is always from the initial pose of the rover.

This project has demonstrated the feasibility of autonomous navigation for the prototype planetary rover both in simulation and in physical experiments, opening opportunities for future works which should first focus on the mechanical issues.

4.1. Future Works

1. Redesign the rover's drivetrain

   A complete redesign of the rover's drivetrain is required to fix mechanical issues related to belt slippage, gear and pin slippage, stripped wheel mounts and inadequate torque. Issues are shown in Appendix D

2. Investigate and correct the frame linking between the camera and the URDF

As discussed in section 3.6, a critical issue facing the software implementation of the autonomous navigation is related to the disjoint frames between the camera and the rover's URDF. Attempts at remapping the camera driver's parent frame has so far been futile and needs to be investigated further. Fixing this issue, will result in correct localisation and autonomous navigation of the physical rover as intended.

3. Fuse the LIDAR and stereo/depth camera in RTAB-Map

RTAB-Map allows for multimodal SLAM and implementing this by fusing the LIDAR and Stereo/Depth camera in the SLAM package, will eliminate over exposure issues related to a purely visual approach as discussed in section 3.5.1

4. Implement frontier exploration

Frontier exploration will allow the rover to autonomously discover or map and unknown environment, removing the need for initial teleoperation. This will further develop the autonomy of the rover as non-populated map wavefronts will automatically be further explored.

5. Bibliography

[1] The National Aeronautics and Space Administration, “Science Objectives,” 04 2024. [Online]. Available: https://science.nasa.gov/mission/mars-exploration-rovers-spirit-and-opportunity/science-objectives/. [Accessed 17 05 2024].

[2] S. Kuthunur, “Space,” 26 09 2023. [Online]. Available: https://www.space.com/perserverance-mars-rover-boulder-field-maneuvers. [Accessed 17 05 2024].

[3] NASA/JPL-Caltech, “Perserverance Rover,” 03 04 2020. [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2020/04/Perseverance_rover. [Accessed 17 05 2024].

[4] P. Brezmen, “Planetary Rover Design and Build Project,” The University of Western Australia, 2023.

[5] S. S. F. Ramachandran, “Smart Walker V: Implementation of RTAB-Map Algorithm,” in *IEEE International Conference on System of Systems Engineering (SoSE)*, Anchorage, AK, USA, 2019.

[6] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, “ROS: an open-source Robot Operating System,” Computer Science Department, Stanford University, Stanford, CA, Stanford, 2009.

[7] M. Bharatheesha, “ROS Services,” TUDelft OpenCourseWare, 2018. [Online]. Available: https://ocw.tudelft.nl/course-lectures/1-3-1-ros-services/#:~:text=ROS%20services%20implement%20these%20request,in%20their%20own%20%2Fsrv%20folder.. [Accessed 12 05 2024].

[8] “Powering the world's robots,” Open Robotics, 2023. [Online]. Available: https://www.openrobotics.org/. [Accessed 13 10 2024].

[9] NOAA, “What is lidar?,” National Oceanic and Atmospheric Administration, 20 01 2023. [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html#:~:text=Lidar%2C%20which%20stands%20for%20Light,variable%20distances)%20to%20the%20Earth.. [Accessed 12 05 2024].

[10] OpenTopography, “What is Lidar?,” OpenTopography, 2011. [Online]. Available: https://opentopography.org/lidar_basics. [Accessed 12 05 2024].

[11] J. Li, H. Ziang and J. Li, “2D LiDAR and camera fusion in 3D modeling of indoor environment,” in *2015 National Aerospace and Electronics Conference (NAECON)*, Dayton, OH, USA, 2015.

[12] e-con Systems, “What is a stereo vision camera?,” e-con Systems, 23 01 2023. [Online]. Available: https://www.e-consystems.com/blog/camera/technology/what-is-a-stereo-vision-camera-2/. [Accessed 17 05 2024].

[13] Longin Jan Latecki, “Temple University - Courses,” 2005. [Online]. Available: https://cis.temple.edu/~latecki/Courses/CIS601-03/Lectures/slides_lec9.pdf. [Accessed 17 05 2024].

[14] MathWorks, “SLAM (Simultaneous Localization and Mapping),” MathWorks, 2024. [Online]. Available: https://www.mathworks.com/discovery/slam.html. [Accessed 12 05 2024].

[15] S. Higgins, “NavVis,” 21 07 2020. [Online]. Available: https://www.navvis.com/blog/how-slam-affects-the-accuracy-of-your-scan-and-how-to-improve-it. [Accessed 12 05 2024].

[16] P. Yu, X. Ruan and X. Zhu, “The loop closure Detection Algorithm Based on Bagof Semantic Word For Robot Navigation,” in *IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA 2020)*, Chongqing, China, 2020.

[17] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): part II,” *IEEE Robotics & Automation Magazine,* vol. 13, no. 3, pp. 108-117, 2006.

[18] Clearview Imaging, “Stereo Vision for 3D Machine Vision Applications,” 2021. [Online]. Available: https://www.clearview-imaging.com/en/blog/stereo-vision-for-3d-machine-vision-applications. [Accessed 17 05 2024].

[19] M. Labbe ´ and F. Michaud, “Multi-Session Visual SLAM for Illumination,” *Frontiers in Robotics and AI,* vol. 9, 2022.

[20] M. Labbé and F. Michaud, “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics,* vol. 36, no. 2, pp. 416-446, 2019.

[21] J. V. Aravind, K. V. S. S. Ganesh and S. Prince, “Real-Time Appearance Based Mapping using Visual Sensor,” *Journal of Physics: Conference Series,* vol. 2335 , 2022.

[22] M. Labbe, “RTAB-Map ROS,” Open Robotics, 19 04 2023. [Online]. Available: http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot#Stereo_A. [Accessed 17 09 2024].

[23] S.-H. Chan, P.-T. Wu and L.-C. Fu, “Robust 2D Indoor Localization Through Laser SLAM and Visual SLAM Fusion,” in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki, Japan, 2018.

[24] M. Labbe and F. Michaud, “Memory Management for Real-Time Appearance-Based Loop Closure Detetction,” in *International Conference on Intelligent Robots and Systems*, San Francisco, 2011.

[25] K. A. Tsintotas, L. Bampis and A. Gasteratos, “The Revisiting Problem in Simultaneous Localization and Mapping: A Survey on Visual Loop Closure Detection,” *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 23, no. 11, pp. 19929-19953, 2022.

[26] S. Macenski, F. Martín, R. White and J. Clavero, “The Marathon 2: A Navigation Systems,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[27] Nvidia, “ROS2 Navigation,” 15 05 2024. [Online]. Available: https://docs.omniverse.nvidia.com/isaacsim/latest/ros2_tutorials/tutorial_ros2_navigation.html#learning-objectives. [Accessed 17 05 2024].

[28] Y. Zhuang, X. Sun, Y. Li, J. Huai, L. Hua, X. Yang, X. Cao, P. Zhang, Y. Cao, L. Qi, J. Yang, N. El-Bendary, N. El-Sheimy, J. Thompson and R. Chen, “Multi-sensor integrated navigation/positioning systems using data fusion:,” *Information Fusion,* vol. 95, pp. 62-90, 2023.

[29] Open Navigation, “Setting Up Odometry,” 2023. [Online]. Available: https://docs.nav2.org/setup_guides/odom/setup_odom.html?highlight=odometry. [Accessed 24 09 2024].

[30] W. Meeussen, “Coordinate Frames for Mobile Platforms,” ROS, 27 10 2010. [Online]. Available: https://www.ros.org/reps/rep-0105.html. [Accessed 04 10 2024].

[31] C. Yeoh, D. Kim, Y. Won, S. Lee and H. Yi, “Constructing ROS Package for Legged Robot in Gazebo Simulation from Scratch,” in *20th International*

Conference on Control, Automation and Systems (ICCAS), Busan, Korea (South), 2020.

[32] P. D. C. Cheng, M. Indri, F. Sibona, M. D. Rose and G. Prato, “Dynamic Path Planning of a mobile robot adopting a costmap layer approach in ROS2,” in *IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Stuttgart, Germany, 2022.

[33] “Planner Server,” Open Navigation, 2023. [Online]. Available: https://docs.nav2.org/configuration/packages/configuring-planner-server.html. [Accessed 04 10 2024].

[34] T. B. Sant’Anna, M. B. Argolo and R. T. Lima, “Comparative analysis in real environment of trajectory controllers on ROS2,” in *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR)*, Salvador, Brazil, 2023.

[35] “Controller Server,” Open Navigation, 2023. [Online]. Available: https://docs.nav2.org/configuration/packages/configuring-controller-server.html. [Accessed 04 10 2024].

[36] “Costmap 2D,” Open Navigation, 2023. [Online]. Available: https://docs.nav2.org/configuration/packages/configuring-costmaps.html. [Accessed 04 10 2024].

[37] R. Górecki, “Navigation,” Husarion, 2024. [Online]. Available: https://husarion.com/tutorials/ros2-tutorials/9-navigation/. [Accessed 04 10 2024].

[38] T. Foote and M. Purvis, “Standard Units of Measure and Coordinate Conventions,” ROS, 31 12 2014. [Online]. Available: https://www.ros.org/reps/rep-0103.html. [Accessed 06 10 2024].

[39] A. M. N and D. F. P. C, “Performance evaluation of ROS on the Raspberry Pi platform as OS for,” *Tekhnê,* vol. 14, no. 1, pp. 61-72, 2017.

[40] D. Franklin, “NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics,” Nvidia, 08 12 2018. [Online]. Available: https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/. [Accessed 05 10 2024].

[41] Richbeam, “2D LiDAR LakiBeam 1S,” Richbeam, Beijing, 2023.

[42] Luxonis, “OAK-D S2,” Luxonis, 2024. [Online]. Available: https://shop.luxonis.com/products/oak-d-s2?variant=42455432233183. [Accessed 17 05 2024].

[43] “AckermannSteering Class Reference,” Gazebo, 2024. [Online]. Available: https://gazebosim.org/api/gazebo/6/classignition_1_1gazebo_1_1systems_1_1AckermannSteering.html. [Accessed 13 10 2024].

[44] C. Theodorou, V. Velisavljevic, V. Dyo and F. Nonyelu, “Visual SLAM algorithms and their application for AR, mapping, localization and wayfinding,” *Array,* vol. 15, p. 100222, 2022.

Appendix A

Component	Quantity	Description
Motor Controller	2	Sabertooth Dual 12A 6V-24V motor controller
Servo Controller	1	Micro Maestro 6-Channel USB servo controller
Depth/Stereo Camera	1	Luxonis Oak-D S2
LIDAR	1	Lakibeam 1S
Controller	1	Nvidia Jetson AGX Xavier Development Kit
Encoder Hub	1	Vint Hub Phidget
Encoders	6	Quadrature Encoder Phidget
Motors	6	FIT0185
Servo’s	6	DS51150 150kgcm Servo

Appendix B

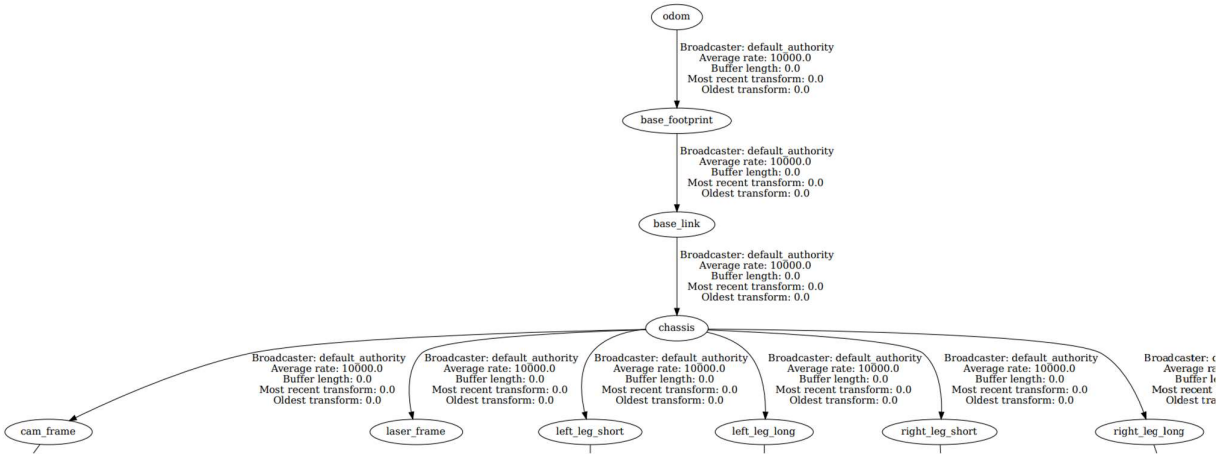


Figure B1: Top View of Transform Tree

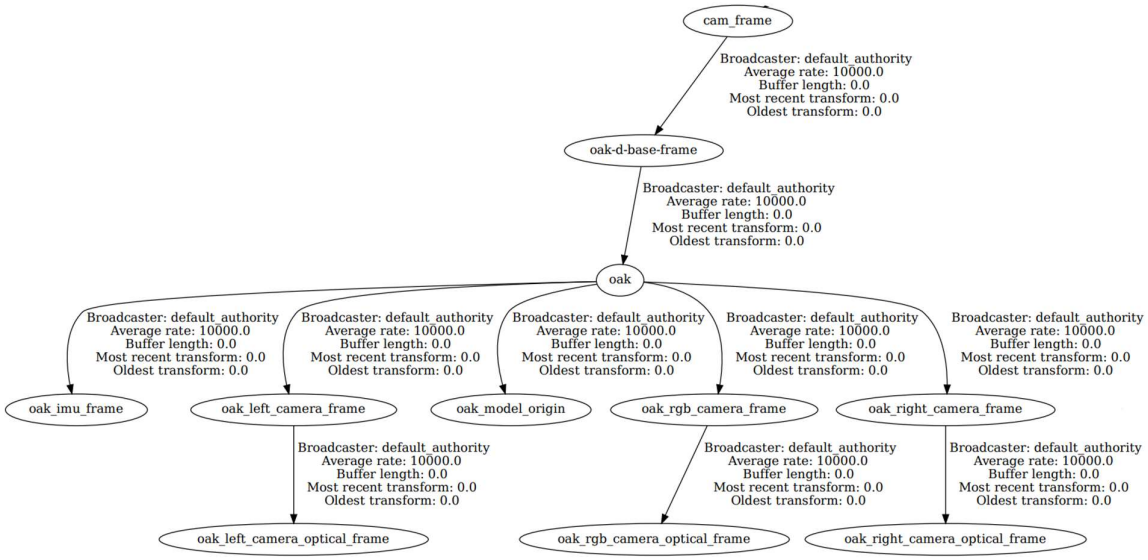


Figure B2: Camera Frames of Transform Tree

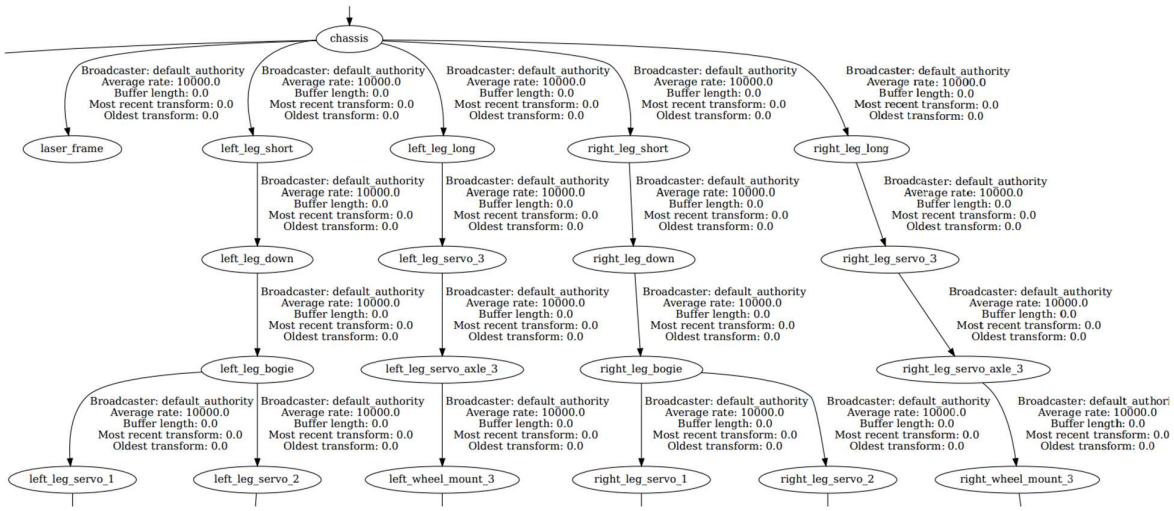


Figure B2: Chassis Top Frames of Transform Tree

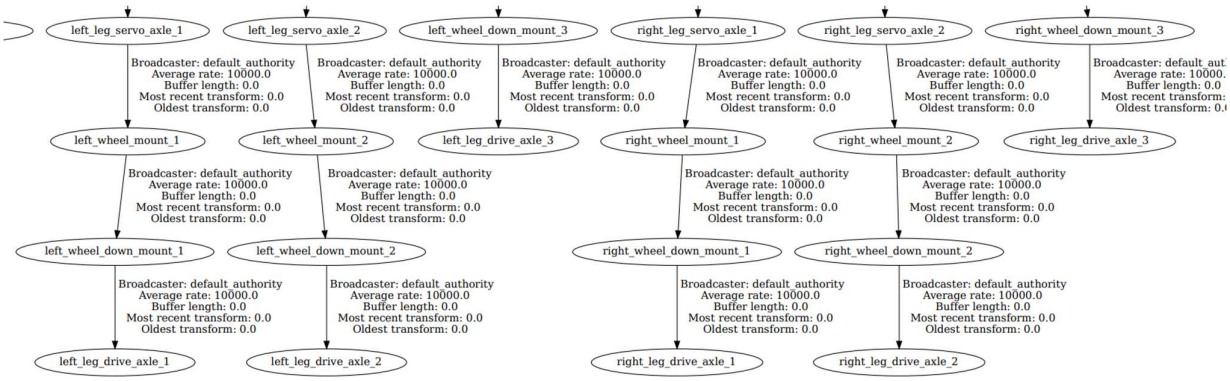


Figure B2: Chassis Bottom Frames of Transform Tree

Appendix C

This is a copy of the navigation.yaml configuration file

```
bt_navigator:
 ros__parameters:
  use_sim_time: True
  global_frame: map
  robot_base_frame: base_footprint
  odom_topic: /odom
  bt_loop_duration: 10
  default_server_timeout: 20
  # 'default_nav_through_poses_bt_xml' and 'default_nav_to_pose_bt_xml' are use defaults:
  # nav2_bt_navigator/navigate_to_pose_w_replanning_and_recovery.xml
  # nav2_bt_navigator/navigate_through_poses_w_replanning_and_recovery.xml
  # They can be set here or via a RewrittenYaml remap from a parent launch file to Nav2.
  plugin_lib_names:
    - nav2_compute_path_to_pose_action_bt_node
    - nav2_compute_path_through_poses_action_bt_node
    - nav2_smooth_path_action_bt_node
    - nav2_follow_path_action_bt_node
    - nav2_spin_action_bt_node
    - nav2_wait_action_bt_node
    - nav2_assisted_teleop_action_bt_node
    - nav2_back_up_action_bt_node
    - nav2_drive_on_heading_bt_node
    - nav2_clear_costmap_service_bt_node
    - nav2_is_stuck_condition_bt_node
    - nav2_goal_reached_condition_bt_node
    - nav2_goal_updated_condition_bt_node
    - nav2_globally_updated_goal_condition_bt_node
    - nav2_is_path_valid_condition_bt_node
    - nav2_initial_pose_received_condition_bt_node
    - nav2_reinitialize_global_localization_service_bt_node
    - nav2_rate_controller_bt_node
    - nav2_distance_controller_bt_node
    - nav2_speed_controller_bt_node
    - nav2_truncate_path_action_bt_node
    - nav2_truncate_path_local_action_bt_node
    - nav2_goal_updater_node_bt_node
    - nav2_recovery_node_bt_node
    - nav2_pipeline_sequence_bt_node
    - nav2_round_robin_node_bt_node
    - nav2_transform_available_condition_bt_node
    - nav2_time_expired_condition_bt_node
    - nav2_path_expiring_timer_condition
    - nav2_distance_traveled_condition_bt_node
    - nav2_single_trigger_bt_node
    - nav2_goal_updated_controller_bt_node
```

```
    - nav2_is_battery_low_condition_bt_node
    - nav2_navigate_through_poses_action_bt_node
    - nav2_navigate_to_pose_action_bt_node
    - nav2_remove_passed_goals_action_bt_node
    - nav2_planner_selector_bt_node
    - nav2_controller_selector_bt_node
    - nav2_goal_checker_selector_bt_node
    - nav2_controller_cancel_bt_node
    - nav2_path_longer_on_approach_bt_node
    - nav2_wait_cancel_bt_node
    - nav2_spin_cancel_bt_node
    - nav2_back_up_cancel_bt_node
    - nav2_assisted_teleop_cancel_bt_node
    - nav2_drive_on_heading_cancel_bt_node
    - nav2_is_battery_charging_condition_bt_node

bt_navigator_navigate_through_poses_rclcpp_node:
  ros__parameters:
    use_sim_time: False

bt_navigator_navigate_to_pose_rclcpp_node:
  ros__parameters:
    use_sim_time: False

controller_server:
  ros__parameters:
    use_sim_time: False
    controller_frequency: 20.0
    min_x_velocity_threshold: 0.001
    min_y_velocity_threshold: 0.5
    min_theta_velocity_threshold: 0.001
    failure_tolerance: 0.3
    progress_checker_plugin: "progress_checker"
    goal_checker_plugins: ["general_goal_checker"] # "precise_goal_checker"
    controller_plugins: ["FollowPath"]

    # Progress checker parameters
    progress_checker:
      plugin: "nav2_controller::SimpleProgressChecker"
      required_movement_radius: 0.5
      movement_time_allowance: 10.0
    # Goal checker parameters
    #precise_goal_checker:
    #  plugin: "nav2_controller::SimpleGoalChecker"
    #  xy_goal_tolerance: 0.25
    #  yaw_goal_tolerance: 0.25
    #  stateful: True
    general_goal_checker:
```

```
  stateful: True
  plugin: "nav2_controller::SimpleGoalChecker"
  xy_goal_tolerance: 0.25
  yaw_goal_tolerance: 0.25
# DWB parameters
FollowPath:
  plugin: "dwb_core::DWBLocalPlanner"
  debug_trajectory_details: True
  min_vel_x: 0.0
  min_vel_y: 0.0
  max_vel_x: 0.36729
  max_vel_y: 0.0
  max_vel_theta: 1.0
  min_speed_xy: 0.0
  max_speed_xy: 0.26
  min_speed_theta: 0.0
  # Add high threshold velocity for turtlebot 3 issue.
  # https://github.com/ROBOTIS-GIT/turtlebot3_simulations/issues/75
  acc_lim_x: 2.5
  acc_lim_y: 0.0
  acc_lim_theta: 3.2
  decel_lim_x: -2.5
  decel_lim_y: 0.0
  decel_lim_theta: -3.2
  vx_samples: 20
  vy_samples: 5
  vtheta_samples: 20
  sim_time: 1.7
  linear_granularity: 0.05
  angular_granularity: 0.025
  transform_tolerance: 0.2
  xy_goal_tolerance: 0.25
  trans_stopped_velocity: 0.25
  short_circuit_trajectory_evaluation: True
  stateful: True
  critics:
   [
    "RotateToGoal",
    "Oscillation",
    "BaseObstacle",
    "GoalAlign",
    "PathAlign",
    "PathDist",
    "GoalDist",
   ]
  BaseObstacle.scale: 0.02
  PathAlign.scale: 32.0
  PathAlign.forward_point_distance: 0.1
```

```
    GoalAlign.scale: 24.0
    GoalAlign.forward_point_distance: 0.1
    PathDist.scale: 32.0
    GoalDist.scale: 24.0
    RotateToGoal.scale: 32.0
    RotateToGoal.slowing_factor: 5.0
    RotateToGoal.lookahead_time: -1.0

local_costmap:
 local_costmap:
  ros__parameters:
    update_frequency: 5.0
    publish_frequency: 2.0
    global_frame: odom
    robot_base_frame: base_footprint
    use_sim_time: False
    rolling_window: true
    width: 3
    height: 3
    resolution: 0.05
    footprint: "[[0.45, 0.45], [0.45, -0.45], [-0.45, -0.45], [-0.45, 0.45]]"
    plugins: ["voxel_layer", "inflation_layer"]
    inflation_layer:
      plugin: "nav2_costmap_2d::InflationLayer"
      cost_scaling_factor: 3.0
      inflation_radius: 0.55
    voxel_layer:
      plugin: "nav2_costmap_2d::VoxelLayer"
      enabled: True
      publish_voxel_map: True
      origin_z: 0.0
      z_resolution: 0.05
      z_voxels: 16
      max_obstacle_height: 2.0
      mark_threshold: 0
      observation_sources: scan
      scan:
        topic: /lidar
        max_obstacle_height: 2.0
        clearing: True
        marking: True
        data_type: "LaserScan"
        raytrace_max_range: 3.0
        raytrace_min_range: 0.0
        obstacle_max_range: 2.5
        obstacle_min_range: 0.0
    static_layer:
      plugin: "nav2_costmap_2d::StaticLayer"
```

```
      map_subscribe_transient_local: True
    always_send_full_costmap: True

global_costmap:
 global_costmap:
  ros__parameters:
    update_frequency: 1.0
    publish_frequency: 1.0
    global_frame: map
    robot_base_frame: base_footprint
    use_sim_time: False
    robot_radius: 0.3
    resolution: 0.05
    track_unknown_space: true
    plugins: ["static_layer", "obstacle_layer", "inflation_layer"]
    obstacle_layer:
      plugin: "nav2_costmap_2d::ObstacleLayer"
      enabled: True
      observation_sources: scan
      scan:
        topic: /lidar
        max_obstacle_height: 2.0
        clearing: True
        marking: True
        data_type: "LaserScan"
        raytrace_max_range: 3.0
        raytrace_min_range: 0.0
        obstacle_max_range: 2.5
        obstacle_min_range: 0.0
    static_layer:
      plugin: "nav2_costmap_2d::StaticLayer"
      map_subscribe_transient_local: True
    inflation_layer:
      plugin: "nav2_costmap_2d::InflationLayer"
      cost_scaling_factor: 3.0
      inflation_radius: 0.55
    always_send_full_costmap: True

map_server:
 ros__parameters:
   use_sim_time: False
   # Overridden in launch by the "map" launch configuration or provided default value.
   # To use in yaml, remove the default "map" value in the tb3_simulation_launch.py file & provide full
path to map below.
   yaml_filename: ""

map_saver:
 ros__parameters:
```

```
    use_sim_time: False
    save_map_timeout: 5.0
    free_thresh_default: 0.25
    occupied_thresh_default: 0.65
    map_subscribe_transient_local: True

planner_server:
  ros__parameters:
    expected_planner_frequency: 20.0
    use_sim_time: False
    planner_plugins: ["GridBased"]
    GridBased:
      plugin: "nav2_navfn_planner/NavfnPlanner"
      tolerance: 0.5
      use_astar: false
      allow_unknown: true

smoother_server:
  ros__parameters:
    use_sim_time: False
    smoother_plugins: ["simple_smoother"]
    simple_smoother:
      plugin: "nav2_smoother::SimpleSmoother"
      tolerance: 1.0e-10
      max_its: 1000
      do_refinement: True

behavior_server:
  ros__parameters:
    costmap_topic: local_costmap/costmap_raw
    footprint_topic: local_costmap/published_footprint
    cycle_frequency: 10.0
    behavior_plugins:
      ["spin", "backup", "drive_on_heading", "assisted_teleop", "wait"]
    spin:
      plugin: "nav2_behaviors/Spin"
    backup:
      plugin: "nav2_behaviors/BackUp"
    drive_on_heading:
      plugin: "nav2_behaviors/DriveOnHeading"
    wait:
      plugin: "nav2_behaviors/Wait"
    assisted_teleop:
      plugin: "nav2_behaviors/AssistedTeleop"
    global_frame: odom
    robot_base_frame: base_footprint
    transform_tolerance: 0.1
    use_sim_time: False
```

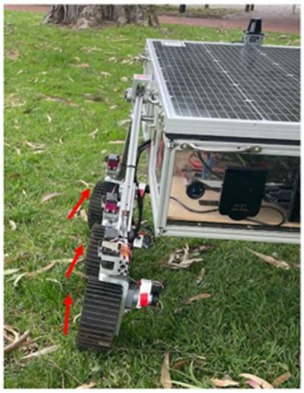
```
    simulate_ahead_time: 2.0
    max_rotational_vel: 1.0
    min_rotational_vel: 0.4
    rotational_acc_lim: 3.2

robot_state_publisher:
  ros__parameters:
    use_sim_time: False

waypoint_follower:
  ros__parameters:
    use_sim_time: False
    loop_rate: 20
    stop_on_failure: false
    waypoint_task_executor_plugin: "wait_at_waypoint"
    wait_at_waypoint:
      plugin: "nav2_waypoint_follower::WaitAtWaypoint"
      enabled: True
      waypoint_pause_duration: 200

velocity_smoother:
  ros__parameters:
    use_sim_time: False
    smoothing_frequency: 20.0
    scale_velocities: False
    feedback: "OPEN_LOOP"
    max_velocity: [0.26, 0.0, 1.0]
    min_velocity: [-0.26, 0.0, -1.0]
    max_accel: [2.5, 0.0, 3.2]
    max_decel: [-2.5, 0.0, -3.2]
    odom_topic: "odom"
    odom_duration: 0.1
    deadband_velocity: [0.0, 0.0, 0.0]
    velocity_timeout: 1.0
```

Appendix D

Mechanical issues experienced are highlighted in the images below

Stripped Grubscrew

Off Centre Wheels



Loose Pin

Broken Wheel

Stripped Grubscrew