

TECHNISCHE UNIVERSITÄT BERLIN

DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS

CHAIR OF SPACE TECHNOLOGY



DEUTSCHES ZENTRUM FÜR LUFT- UND
RAUMFAHRT

INSTITUTE OF ROBOTICS AND MECHATRONICS
OBERPFAFFENHOFEN

Master Thesis

Systems Integration with Autonomous Navigation of the Lunar Rover Mini for a Space Demo Mission

Ricardez Ortigosa, Adrian

Matriculation Number: 452150

supervised by
Prof. Dr.-Ing. Enrico Stoll
Dr.-Ing. Armin Wedler
Dipl.-Ing. Cem Avsar

February 7, 2023

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgment in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged.

Munich, February 7, 2023

.....

Adrian Ricardez Ortigosa

Agreements on rights utilization

The Technische Universität Berlin, represented by the Chair of Space Technology, may use the results of the thesis at hand in education and research. It receives simple (non-exclusive) rights of utilization as according to § 31 Abs. 2 Urheberrechtsgesetz (Urhg). This right of utilization is unlimited and involves the content of any kind (e.g. documentation, presentations, animations, photos, videos, equipment, parts, procedures, designs, drawings, software including source code and similar). An eventual commercial use on part of the Technische Universität Berlin will only be carried out with the approval of the author of the thesis at hand under the appropriate share of earnings.

Place and date

.....

Professor Dr.-Ing. Enrico Stoll
Head of the Chair of Space Technology

Acknowledgments

I would like to express my gratitude to my supervisor Dr. Armin Wedler, for his valuable technical feedback and continuous moral support throughout this thesis work. I would also like to thank Cem Avsar for all the time and motivation he gave me as a supervisor and mentor.

My family has been a pillar throughout my career as a professional. They have stayed by my side in the best and most difficult moments. I also appreciate all the close friends I have, whose presence was also of great support to me, here in Europe, and in Mexico.

I want to give a special thanks to CONACYT and the DAAD for having supported me in most part of my master's degree.

Finally, I would like to mention the important contribution to the project from the DLR colleagues who guided me in solving small to big problems.

Abstract

Over the last few years, the area of robotic exploration has been growing very quickly. There are already various rover models on the Moon and Mars for planetary exploration purposes, among other functions. However, launching these missions is often too expensive and complex. Therefore, it is sought at DLR for ways to create low-cost prototypes for experimental testing, algorithms development, and even usage for educational lectures at many institutions. This work, the Lunar Rover Mini, shows an approach to the development and integration of modular systems at the Robotics and Mechatronics Center. Such prototype, based on the ExoMars rover, will be used as a testing platform to execute three driving modes on terrains similar to those on the Moon and Mars, which can be found in the Vulcano island and on artificially-created testbeds. The systems, as well as the data pipeline, will be validated and evaluated. It will be determined if the LRM can be remotely and manually operated and if the autonomous navigation featured successful obstacle avoidance and path planning to reach a final objective.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives and Scope	1
1.2.1	Primary Objectives	2
1.2.2	Secondary Objectives	2
1.3	System Requirements	2
1.3.1	Functional Requirements	2
1.3.2	Design Requirements	3
1.3.3	Validation Requirements	4
1.4	Main Events	4
1.4.1	Vulcano Summer School	4
1.4.2	Festival der Zukunft	5
1.5	Outline	6
1.5.1	Use cases	7
2	Literature Review	8
2.1	System Overview	9
2.1.1	High-level Hardware	9
2.1.2	Low-level Hardware	11
2.1.3	On-Board Computer Subsystem	12
2.2	Software Framework	13
2.2.1	Graphical User Interface	13
2.2.2	Simulink	15
2.2.3	Links and Nodes Manager	16
2.2.4	Cissy	17

3	Development	18
3.1	Previous Rover Version	18
3.2	New Electrical Power System	19
3.2.1	Graphical Interconnection Diagram	21
3.3	Communications System	22
3.4	Algorithms for the OBC	24
3.4.1	Driving Modes	25
3.4.2	LN World: Breaking Down The Modules	30
3.4.3	ROS World: Breaking Down The Modules	38
3.4.4	Final Data LN-ROS Pipeline	42
4	System Performance	44
4.1	Electrical Behavior	44
4.2	Communication Behavior	45
4.3	Low-Level Data Handling Behavior	45
5	Tests Results & Analysis	46
5.1	Vulcano Summer School	46
5.1.1	Driving Modes In Real Environments	47
5.1.2	Mapping of the Dried “Moon Lake”	48
5.1.3	WiFi Connection Results	50
5.1.4	Presentation	51
5.2	Festival der Zukunft	51
5.2.1	Mapping - Deutsches Museum Testbed	52
5.3	Space Demo Mission	53
5.3.1	Indoor Evaluation Setup	54
5.3.2	Manual Mode	58
5.3.3	Autonomous Navigation	59
6	Conclusions	65
6.1	Future Work	66
6.1.1	Proposal for a Permanent Exhibition	67
6.1.2	New Pipeline Version	68
6.1.3	Migration to New Chassis	68
6.2	Work Contribution	69
6.2.1	GitHub Documentation	69

A	Relevant Links at DLR	74
B	Mission Instructions	76
B.1	Establishing The WiFi Connection	76
B.2	Running The Processes	76
C	Simulink: Driving Modes	78
C.1	Ackerman mode script	78
C.2	Rotation mode script	82
C.3	Crabwalk mode script	83
D	gamepad_controller code	85
E	rover_communication with Simulink	91
F	rover_communication no Simulink	99
G	pantilt tf publisher code	109

List of Figures

1.1	Vulcano Summer School in Sicily, Italy	5
1.2	Festival Der Zukunft at the Deutsches Museum	6
2.1	LRM system overview	8
2.2	Intel RealSense Depth Camera D435i [7]	9
2.3	Next Unit of Computing (NUC) model 8i7BEK [8]	10
2.4	Logitech Gamepad F710 [9]	10
2.5	WiFi antenna module	11
2.6	MiddleBoard PCB	11
2.7	BogieBoard PCB	12
2.8	Servomotor for Locomotion: linear velocity and steering [15]	12
2.9	On-Board Computer Subsystem (OBC)	13
2.10	LRM's GUI for testing and calibration [13]	14
2.11	Simulink model: home view [13]	15
2.12	LN Manager processes example [13]	16
3.1	Electrical Power Subsystem diagram	20
3.2	14.8V-LiPo battery [14]	20
3.3	14.8V to 9.0V DC-DC Converter [16]	21
3.4	Graphical Interconnection Diagram	21
3.5	WiFi modules	22
3.6	WiFi Antenna Station	23
3.7	Ackerman Steering geometry	26
3.8	Rotation geometry	28
3.9	Crabwalk geometry	30
3.10	gamepad_controller node diagram	31
3.11	LRM_Simulink node diagram	32
3.12	rover_communication node diagram	33

3.13	Autonomous Navigation: driving modes conditioning	36
3.14	rostopic2ln node diagram	37
3.15	LN world integration diagram	38
3.16	Realsense ROS Diagram	39
3.17	Scaled-down OBC prototype with navigation concept and locomotion	40
3.18	RMC GBR Navigation Diagram	41
3.19	RMC Local Mapping node diagram	42
3.20	lrm_pan_tilt_tf_publisher node diagram	42
3.21	Final LN-ROS data pipeline scheme	43
5.1	Terrains at Vulcano island	47
5.2	2D Map of the dried Moon Lake	48
5.3	Occupancy Grid of the Moon dried Lake	49
5.4	WiFi Antenna Range	50
5.5	LRM's poster at the Vulcano Summer School	51
5.6	Testbed at the Deutsches Museum	52
5.7	PointCloud of the Deutsches Museum's testbed	53
5.8	Space Demo Mission setups	56
5.9	Space Demo Mission experiment	57
5.10	Experiment Validation Results	57
5.11	Instantaneous Pointcloud	60
5.12	2D Map and trajectory	61
5.13	3D Map, path planner with obstacle avoidance	62
5.14	Costmap from the RMC Local Mapping	63
6.1	Apriltag code example [26]	67
6.2	New design migration concept [13]	69
6.3	Work contribution statistics	69

Acronyms

6DoF 6-Degrees-of-Freedom. 10

ARCHES Autonomous Robotic Networks to Help Modern Societies. 4

Cissy Continuous Integration Software System. 16, 17, 70

COMMS Communications System. 3, 19, 22

CPU Central Processing Unit. 9, 19

DC-DC Direct Current to Direct Current. VIII, 21, 45

DER Design Requirements. 3, 4, 19, 22, 24, 25

DLR Deutsches Zentrum für Luft- und Raumfahrt. I, 2, 4–6, 10, 11, 13, 17, 22, 23, 25, 30, 41, 42, 51, 70, 76

EPS Electrical Power Subsystem. 3, 19, 21

ESA European Space Agency. I, 51

FNR Functional Requirements. 2, 3, 24

GND Ground. 12

GPS Global Positioning System. 48

GUI Graphical User Interface. VIII, 13, 14, 18, 30, 31

IDE Integrated Development Environment. 12, 77

IMU Inertial Measurement Unit. 9, 10, 52

IT Information Technology. 50

LED Light-Emitting Diode. 22, 23

LiPo Lithium-Polymer. 19, 20, 44

LN Links and Nodes Manager. VIII, 2, 16, 24, 30–33, 37, 38, 55, 58, 60, 65, 70, 77

LRM Lunar Rover Mini. I, VIII, IX, 2–10, 14, 16, 18–20, 23, 34–36, 45, 48, 50–56, 59, 61, 64, 66, 68–70, 76

LRU Lightweight Rover Unit. 6, 25, 41, 51, 53, 64, 66

NASA National Aeronautics and Space Administration. I

NUC Next Unit of Computing. VIII, 10–13, 19, 21, 22, 24, 31, 44, 55, 59, 76

OBC On-Board Computer Subsystem. VIII, IX, 3, 4, 12, 13, 19, 24, 25, 40

OPR Operational Requirements. 22

PCB Printed Board Circuit. VIII, 11, 12, 44, 45

PEL Planetary Exploration Laboratory. 53, 58

PWM Pulse Width Modulation. 12, 13

RGB Red, Green, and Blue. 39, 40

RMC Robotics and Mechatronics Center. IX, 10, 17, 22, 24, 41, 42, 50, 63, 70

RMPM Robotics and Mechatronics Package Management. 17

ROS Robot Operating System. 2, 30, 35, 37–39, 46, 55, 58, 64, 65, 77

RTAB-Map Real-Time Appearance-Based Mapping. 40, 59–61, 77

SLAM Simultaneous Localization and Mapping. 40, 49, 53, 56

TF Transformation. 39, 41

TRL Technology Readiness Level. 66

UART Universal Asynchronous Receiver-Transmitter. 11, 12

USB Universal Serial Bus. 9–11, 18, 31

USC Use Cases. 7

VAR Validation Requirements. 4, 46, 54

VSS Vulcano Summer School. 4, 5, 46, 54, 58

WiFi Wireless Fidelity. VIII, IX, 10, 11, 13, 21–24, 45, 48, 50, 55, 58, 76,
77

Introduction

1.1 Motivation

Over the last few years, the area of robotic exploration has been growing very quickly. There exist the classic exploration rovers such as the ExoMars rover of European Space Agency (ESA) [1] and the Curiosity rover of National Aeronautics and Space Administration (NASA) [2]. However, there are many environments that are not explorable yet with such systems. For these areas, a large number of new robots, such as unmanned aerial vehicles, lava-tube explorers, and underwater mobile mechanisms are being developed. Building and launching these space missions is extremely expensive, therefore it is sought to find a way to demonstrate some of these concepts through low-cost prototype development at the Deutsches Zentrum für Luft- und Raumfahrt (DLR) for science contribution, engineering, and education purposes.

1.2 Objectives and Scope

The focus of this master thesis is on a partial development and a full integration of a system mainly based on the ExoMars rover: the Lunar Rover Mini (LRM).

The scope of this thesis includes the adaptation and standardization of data exchange and variables, as well as some mathematical equations for the demonstration of planetary exploration concepts during indoor and outdoor experiments.

In order to boost the motives for space rover missions, some reachable

objectives were stated in order to help the low-budget research process here on Earth.

1.2.1 Primary Objectives

1. To achieve a fully functional system integration of the LRM.
2. To validate the system performance through an experiment in a controlled environment.
3. To use the LRM prototype as an open-source learning platform for students.¹

1.2.2 Secondary Objectives

1. To integrate an autonomous navigation algorithm into the LRM data pipeline ².
2. To contribute to the application for a space demo mission proposal as a permanent exhibition.

1.3 System Requirements

In order to fulfill the previously stated objectives, different requirements were derived at the system level. They were based on the inputs from the DLR project supervisors.

1.3.1 Functional Requirements

The Functional Requirements (FNR) are unique and are the heart of what the system is expected to do and describes the capabilities the LRM needs to have to accomplish its mission [3].

¹Educational concepts in school to learn: robotic frameworks (e.g. ROS, LN, LN Manager, Robotkernel, Sensornet, DDS, ROS2), high-level programming languages (Python, C++), high-level mission planning (Skretch, RAFCON-Mission Logic), and control theory using Simulink (drive modes, signal observation

²It is important to mention this “data pipeline” since it is a special one. In spite of being an open-source project, this framework works with Cissy and other software tools with multiple packages released only in the DLR framework, **which will be explained in more detail in the coming sections.**

Table 1.1: Functional Requirements

ID	FNR_01
Description	The LRM shall perform three rover driving modes: Ackerman, Rotation, and Crabwalk modes.
Justification	This provides the locomotion with many capabilities to move in most required directions, which let the rover adapt better to the terrain.
Flow down to	System requirements: OBC
Verification Method	Review of Framework Integration, Test
ID	FNR_02
Description	The LRM shall allow remote manual operation by using a GamePad.
Justification	This allows the user to control the rover and to analyze real-scenario situations.
Flow down to	System requirements: OBC
Verification Method	Review of Framework Integration, Test, Analysis
ID	FNR_03
Description	The LRM shall be able to perform autonomous navigation.
Justification	This provides a full-robotics integration level, in which the main goal is to achieve a completed task through an autonomous behavior.
Flow down to	System requirements: OBC
Verification Method	Review of Framework Integration, Test, Analysis
ID	FNR_04
Description	The LRM shall provide a mission interface, which external users can use to start the rover processes.
Justification	This promotes the practical experience of the working rover principles to students, researchers, and the general public.
Flow down to	System requirements: OBC
Verification Method	Review of Framework Integration, Test, Analysis

1.3.2 Design Requirements

The Design Requirements (DER) allows defining the functional attributes that enable the LRM to convert ideas into design technical features [3].

Table 1.2: Design Requirements

ID	DER_01
Description	The LRM shall feature at least all previously used hardware: F710 GamePad, MiddleBoard, BogieBoards, D435i camera, NUC, LiPo battery, DC-DC converter, WiFi antenna modules, locomotion, and Pantilt servos.
Justification	This eases the hardware selection process by using all existing modules used by previous students and focusing only on the high-level algorithms and integration.
Flow down to	System requirements: OBC, EPS, and COMMS
Verification Method	Review of Framework Integration, Review of Design, Test

ID	DER_02
Description	The LRM shall use Simulink as the centralized framework tool for control loops.
Justification	Simulink/Matlab reduces the complexity and effort of programming compared to using a non-standardized software tool or integrating everything in Python.
Flow down to	System requirements: OBC
Verification Method	Review of Framework Integration, Test

1.3.3 Validation Requirements

The Validation Requirements (VAR) are scenario specifications that describe how the concept validation should be performed. These are pragmatically stated terms to describe an environment setup, for example.

Table 1.3: Validation Requirements

ID	VAR_01
Description	The LRM's locomotion shall be tested on rocky terrains, such as the ones at Vulcano island.
Justification	This validation makes possible to qualify and quantify how suitable the prototype is for more realistic terrains.
Verification Method	Test, analysis
ID	VAR_02
Description	The LRM shall be tested on an artificial testbed that simulates a Lunar environment.
Justification	This validation will provide an opportunity to analyse more specific behaviors in a realistic controlled environment.
Verification Method	Test, analysis

1.4 Main Events

During a full year of internship at the DLR from Feb. 2022 til Feb. 2023, the LRM was used at two main events: the Vulcano Summer School (VSS), and the Festival der Zukunft at the Deutsches Museum.

1.4.1 Vulcano Summer School

In June 2022, the VSS, sponsored by the Autonomous Robotic Networks to Help Modern Societies (ARCHES) [4] took place at the Vulcano Island in Sicily, Italy. Here, students and researchers from DLR Oberpfaffenhofen, DLR Berlin, and Jacob's University, daily presented lectures and performed

their own project experiments for 13 days.



(a) The LRM at Vulcano



(b) Vulcano group: around 35 people

Figure 1.1: Vulcano Summer School in Sicily, Italy

The LRM (Fig. 1.1a) was tested on the top of the vulcano, where other people (Fig. 1.1b) brought scientific experiments, such as land mapping using drones.

The obtained results from the Cratere della Fossa were analyzed and shared with all the VSS participants at the end of the event. This was a unique experience, a mixture of knowledge between geologists, physicists, and engineers. It was a perfect opportunity for networking and testing the rover in more realistic and uncontrolled environments, terrains similar to those on the Moon and Mars.

In the following sections, the results of the mapping and locomotion tests will be explained in detail.

1.4.2 Festival der Zukunft

In July 2022, the Deutsches Museum invited the DLR to participate in the Festival der Zukunft [5], an event sponsored by IE9, where many scientific and technological companies came together to announce their idea and/or market their product. To the LRM team, this was another unique opportunity to show the working rover principles within a controlled environment.

The experiments were presented and explained to the public. The Bavarian ambassador and the head of the museum were also present at the event.



Figure 1.2: Festival Der Zukunft at the Deutsches Museum

The museum organizers provided the DLR with a 3x4 m testbed to show three rover models: the LRM version of this thesis, a future model of the LRM, and the famous Lightweight Rover Unit (LRU), which can be seen in Fig. 1.2.

As well as in the Vulcano section, the results obtained during the exhibition at the Deutsches Museum will be explained.

1.5 Outline

In the following chapters, the system characteristics, performed experiments, results, and analysis will be explained.

The results of the indoor evaluation, labeled as the “space demo mission”, as well as the and outdoor experiments at Vulcano, are mainly based on integration validation and general performance accuracy.

With the conclusions and future work, it is expected that the next generation of students understand all the working principles explained in this thesis and can improve the system features such as the Simulink model and further autonomous navigation integration.

1.5.1 Use cases

Table 1.4: Use Cases

ID	Description
USC.01	The LRM can be used to demonstrate the working principles of remote manual control.
USC.02	The LRM can be used to test different controller interfaces via standardized parameters.
USC.03	The LRM can be used to demonstrate working principles of autonomous navigation.
USC.04	The LRM can be used to improve path-planning algorithms.
USC.05	The LRM can be used to demonstrate the application of planetary exploration concepts.
USC.06	The LRM can be used to show working principles of module integration for space demo missions.

Literature Review

In order to understand the systems integrated into the rover, it is necessary to explain a little about some principles and features used in the instrumentation.

In previous works, only partial documentation existed along with several outdated modules without maintenance. Indeed, there was a Manual (2021) [6] that mainly explained the low-level architecture, such as the communication protocol for locomotion, but it did not contain updated status information. Hence, a theoretical and practical investigation was carried out to define all functions of the system.

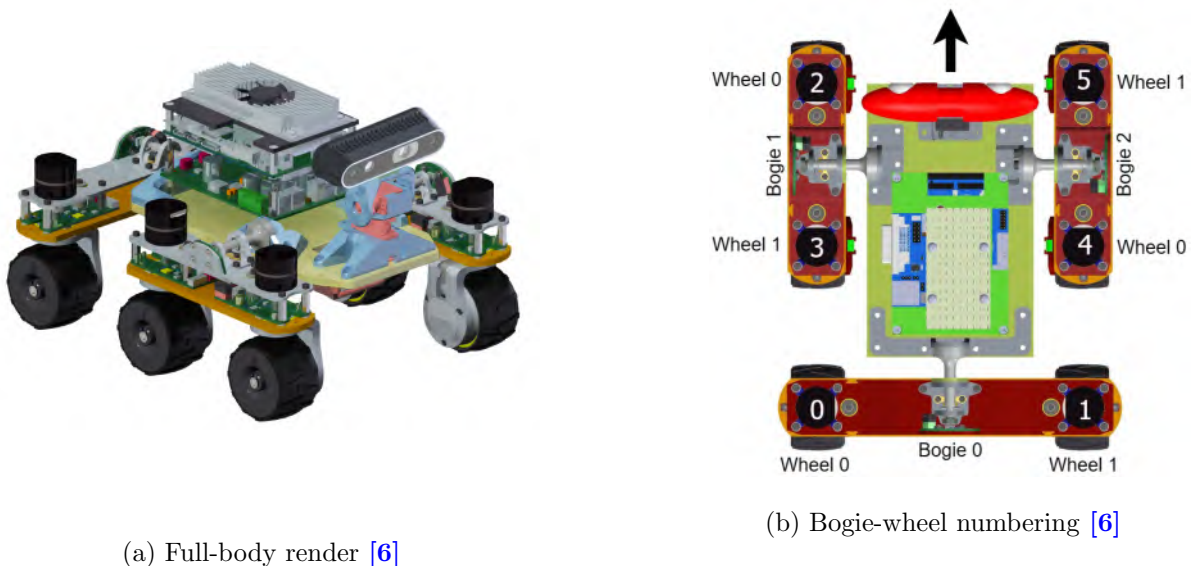


Figure 2.1: LRM system overview

2.1 System Overview

In this section, the LRM's subsystems are described. Fig. 2.1a shows a rendered graphical representation of the rover model, and Fig.2.1b shows the bogie numbering convention for all algorithms and protocols, which fits the standard followed by the Python, C++, and Simulink/Matlab scripts.

2.1.1 High-level Hardware

The high-level hardware is where most of this thesis development was implemented since these are tools directly connected to programs with Python or devices with encapsulated functionalities.

Camera

For the image capture, a stereoscopic depth camera model D435i is being used (Fig. 2.2), which extracts depth points from the environment. This image processing, which is managed by the Central Processing Unit (CPU), can perform object recognition and self-localization, depending on the implemented algorithm.



Figure 2.2: Intel RealSense Depth Camera D435i [7].

This camera also features an Inertial Measurement Unit (IMU), which helps to increase the accuracy of the odometry, unlike its companion the D435 model, which does not have this integrated circuit [7].

The main features of the D435i are:

- indoor/outdoor device usage
- vision processor D4
- up to 1280 x 720 active stereo depth resolution
- up to 1920 x 1080 RGB resolution with 30 fps
- Universal Serial Bus (USB)-C* 3.1 Gen 1* connector
- diagonal field of view over 90°

- range 0.2 m to over 10 m (it depends on the light conditions)
- depth frame rate up to 90 fps
- a 6-Degrees-of-Freedom (6DoF) IMU

Since this device is used extensively for mapping and vision prototyping within the DLR, it was kept for the LRM.

Intel - NUC



Figure 2.3: Next Unit of Computing (NUC) model 8i7BEK[8].

TheNext Unit of Computing (NUC) is a mini computer with diverse characteristics that allow the LRM to operate competently with the grouped processes, in addition, to handle different peripheral ports for all needed devices: the depth camera, the MiddleBoard, and the WiFi antenna. For more detailed features, one can visit its official website [8].

The encasing of Fig. 3.3 fits perfectly with the LRM's dimensions, being much better than a Raspberry Pi with respect to the size-performance ratio.

Python is the main programming language used in this device due to its compatibility with various software tools.

Logitech GamePad

This device is often used at the RMC. The F710 GamePad (D-pad) of Fig. 2.4 is a four-switched precise control device. It uses a 2.4GHz Bluetooth Universal Serial Bus (USB) nano receiver, and two AA batteries as the power source [9]. It is mainly used for the manual control and a graphical control interface.



Figure 2.4: Logitech Gamepad F710 [9].

Wireless Fidelity (WiFi) Antenna

There are two ways to establish communication with the rover. The first one is through Ethernet cable, which comes from the DLR internal network and allows to:

- update software packages
- improve big data sharing, especially graphics
- and other features



Figure 2.5: WiFi antenna module

The second way is with the WiFi antenna of Fig. 2.5, which is connected through the Ethernet port of the NUC. The great advantage of this setup is that the rover can be remotely operated without any cable attached from outside, making easier the experimental setup.

The biggest disadvantage is that, as one might think, data exchange is slower and a delay is even created by default.

A more detailed description of how to use it and to connect to it will be explained in Sec. 3.3.

2.1.2 Low-level Hardware

The low-level hardware, such as the Printed Board Circuit (PCB)s, are devices mostly designed and manufactured at the DLR. These components feature communication with commands managed by C++ programs, which were developed by students from previous generations.

MiddleBoard

The MiddleBoard receives data from the main computer via USB, processes it, and sends it to the BogieBoards as quickly as possible. It uses two Atmel microcontrollers for on-board data handling.

This PCB transmits and receives data via USB from the NUC, and transmits the Universal Asynchronous Receiver-Transmitter

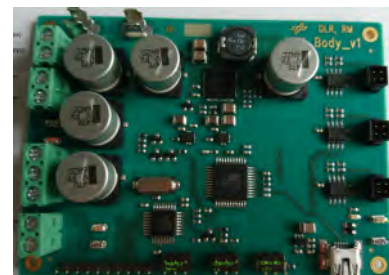


Figure 2.6: MiddleBoard PCB

(UART) protocol over three buses of four wires each to the BogieBoards. Also, it sends Pulse Width Modulation (PWM) signals to the PanTilt servomotors with two data buses of three wires each (GND, VCC, PWM). It uses a C++ program, developed with the Integrated Development Environment (IDE) Atmel Studio for data packaging.

BogieBoards



Figure 2.7: BogieBoard PCB

A BogieBoard could be said to be the PCB that works at the lowest level. The six BogieBoards receive data from the MiddleBoard and convert the commanded values into electrical power to the actuator coils for the steering and speed. In addition, they have a magnetic sensor that allows calibration. They were also flashed with a C++ program.

Servomotors

The Faulhaber SR-FLAT series 2619S006 servomotors have a 207 to 1 reduction with an extremely flat shape. Their length range goes from 6 mm to 19 mm and it has a four poles design. The moment of inertia is reduced to the minimum, and it has integrated optical encoders [15].



Figure 2.8: Servomotor for Locomotion: linear velocity and steering [15]

These servomotors are used to move the rover's locomotion, both in steering and linear speed. Each bogie counts with two servos, counting 12 in total, which are directly connected to the BogieBoards.

2.1.3 On-Board Computer Subsystem

The On-Board Computer Subsystem (OBC) is responsible for communication, data processing, control, and monitoring. Its architecture is shown in Fig. 2.9 and is described as follows:

- The NUC receives data from the depth camera and processes it.

- The NUC is connected to the process manager mission interface via DLR's Intranet or via local WiFi network.
- The MiddleBoard receives the high-level data protocol from the NUC, and commands the PanTilt servomotors with PWM. In addition, the MiddleBoard transmits the locomotion values to the BogieBoards.
- The BogieBoards transmit the low-level power data to the wheel's servomotors.

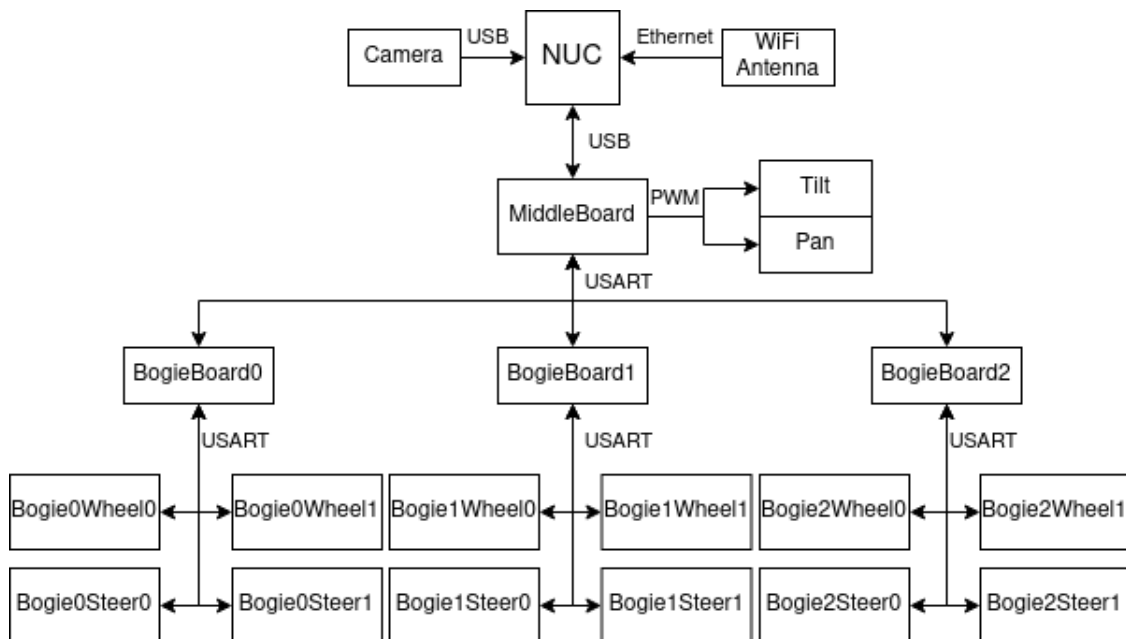


Figure 2.9: On-Board Computer Subsystem (OBC)

2.2 Software Framework

The software tools that were already utilized for the codes and interfaces are here described.

2.2.1 Graphical User Interface

The Graphical User Interface (GUI) of Fig. 2.10 was developed by a previous student, who decided to create a locomotion testing and calibration tool. Such GUI sends a protocol of 68 integer and float values to command

the movement [13].

It is planned that it will continue to be used as an offline tool, without having a dependency on middleware or other programs, just a python script with which the base commands can be tested.

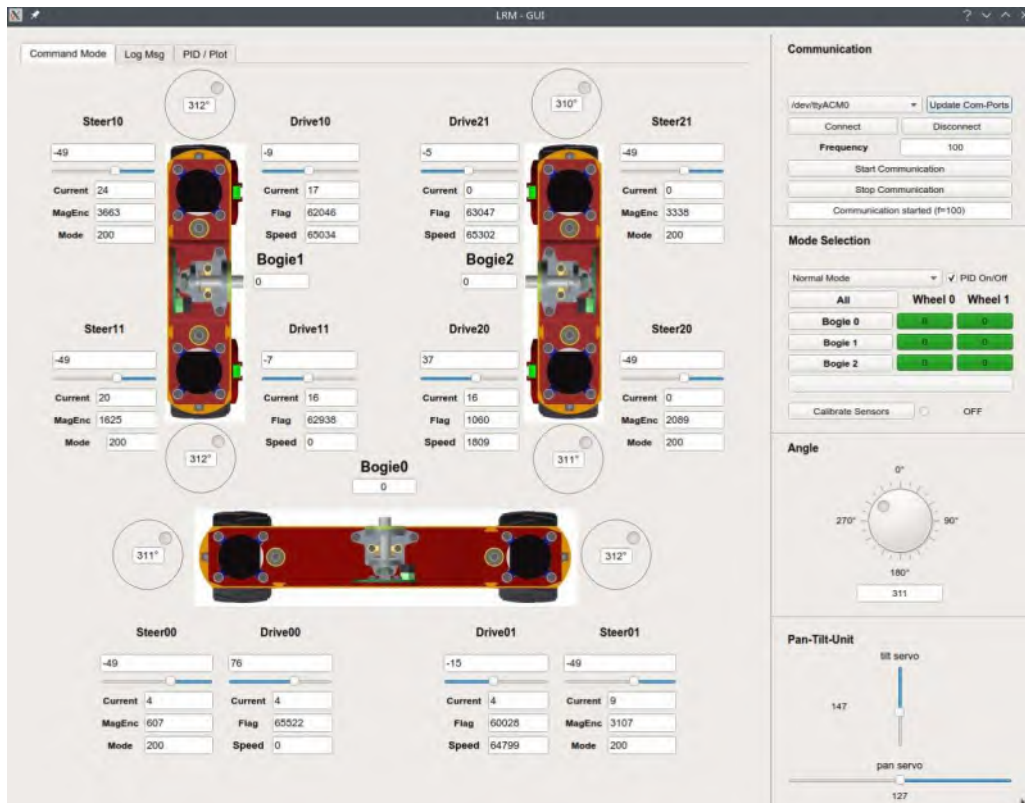


Figure 2.10: LRM's GUI for testing and calibration [13]

Besides connecting the GamePad program with the MiddleBoard program, the GUI allows the operator to calibrate the wheels so that they mark the same steering angle at the beginning of each test, taking advantage of the BogieBoard's magnetic sensor that detects the actual position. With this, it does not matter if any mechanical modifications are made, the rover will always have this calibration option, which is suggested to be carried out every two months.

2.2.2 Simulink

Simulink is used in this project to combine all Cartesian control loops at one place in one control framework. All sent commands and received sensor values are first processed and passed by this pipeline.

The model consists of the controlling part of the rover and provides the data for communication which states how the locomotion should move next depending on its input. It features a Controller (Prio1, Prio2, Prio3), a Controller switch, Pantilt, and Wheels blocks (Figure 2.11).

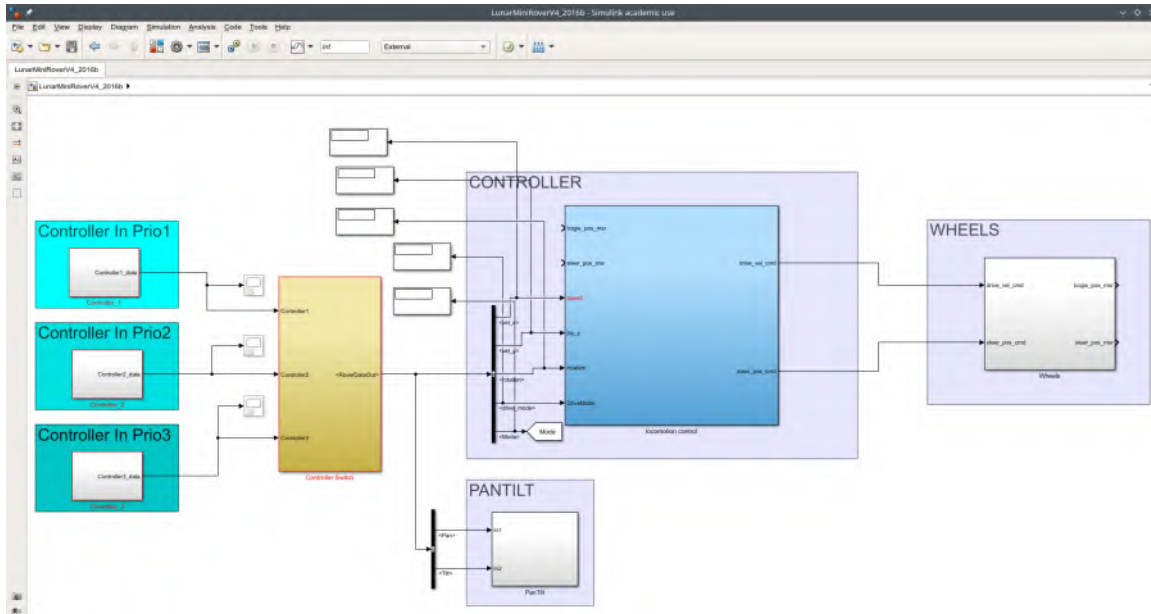


Figure 2.11: Simulink model: home view [\[13\]](#)

These three controllers represent different input devices: the GamePad, autonomous navigation, and an open slot for future implementations. The main idea is that the input values from the control devices are directly transferred to Simulink. Then, the output of the model is transferred to a python communication script. It was planned that, for the work of this thesis, the Ackerman, Rotation, and Crabwalk modes could be implemented in the Simulink model.

2.2.3 Links and Nodes Manager

The Links and Nodes Manager (LN) is a system deployment software, a middleware. It aims to provide a clear view of the running modules (viewer, Simulink, vision processing) and the way they are exchanging data. It could be said that LN is the tool that directly helps to group the necessary processes (modules/programs) to launch a mission (experiment). It is compiled and deployed by Cissy, which is explained in the next section.

Fig. 2.12 shows the grouping of processes that the LRM has to carry out for the mission. At the bottom, one can find the Log, where everything that is displayed in the terminal is printed (for example, Python print functions, or errors). At the top part, the user can manage the tabs to navigate to the different properties of the nodes and topics, as well as to analyze their variable messages and other useful tools. On the right side are the tabs to activate or cancel each process. The relevant links, as well as for the next subsections, can be found in Appx. A.

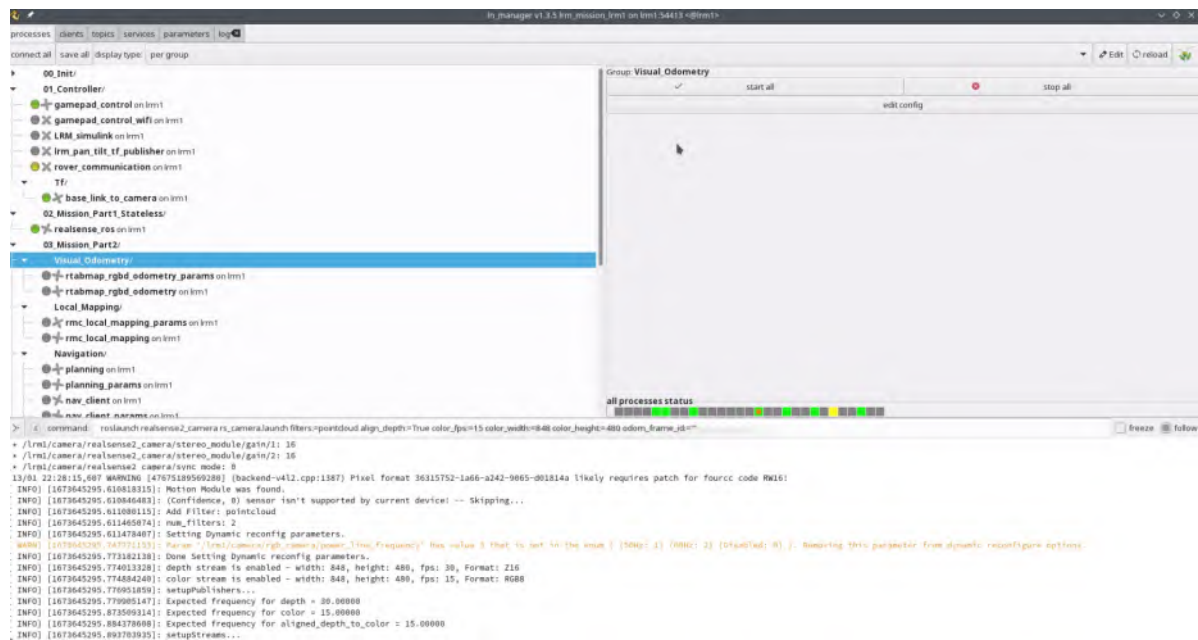


Figure 2.12: LN Manager processes example [13]

2.2.4 Cissy

Continuous Integration Software System (Cissy) is the successor of the old tool Robotics and Mechatronics Package Management (RMPM), which was a system for the handling of software (and other) packages before it was replaced by Conan (another package manager for C/C++).

Cissy provides the main development pipeline for software at the institute. It is a combination of several modern software engineering tools, and its purpose is to smoothen the software development pipeline, using the RMC-GitHub. More information can be found in the DLR relevant links of Appx. A.

Development

This chapter talks about the methods for the development and integration of the LRM modules and subsystems. It is described in detail the way in which the algorithms were implemented until reaching a final functional version.

3.1 Previous Rover Version

In the past, the LRM used to work only in one way: connected to a computer/laptop with USB cable and using the GUI without any remote monitoring tool. Some technical references, such as several voltage lines, were taken from [6], but the integration and documentation were something that completely changed during the development of this thesis.

In order to achieve the primary and secondary objectives stated in Sec. 1.2.1, a list of feature tasks was noted down. Table 3.1 shows these tasks and processes that needed to be completed at the end 2022, where the percentages represented the status progress from previous works at the beginning of the mentioned year.

Table 3.1: List of planned tasks

Previous version feature	Progress	Status description
GUI	100%	Successful development of a GUI for locomotion testing
Driving modes	30%	Partial development of Ackerman, Rotation, and Crab-walk
Data pipeline	20%	Only Middleboard and external computer via USB communication
Antenna range info	0%	Working driving area experiment to be set
Mapping	0%	Not developed yet
Autonomous Navigation	0%	Not developed yet

3.2 New Electrical Power System

Table 3.2: EPS Requirements

ID	DER_EPS_01
Description	The EPS shall use the 14.8V, 99.98Wh LiPo battery as the main power source.
Justification	Apart from being already tested in multiple experiments, this device meets the power capacity and voltage requirements to let all devices operate for approximately two and a half hours.
Flow down from	DER_01
Verification Method	Test

ID	DER_EPS_02
Description	The EPS shall use two TSR2-2490 as the DC-DC converters for the power source.
Justification	In the past, only one was used to condition the voltage input for the other devices. The overall peak current of the LRM was estimated to be ca. 4.0A. Hence, two of these modules in parallel are required.
Flow down from	DER_01
Verification Method	Review of Framework Integration, Test

The main task of the EPS is to deliver power to the instruments, starting with the main power supply which, in this case, can be the battery of Fig. 3.2, or a stationary one powered by a common electrical outlet.

Previously, there were multiple faulty electrical connections, and sometimes the CPU was working very slowly. This often caused the NUC to shut down or make some programs stop suddenly. Therefore, it was assumed that there existed some hardware issues, hence the power budget of Table 3.3 was calculated.

Table 3.3: Power Budget

Mode/Power(W)	EPS	OBC	COMMS	Camera	Actuators	Total—(W)
Sleep	0.1	0.0	0.0	0.0	0.0	0.05
Stand-by	0.35	18	3	0.0	0.0	21.35
Camera-only	0.35	30	3	2	0.0	35.5
Actuators-only	0.35	22	3	0.0	3.5	28.85
All active (avg)	0.35	30	3	2	3.5	38.85

Fig. 3.1 is a representative diagram of all power buses and distributions for the LRM's components. The Lithium-Polymer (LiPo) batteries are well

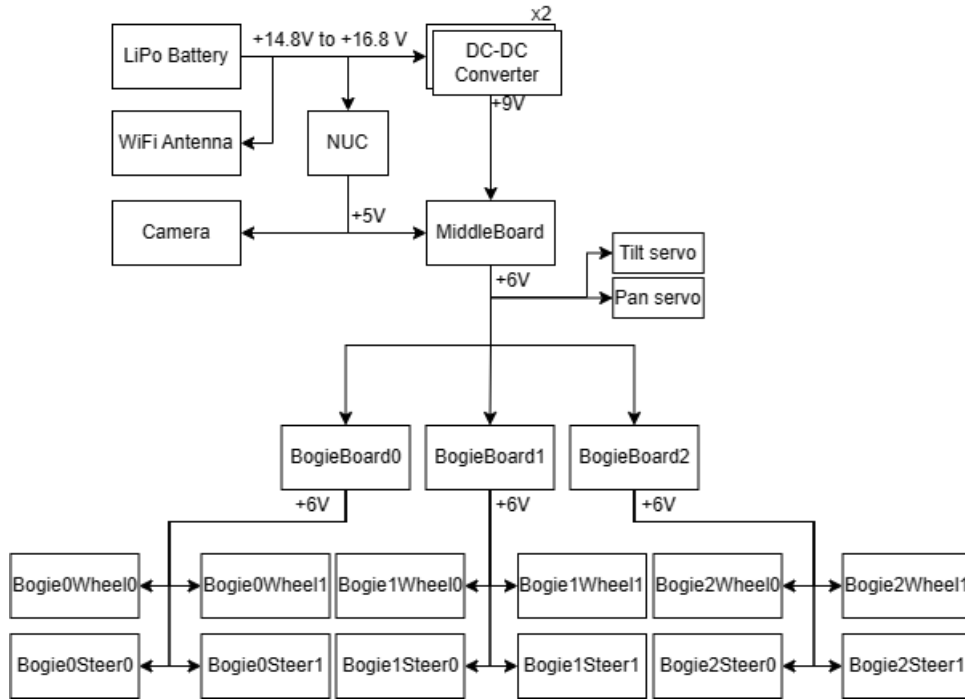


Figure 3.1: Electrical Power Subsystem diagram

known to have great performance in terms of their capacity and power in relation to their weight. Even so, they are more delicate than other kinds of batteries, since their chemical components could be a bit unstable, which could lead to an explosion, corrosion, or human intoxication as their chemical particles harbor a certain danger.

The battery used in this rover version is shown in Fig. 3.2. It is an XTRON LiPo battery [14] with a capacity of 6750 mAH, 99.98Wh, and a discharge rate of 25C, which means that, in order to know the average time the LRM could operate with all devices active, the calculation of Eq. 3.1 was done.

$$t_{operation} = \frac{Ph_{battery}}{P_{rover}} = \frac{99.98Wh}{38.85W} = 2.57h \quad (3.1)$$

However, since some tolerance required to be considered, the estimated time was decreased to 2 hours and 15 minutes.



Figure 3.2: 14.8V-LiPo battery [14]



Figure 3.3: 14.8V to 9.0V DC-DC Converter [16]

Once the battery delivers energy to the WiFi antenna and the NUC which feeds the depth camera, a Direct Current to Direct Current (DC-DC) converter modifies the power bus for the MiddleBoard (the NUC also feeds the MiddleBoard's logic). The DC-DC converter feeds the high-power circuit inside the MiddleBoard, which reduces the voltage to the PanTilt and BogieBoards for the locomotion servomotors.

It was noted that a single TSR 2-2490 [16] DC-DC converter was not enough for feeding all the system, so a doubled device was implemented in parallel. This small encapsulated integrated circuit uses a 390uF, 50V-electrolytic capacitor for signal smoothing.

3.2.1 Graphical Interconnection Diagram

Based on the previous EPS diagram, a more graphical one is shown in Fig. 3.4 to help the developer understand the physical interconnections between the robot components.

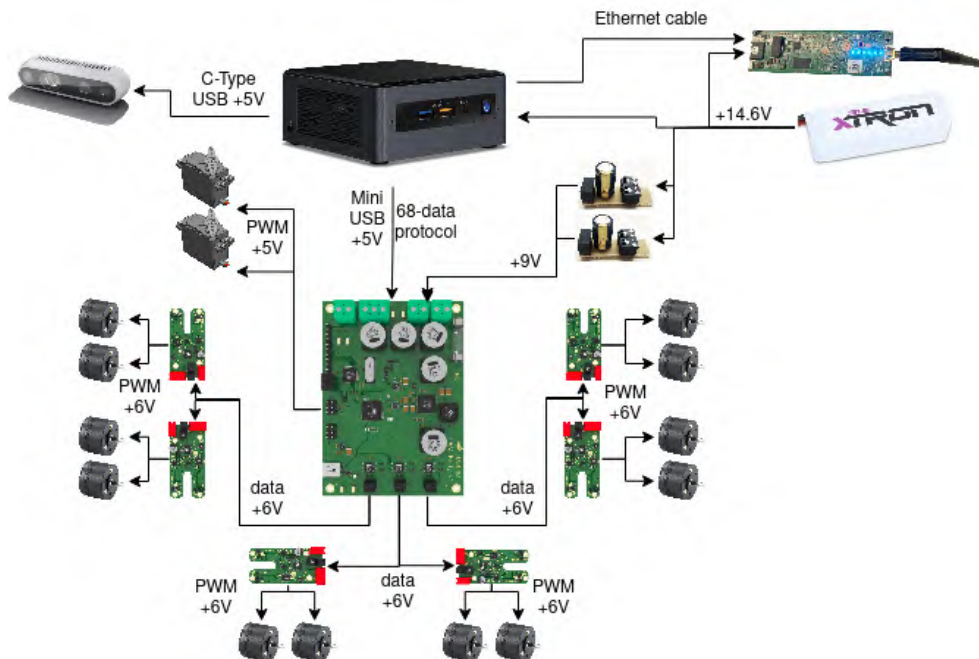


Figure 3.4: Graphical Interconnection Diagram

3.3 Communications System

Table 3.4: COMMS Requirements

ID	OPR_COMMS_01
Description	The COMMS shall use the 2,4/5GHz WiFi communication modules which have compatibility with the RMC computers.
Justification	This eases the hardware selection process by using the already tested WiFi modules.
Flow down from	DER_01
Verification Method	Test

It was defined that the communication had to be through WiFi data reception and transmission since most of the rovers at the DLR operated in this manner.

WiFi Connection

This type of connection is suitable for demo missions and remote local tests. It involves more realistic characteristics, but the data transmission is certainly slower than the one with direct DLR Ethernet cable.



(a) WiFi station rocket



(b) WiFi station bullet

Figure 3.5: WiFi modules

As it was mentioned, the WiFi module of Fig. 2.5 is mounted in the middle of the rover and is connected via a small Ethernet cable to the NUC. This green module, once turned on, seeks to connect to the Rocket module of Fig. 3.5a. All blue illuminated Light-Emitting Diode (LED)s from both

modules are a sign that indicates a successful connection.

Finally, since the Rocket AC module (providing a Ubiquiti WiMAX Protocol, not so important to be described) is not directly compatible with the DLR WiFi (2.4GHz,5GHz) laptops, the Bullet M5 module of Fig. 3.5b is required. A sign that the LRM has been successfully connected is that all monitor LEDs are glowing.

To activate the mentioned connection, Appendix B.1 explains an already tested set of instructions.



Figure 3.6: WiFi Antenna Station

To mount the entire WiFi station, an Ethernet signal distributor is required. In this case, a Netgear G5108 was used with its respective voltage source of 12.0V at 0.5A and two voltage converters (black boxes) shown in Fig. 3.6, which provide the necessary power for the Rocket and Bullet modules.

It is important to highlight that some radar signals can affect the quality and effectiveness of this remote communication. Something that helps to avoid interference is to mount the station on a tripod at a medium height.

3.4 Algorithms for the OBC

The On-Board Computer Subsystem (OBC) is the “brain” of the rover. For new implementations, this subsystem required doing a lot of more advanced tasks, such as handling all modules needed for navigation: PointClouds, image streaming, path planning, 2D and 3D mapping, etc. Here is where the **core** of this thesis development was done.

Table 3.5: OBC Requirements

ID	DER_OBC_01
Description	The OBC shall run the Ackerman, Rotation, and Crabwalk driving modes.
Justification	This provides the locomotion with many capabilities to move in most required directions, which let the rover adapt better to the terrain.
Flow down from	FNR_01
Verification Method	Review of Framework Integration, Test
ID	DER_OBC_02
Description	The OBC shall allow remote manual operation by using the GamePad.
Justification	This allows the user to control the rover and to analyze real-scenario situations.
Flow down from	FNR_02
Verification Method	Review of Framework Integration, Test, Analysis
ID	DER_OBC_03
Description	The OBC shall include the RMC Navigation Stack for autonomous navigation.
Justification	Using this already implemented algorithm, the Navigation Stack integration provides all modules to perform an autonomous behavior.
Flow down from	FNR_03
Verification Method	Review of Framework Integration, Test, Analysis
ID	DER_OBC_04
Description	The OBC shall use LN Manager as the main interface for process administration.
Justification	This open-source software tool helps to unify all processes to show a cleaner launching mission interface.
Flow down from	FNR_04
Verification Method	Review of Framework Design, Analysis
ID	DER_OBC_05
Description	The OBC shall feature all previously used devices: F710 GamePad, MiddleBoard, BogieBoards, D435i camera, NUC, WiFi antenna module, locomotion, and PanTilt servos.
Justification	This eases the hardware selection process by using all existing modules and helps to focus only on the high-level algorithms and integration.
Flow down from	DER_01
Verification Method	Review of Framework Integration, Review of Design, Test

ID	DER_OBC_06
Description	The OBC shall use Simulink as the centralized framework tool for control loops.
Justification	Simulink reduces the complexity and effort of programming compared to using a non-standardized software tool.
Flow down from	DER.02
Verification Method	Review of Framework Integration, Test

3.4.1 Driving Modes

The driving modes, which are based on the ExoMars rover and the DLR's LRU, are mainly three: Ackerman Steering, Rotation, and Crabwalk. Each one has a specific function that helps the rover to move through different terrains and reach a target point. Such driving modes can be used in addition to trajectory planning to avoid obstacles. This opens the panorama of possibilities for better mapping with fewer restrictions.

The four most important variables to be calculated are explained here:

- v_{rover} : linear velocity of the rover, where the coordinate system is at the center of mass of the body.
- v_{wheel} : linear velocity of the wheel, where the coordinate system is at the center of the tire.
- ω_{rover} : angular velocity of the rover, where the coordinate system is at the center of mass of the body.
- ω_{wheel} : angular velocity of the wheel, where the coordinate system is at the center of the tire.

It is important to mention that the three driving modes follow the numbering standard explained in Fig. 2.1b for all programs.

Ackerman

Rudolf Ackerman was the one who discovered and defined this principle early in the 19th century. This concept is the relationship between the front inside tire and the front outside tire in a corner or curve.

The calculations define the application of the geometry for 2- or 4-wheel vehicles and enable a precise turning angle to negotiate a changing curved

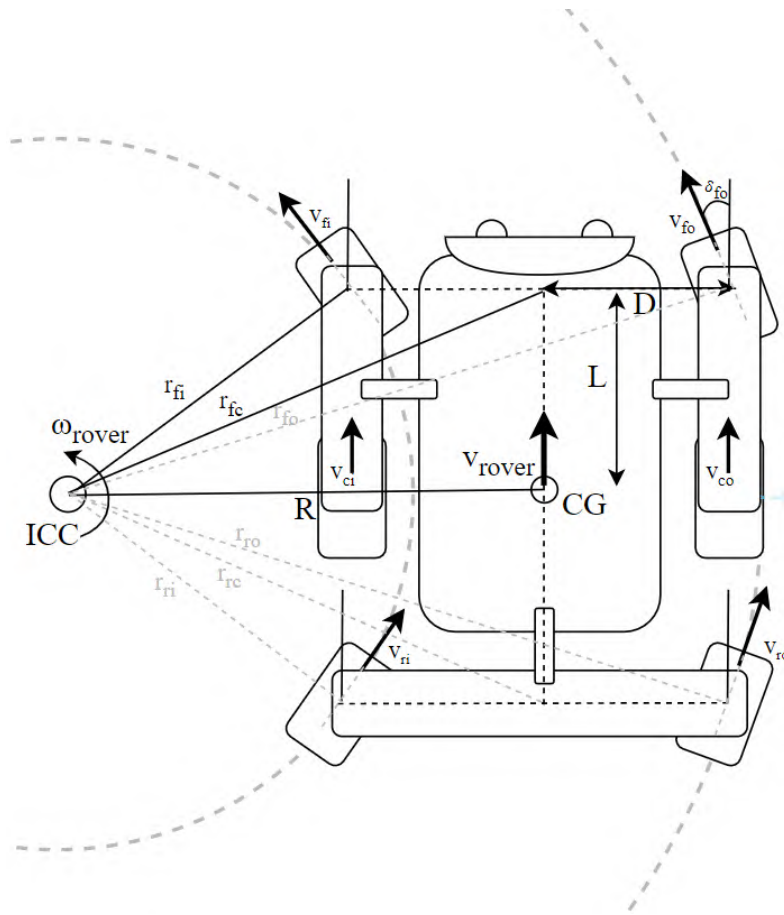


Figure 3.7: Ackerman Steering geometry

trajectory. This concept is to have all four wheels rolling around a common point during a turn. This can greatly improve cornering ability and performance [11].

The steering arms need to be angled to turn the inside wheel at a sharper angle than the outside wheel. Moreover, it allows the inner wheel to track smaller circles than the outer wheel, a relation that prevents the scrubbing of the steer tires.

The inner wheel angle needs to be slightly sharper compared to the outer wheel to reduce tire slippage. This allows softer and improved performance navigation [12].

One can derive the Ackerman Steering equation by considering the

formed angles from the parallel verticals set on each tire center and then adding or subtracting half of the track width ω . After the geometric approach of the Fig. 3.7, each wheel's velocity and steering angle equations are obtained in the following equations (3.2):

$$\begin{aligned}
 R &= \frac{L}{\tan \delta}, \\
 \omega_{rover} &= \frac{v_{rover}}{R}, \\
 r_{fi} &= \sqrt{(R - D)^2 + L^2}, \\
 r_{ci} &= R - D, r_{co} = R + D \\
 r_{fo} &= \sqrt{(R + D)^2 + L^2}, \\
 v_{fi} &= r_{fi} \times \omega, v_{fo} = r_{fo} \times \omega, \\
 \delta_{fi} &= \tan^{-1} \frac{L}{R - D} \\
 \delta_{fo} &= \tan^{-1} \frac{L}{R + D} \\
 v_{fi} &= v_{ri}, v_{ci} = v_{co}, v_{fo} = v_{ro}, \\
 d_{fi} &= d_{ri}, d_{fo} = d_{ro}
 \end{aligned} \tag{3.2}$$

where:

R: turning radius of the vehicle.

L: half of the wheelbase of the vehicle = 0.102 m.

D: half of the tread of the vehicle = 0.085 m.

v_{rover} : reference velocity of vehicle.

ω_{rover} : angular velocity of vehicle.

$r_{fi/fo}$: turning radius of front inner/outer wheel.

$r_{ci/co}$: turning radius of center inner/outer wheel.

$v_{fi/fo}$: velocity of front inner/outer wheel.

$v_{ci/co}$: velocity of center inner/outer wheel.

$d_{fi/fo}$: steering angle of front inner/outer wheel.

As we know, the d_{ci} and d_{co} are zero, because the steering angle never changes, since there is no rotation circle.

Rotation mode

The rotation mode from Fig. 3.8 allows the robot to turn around its own axis. In other words, there does not exist linear, but angular speed. This driving mode is suitable when there is not much space to turn with Ackerman, or when the goal is facing away from the robot.

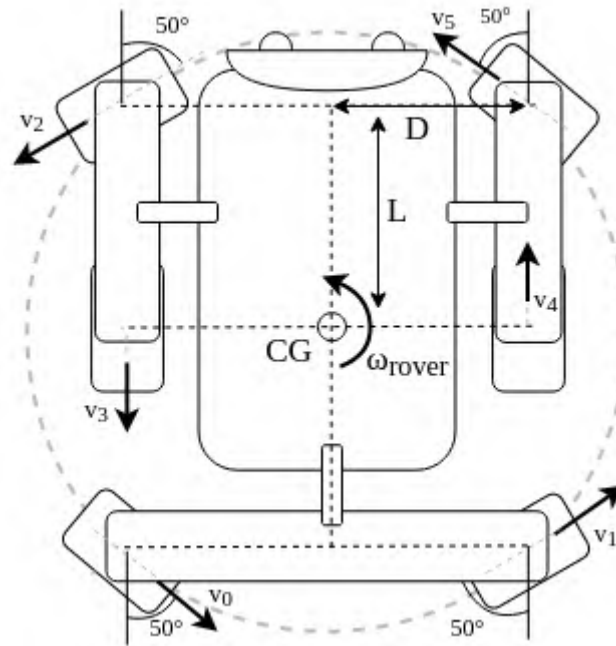


Figure 3.8: Rotation geometry

To precisely compute the rover's angular velocity, the wheel steering angle needs to be calculated first. Since the rotation circle does not correspond to a perfect square but to a rectangle, the corner tires do not lean at 45° . Eq. (3.3) calculates the rotation angle of the wheels.

$$\theta = 90^\circ - \arctan\left(\frac{D}{L}\right) = 90^\circ - 39.79^\circ = 50.21^\circ \quad (3.3)$$

In Eq. (3.4), the steering angle for each wheel is rounded so that the hardware capabilities can command it in an easier way.

$$\begin{aligned} \theta_2 &= 310^\circ, \theta_5 = 50^\circ \\ \theta_3 &= 0^\circ, \theta_4 = 0^\circ \\ \theta_0 &= 50^\circ, \theta_1 = 310^\circ \end{aligned} \quad (3.4)$$

And finally, Eq. (3.5) calculates the angular velocity of the vehicle where all wheels must be commanded by the same linear velocity to avoid hardware issues.

$$\begin{aligned} v_{wheel} &= v_1 = v_2 = v_3 = v_4 = v_5 = v_6 \\ v_{rover} &= 0 \frac{\text{m}}{\text{s}} \\ \omega_{rover} &= \frac{v_{wheel}}{\sqrt{L^2 + D^2}} = (7.532 \frac{1}{\text{m}})v_{wheel} \end{aligned} \quad (3.5)$$

Crabwalk mode

Just exactly like a sea crab, this driving mode from Fig. 3.9 allows the rover to move with only linear velocity by using the same steering angle value for all wheels. This mode is particularly interesting because it provides extra accuracy when the rover is driving within the final goal's neighborhood. When combined with the Ackerman and the Rotation for navigation, this mode can considerably increase the efficiency in various aspects such as mechanical energy and time, as well as getting out of possible mechanical stuck.

The calculation of the linear speed in Eq. 3.6 is relatively simple, and since there is no rotation, the angular speed is reduced to zero.

$$\begin{aligned} \omega_{rover} &= 0 \frac{1}{\text{s}} \\ v_{rover} &= v_{wheel} = (r_{wheel})\omega_{wheel} = (0.085 \text{ m})\omega_{wheel} \end{aligned} \quad (3.6)$$

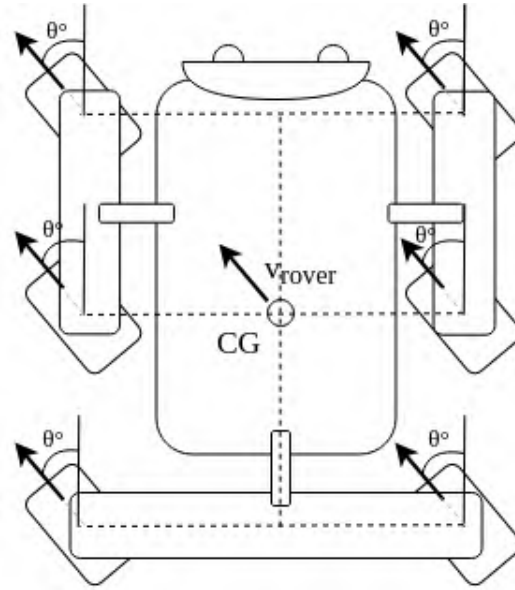


Figure 3.9: Crabwalk geometry

3.4.2 LN World: Breaking Down The Modules

There exist two middleware worlds: Links and Nodes Manager (LN) and Robot Operating System (ROS).

LN is a real-time-capable software tool, while Robot Operating System (ROS)¹, another open-source tool, is a set of software libraries that help developers build robot applications [17]. DLR developers use LN packages for control and required data during real-time communication, and ROS for higher-level processes where quick processes are no longer required.

It could be said that, in the world of LN, the processes are more centralized and are in charge of managing the direct commands for the hardware operation, as well as the management of higher-level language (mainly Python), while, in the world of ROS, the processes are more modular and standardized for all DLR rovers, and have greater complexity (mainly done in C++, XML, and Python). In LN, simpler data strings are handled, such as floats and integers, while in ROS, matrices for image processing, Point-Clouds, maps, and vectors are handled.

The GUI can still be used as an offline mode for testing purposes, as

long as it is connected via USB to a computer. The great advantage of this tool, as mentioned in Sec. 2.2.1, is to have an independent process for control and calibration.

GamePad Controller

The **gamepad_controller** node is the first to act on the process list. Once activated, it will be in charge of recognizing the commands received from the Bluetooth module and transmitting the data to the LN Manager pipeline. In the previous version, there already existed Python developments for the GUI. However, the following improvements have been done:

- variable renaming for better understanding
- joystick filtering for softer behavior
- LN Manager pipeline integration
- printing information for monitoring purposes



Figure 3.10: gamepad_controller node diagram

Table 3.6: Controller_1 message

Parameter	Range or values	Type	Variable description
power	[-1.0, 1.0]	double	Raw wheel power
rotation	[0, 360]	double	Raw steering angle
drive_mode	1, 2, 3	int	Ackerman, Rotation, or Crabwalk
mode	90, 200,...	int	Modes received by BogieBoards, coded in C++
Pan	[0, 180]	int	Angle for the Pan
Tilt	[0, 180]	int	Angle for the Tilt
Permission	0, 1	int	To activate autonomous navigation

Fig. 3.10 shows the data transfer. Firstly, the raw data from the GamePad is sent via Bluetooth to the NUC. Secondly, the Python code processes and maps the data with proper vectors to let the rover understand what

the user wants to do. Finally, the parameter data explained in Table 3.6 is sent to Controller_1 topic via LN Manager.

The GamePad controller code can be found in Appx. D

LRM Simulink

The output of the three controllers: GamePad, autonomous navigation, or an open one, goes directly to the input of the LRM_Simulink, which has been described in Sec. 2.2.2. Here, in Fig. 3.11, the output of the three driving modes is computed and sent to six topics that have data for the locomotion. Furthermore two more topics are provided: PanTilt and Modes.

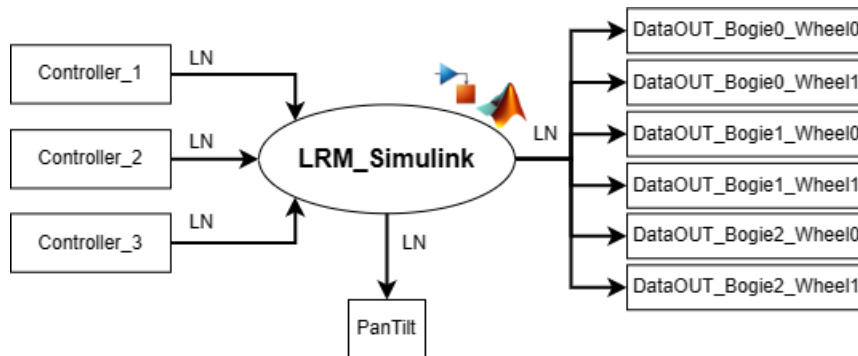


Figure 3.11: LRM_Simulink node diagram

Table 3.7: DataIN_BogieX_WheelX, PanTilt, and Modes message

Parameter	Range or values	Type	Variable description
setMode	90, 200,...	int	Modes received by BogieBoards, coded in C++
setPower	[-1.0, 1.0]	float	Processed wheel power depending on the driving mode
setAngle	[0, 360]	int	Processed steering angle depending on the driving mode
setPan	[0, 180]	int	Movement of the Pan servo
setTilt	[0, 180]	int	Movement of the Tilt servo
setDriving	1, 2, 3	int	Ackerman, Rotation, or Crabwalk
setPermission	0, 1	int	To activate autonomous navigation

The three driving modes are operated here inside three independent Matlab scripts. These scripts can be found in Appx. C.

Rover Communication

The **rover_communication** has a direct link between the locomotion and the body control while it receives the command expressions from the

mathematical Simulink model. **This is probably the most important module of all development of the thesis** since it integrates, processes, and conditions the inputs from Simulink, as well as for autonomous navigation.

Fig. 3.12 shows the data transfer coming from the Simulink model, and the most important variables (from many provided) are the same as the ones explained in Table 3.7.

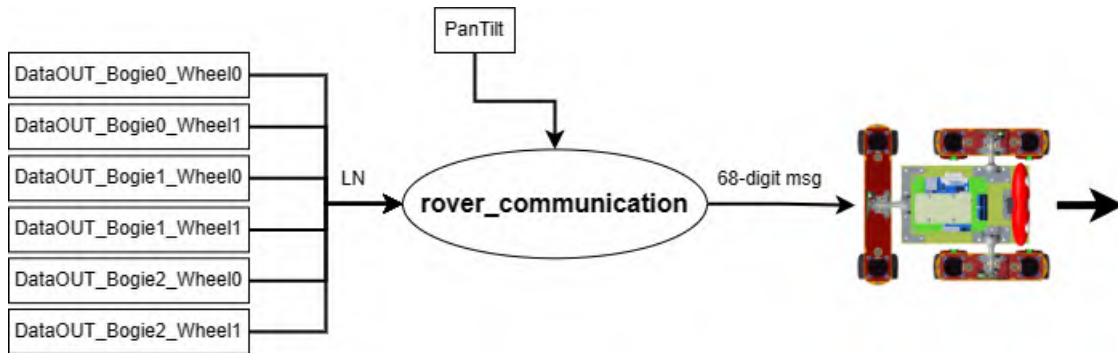


Figure 3.12: rover_communication node diagram

These parameters are sent through the LN Manager pipeline to the rover_communication node made in Python, which processes the input depending on the controller (GamePad or autonomous navigation). Once the right conditioning has been set, the output is sent through a 68-digit protocol message directly to the MiddleBoard, which has a direct command to the BogieBoards and all actuators. the structure of the 68-digit protocol can be found in [13].

From both, manual and autonomous navigation controllers, there is an internal calculation in this node for the body's linear (v_{rover}) and angular velocities (ω_{rover}). As we know, there exist some hardware limitations that needed to be conditioned in the program in order to try to match the autonomous navigation vector explained in Sec. 3.4.3. In other words, the calculations differ from each other when changing the controller.

Two versions were refactored: one with a working Simulink integration (only using the GamePad controller), and another one without Simulink

(which includes the autonomous navigation and the three driving modes inside its own Python code). Both releases are in Appx. E and Appx. F, respectively.

GamePad Values Conditioning

If the green button of the GamePad is not pressed, the ‘permission’ parameter stays as ‘0’. The LRM will be set to GamePad control, which means that all values from the user’s hands will be commanded as desired and delivered to the Simulink model.

From a controlled experiment performed on 09.11.2022, it was found that the maximum linear and angular velocities that the rover could move at were 0.115 m/s and 0.49 rad/s, respectively. If the rover moves only with the GamePad, the python code is receiving a body velocity range of $[-1.0, 1.0]$ (unitary signed floating vector) from the Simulink model, but the MiddleBoard can only process a wheel value range of $[0, 15000]$. The Manual Mode equations (3.7) in m/s for printing purposes and in $[-]$ (non-dimensional) for the MiddleBoard were applied.

$$\begin{aligned} v_{rover} &= (0.115 \frac{m}{s}) v_{simulink} \\ v_{rover} &= (15000) v_{simulink} \end{aligned} \tag{3.7}$$

Each calculation depends on the equations set of each driving mode already explained in 3.4.1. Therefore, it is suggested to the developer to check the Matlab scripts.

In order to not exceed the limit, the velocity value for each wheel was truncated to a maximum of 15000. However, a conditioning implementation to get the final linear and angular velocities based on the hardware limitations were still needed.

For the negative values that the MiddleBoard can not receive, a two’s binary complement [18] was applied for each received value. An example

of the maximum linear velocity is shown in Table 3.8. With this, the LRM could drive in all directions and quadrants of the coordinate system.

Table 3.8: Two's Binary Complement Example

Bits	Unsigned Value	Two's Complement Value	Signed Value
1100 0101 0110 1000	15000	0011 1010 1001 1000	-15000

Some noise from the joystick was noticed and then reduced with a simple filter application that can be found within the `Rover.py` code, and finally, all relevant values were packaged and printed. A Crabwalk example could look like this:

```

W2: 15000   63°   W5: 15000   63°
W3: 15000   63°   W4: 15000   63°
W0: 15000   63°   W1: 15000   63°
drive_mode: 3 rover_mode: 200 pan: 139 tilt: 164 permission: 0
From GamePad:
power_gamepad: 1.000   angle_gamepad: 63°
body_vel: 0.115 m/s   body_w: 0.000 rad/s

```

Autonomous Navigation Conditioning

This was the most challenging part of all the thesis work, since it includes coordinate frames standards, ROS-type Twist vector management, and characterization of maximum and minimum values for ‘if’ conditionals.

When the green button is pressed, the ‘permission’ parameter changes from ‘0’ to ‘1’, and the mathematical context changes completely. The Navigation Stack already provides the LRM with one Twist-message type vector, which includes linear velocity in m/s and angular velocity in rad/s. Therefore, it is no longer necessary to transform these values, but only to adapt the units for the MiddleBoard.

In order to change between the three driving modes and make the navigation softer and more effective, the logic of the vectors from the topic `lrm1.cmd_vel_autonomy` was taken as a reference. Here is a simple explanation of the three driving modes:

1. **Ackerman:** If the linear velocity is commanded in x and the angular velocity is commanded in z (please note that neither v_z , ω_x nor ω_y will

appear), then it is best to set the LRM to this mode since this allows the robot to move more freely by negotiating big circles.

2. **Rotation:** If there is no linear velocity, but there exists ω_z , then it means rotation.
3. **Crabwalk:** If there is only linear velocity in x and y, but no rotational velocity, the path planner is communicating that the LRM is close to its target and it is easier for it to perform Crabwalk (similar to when a car starts parking).

Fig. 3.13 explains the same in a more graphical way.

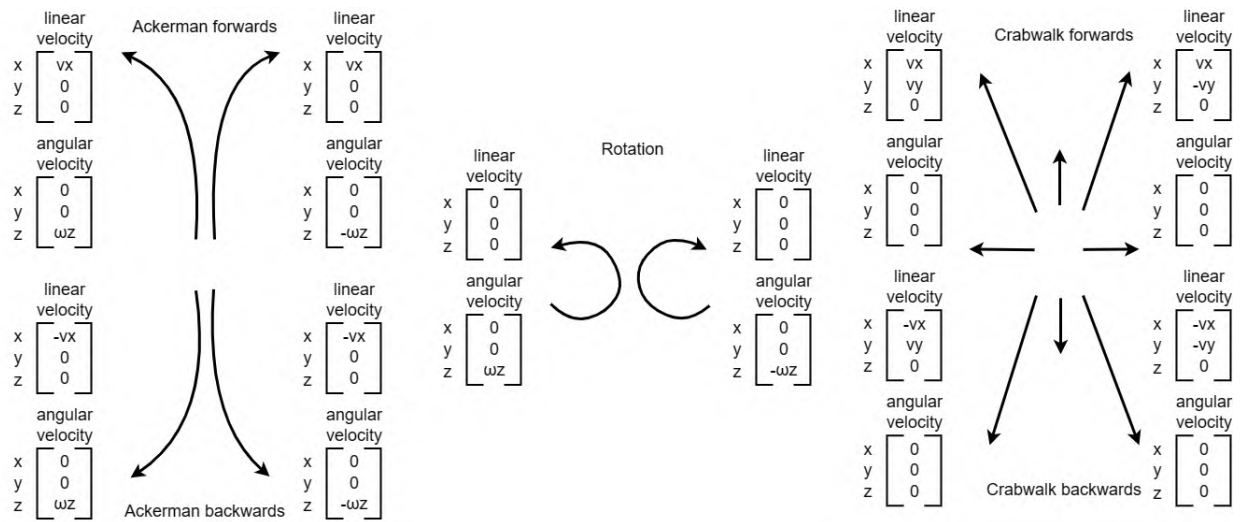


Figure 3.13: Autonomous Navigation: driving modes conditioning

During the experiments, some constants were found and adjusted by which the linear and angular speeds are multiplied (and they are highlighted in the codes), allowing the values to stay within the desired range already explained in the previous section.

It was experimentally found that the Twist vector that the autonomous navigation algorithm provides to the Rover Communication program, in the face of a new planned trajectory as long as its offset is less than ca. 35 degrees from the previous one, the rover can continue operating with Ackerman mode.

However, after 35 degrees, the Ackerman mode exceeds its limit, and the rover stays in a loop, so it was established that, based on the analysis of the driving conditioning of Fig. 3.13, it changes to a Rotation mode, clockwise or anticlockwise depending on the sign transmitted.

And of course, when the rover is close enough to the target (about 30-40 cm away), the algorithm moves only in translation without rotation, indicating that it is best to switch to Crabwalk.

Finally, all relevant values were packed and printed. A Rotation mode example would look something like this:

```
W2: 15000   310°   W5: -15000  50°
W3: 15000   0°    W4: -15000  0°
W0: 15000   50°   W1: -15000  310°
drive_mode: 2 rover_mode: 200 pan: 139 tilt: 164 permission: 1
From Navigation Stack:
nav_vx: 0.000 m/s nav_vy: 0.000 m/s nav_wz: -0.25 rad/s
body_vel: 0.000 m/s   body_w: -0.250 rad/s
```

Please note that the navigation topic can provide negative angular velocities. Due to this, the angle was conditioned to stay within the range of $[0, 360]$, which is the range the servos can only operate with.

Rostopic to LN

The **lrm1_rostopic2ln** node configures the topics coming from the ROS world and transforms them to be used by the LN world. Therefore, the autonomous navigation vector calculated from the Navigation Stack's path planner is integrated into the Simulink model and later to the rover_communication, nodes that are in charge of the locomotion function. Fig. 3.14 shows the data transfer.

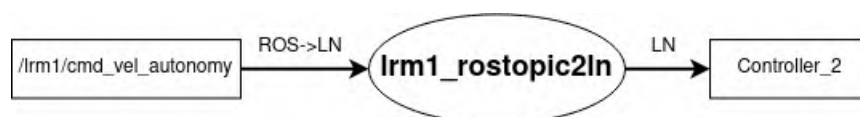


Figure 3.14: rostopic2ln node diagram

LN World: Integration

Once the previous modules have been applied in the LN world, the integration of such processes can be fully shown in Fig. 3.15. This is achieved through the correct use of functions and methods in the LN package, which is stored in the GitHub repositories, managed by Cissy.

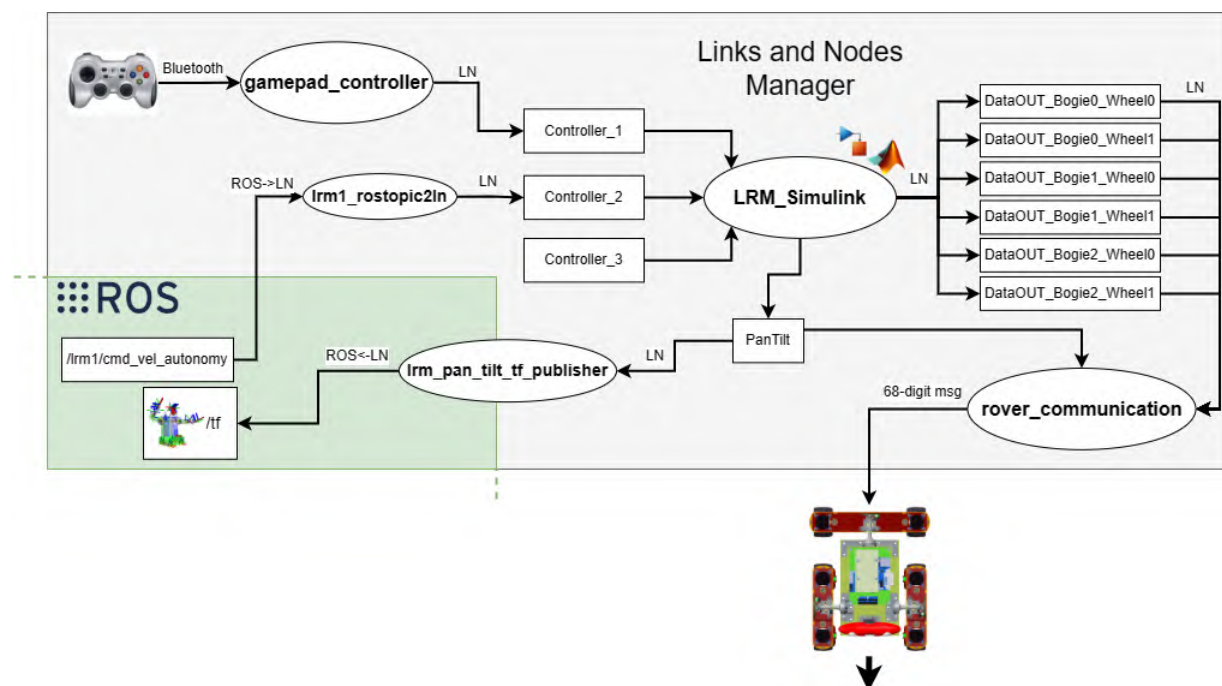


Figure 3.15: LN world integration diagram

The use of LN when is the only world running is a huge advantage because it allows the use of only the Manual Mode without the need to activate other types of controllers, such as the autonomous navigation (in ROS). This is very convenient when only instrumentation and quick locomotion testing are required. This configuration takes from five to ten minutes to be set up, which makes it suitable for rover demos and concept proof.

3.4.3 ROS World: Breaking Down The Modules

As it has been described at the beginning of Sec. 3.4.2, ROS is an open-source tool used as a set of software libraries that help developers build

robot applications [17]. Multiple packages were used in this world and will be described below.

Realsense ROS

The realsense-ros repository contains three packages that allow using the Intel RealSense Depth Camera D435i with ROS: the `realsense2_camera`, `realsense2_camera_msgs`, and `realsense2_description`.

These 3 packages provide messages such as raw rectified image (for RGB image streaming, for example), depth points, etc.

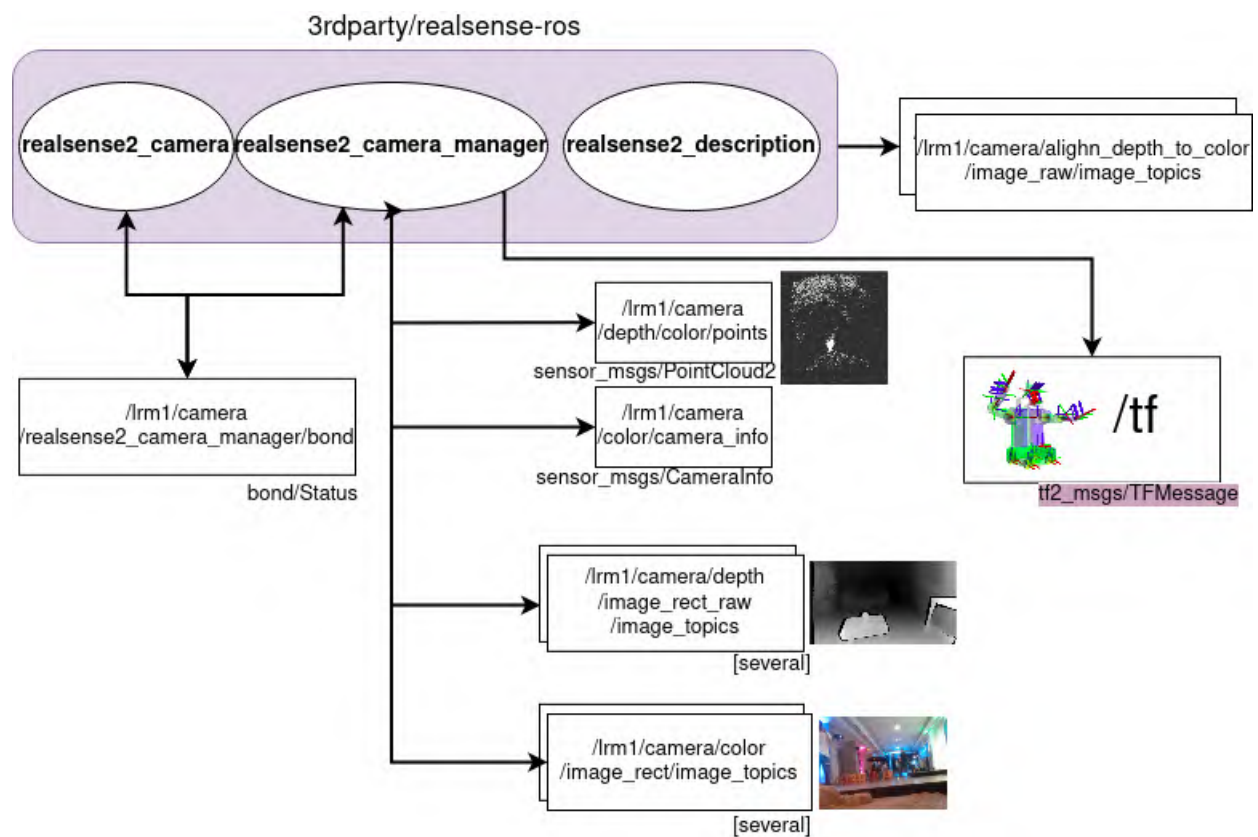


Figure 3.16: Realsense ROS Diagram

Fig. 3.16 shows the data transfer. Note that it is also connected to the Transformation (TF) topic, which allows the user to track multiple coordinate frames over time. The TF package maintains coordinate frame relationships in the form of a time-buffered tree structure. This allows the developer to transform points, vectors, etc., according to the desired

coordinate system [19].

RTAB-Map

The Real-Time Appearance-Based Mapping (RTAB-Map) is an RGB-D, Stereo, and LiDAR Graph-Based Simultaneous Localization and Mapping (SLAM) approach based on an incremental appearance-based loop closure detector, which uses a so-called ‘bag-of-words’ approach to state how likely a new image will come based on a new or previous location. When this hypothesis is accepted, a new constraint is added to the map’s graph, then a graph optimizer minimizes the errors in the map [10].

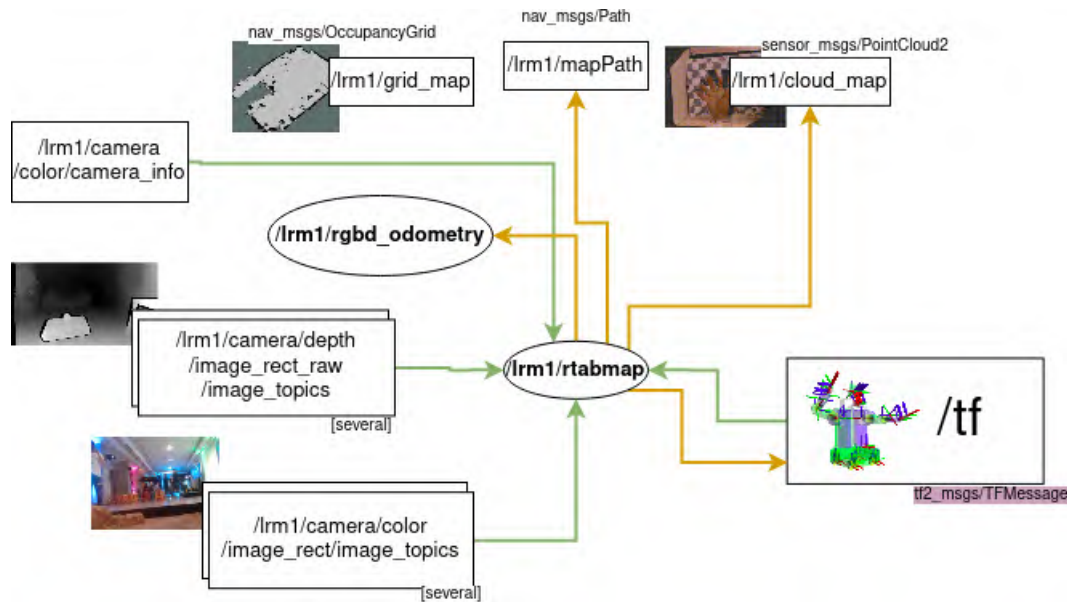


Figure 3.17: Scaled-down OBC prototype with navigation concept and locomotion

Fig. 3.17 shows the data flow inside its package.

Navigation Stack: RMC GBR Navigation

The **rmc_gbr_navigation** package provides local navigation for ground-based rovers, including path planning with obstacle avoidance, motion generation for following a path, and motion generation for driving the robot to a local goal.

Fig. 3.18 shows the data flow inside its package.

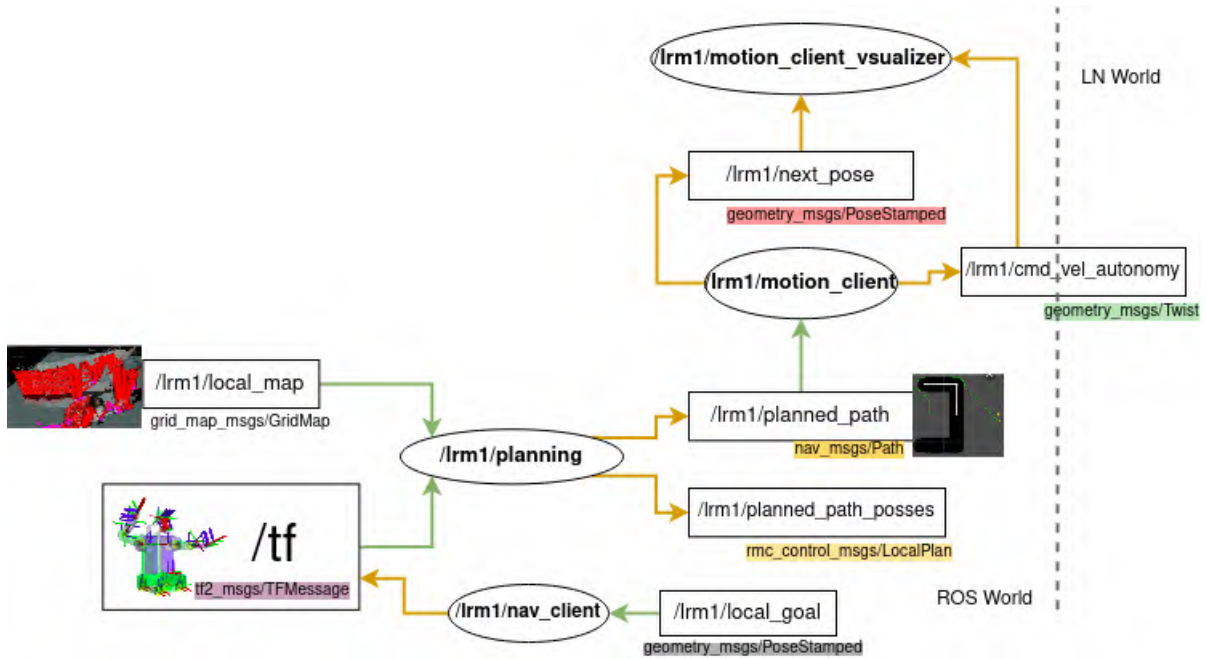


Figure 3.18: RMC GBR Navigation Diagram

It uses a so-called ‘A*’ algorithm, which searches down all path options to the goal and decides upon the most optimal path. After some attempts, as a gap between obstacles is discovered, and it must stop and re-plan from scratch this new optimal path to the goal [25].

Navigation Stack: RMC Local Mapping

This package does the local mapping. It is being developed to be used both on the LRU and SHERP (another DLR project). Currently, the mapping included in the gridmap is restricted to terrain information. The elevation layer (built using only geometric data), semantic layer (color-coded by most likely terrain type), probability distribution layer for first rock type, and probability distribution layer for second rock type, are features that this package provides.

lrm_pan_tilt_tf_publisher

This node, created in Python, is used to relate the frames of the TF tree with respect to the PanTilt servos. This not only gives better accuracy to the mapping but also more precision related to the position of the servos devices under the camera. This allows more dynamic possibilities which

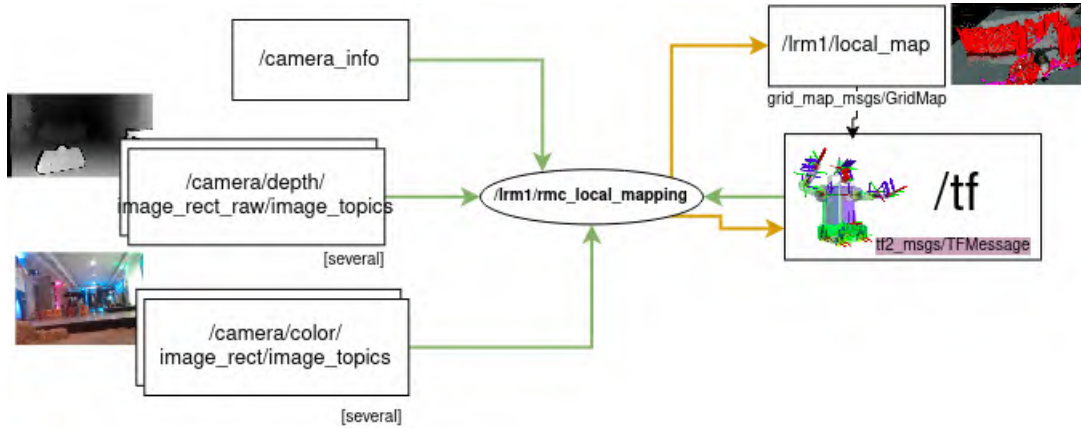


Figure 3.19: RMC Local Mapping node diagram

are to be explored and tested G.



Figure 3.20: lrm_pan_tilt_tf_publisher node diagram

3.4.4 Final Data LN-ROS Pipeline

All modules, including the /lrm1/rgbd_odometry from the Navigation Stack, are integrated into this single diagram of Fig. 3.21. Each module was individually verified by the author with guidance from some DLR colleagues at the institute. With this, the task of module and subsystem development and integration was completed.

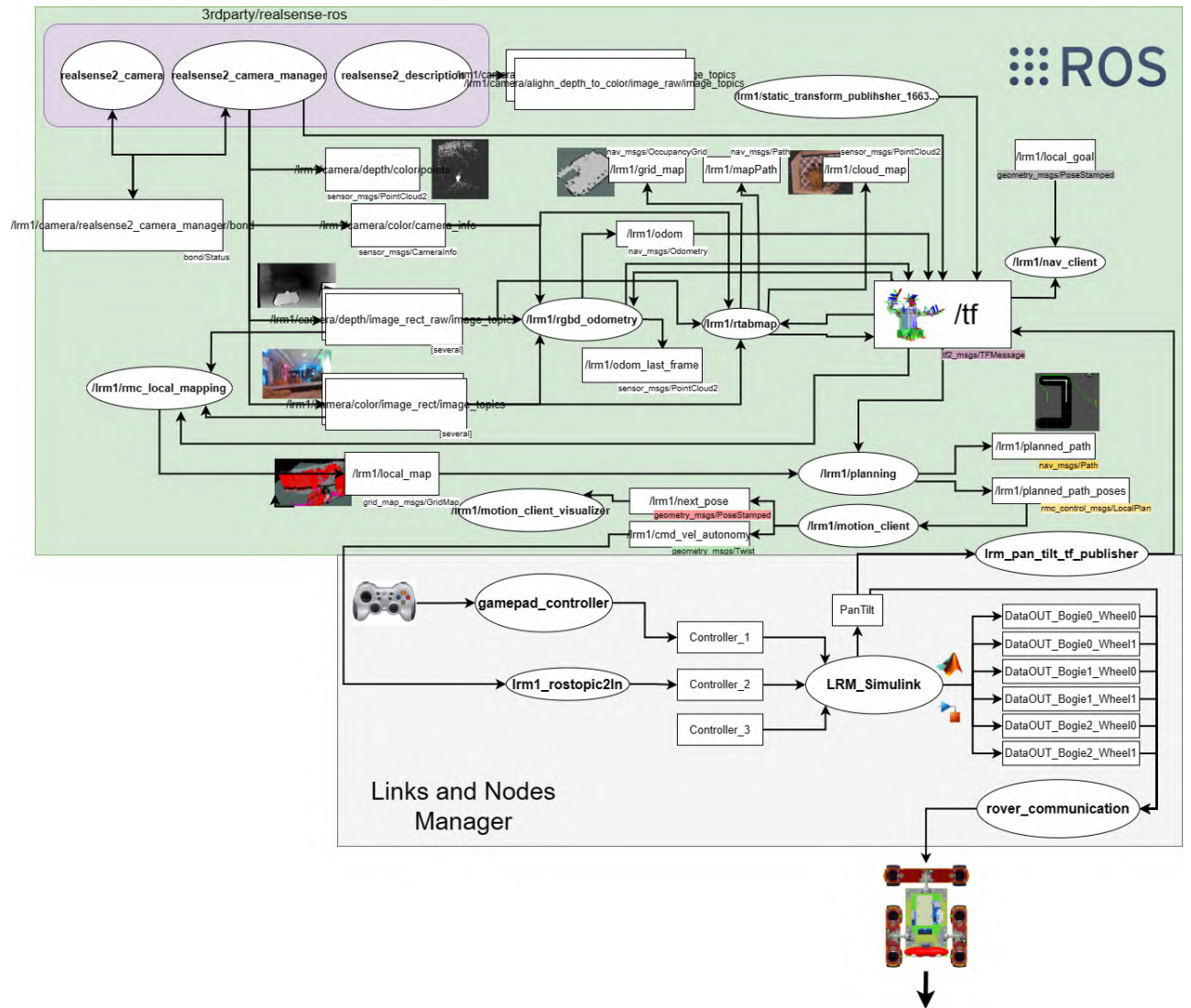


Figure 3.21: Final LN-ROS data pipeline scheme

System Performance

4.1 Electrical Behavior

Looking at Figs. 3.1 and 3.4, the analysis will be done from top to bottom of the block diagrams. Starting with the LiPo battery, it was able to operate as calculated in Eq. (3.1) for approximately two hours and fifteen minutes (+/- 15min). This means, that the camera streaming and the state of communication between devices were maintained without interruption during that period. After the calculated hours, the voltage might drop below thirteen volts and could generate current variations throughout the system, causing sudden blackouts in the NUC or the camera and poor quality of locomotion movement. Therefore, it is recommended to have at least one backup battery, so that after a proper shutdown of the system (see Appendix B) the power source could be changed immediately and the experimentation could have a break for not more than five minutes. This proves that the assumed power budget was precise.

On the other hand, if the user does local testing, the 2.1A power supply can always be used, which has about the capacity of an average laptop charger. The author suggests using the batteries only in required remote experiments.

The PanTilt servomotors pointed at the commanded angle. However, the MiddleBoard did not always provide enough power when the user logged in from the NUC, therefore an electronic design inspection should be done in the future. It is believed to be an internal issue in the PCB itself, the author suggests.

The setup of the two DC-DC converters in parallel was the solution to avoid sudden system blackouts, in addition to providing enough power to the locomotion to not stop while operating remotely (a fully-charged battery). It is very likely that the 2.0A capacity of the TSR 2-2490 [16] is not enough to handle all system's current peaks. Taking into account the numbers from the calculated data budget, it is estimated that it is necessary to implement a DC-DC converter of a minimum of 3.4A of capacity in the future if only one is required to be used.

4.2 Communication Behavior

The WiFi antenna and all other components performed as expected. There were found some radar-related signals interfering with the LRM, but in most occasions, the entire communication system was established almost automatically once the IP of each device was known. There were not too many setbacks that prevented experimentation throughout the day.

4.3 Low-Level Data Handling Behavior

The communication between the MiddleBoard and the BogieBoards was never interrupted, and as long as the tires were mechanically tight, no faulty movements were found. However, for two tire servos (not the steering ones), it is believed that there is a bug in the low-level C++ code in the BogieBoard's PCBs, because the buffer exceeds its maximum raw value (ca. 15000), and the servo keeps moving even though it is commanded to stop. This can be fixed by reviewing the code of the BogieBoards and flashing again the ones that are faulty.

Tests Results & Analysis

5.1 Vulcano Summer School

Table 5.1: Validation Requirement 1

ID	VAR_01
Description	The LRM's locomotion shall be tested on rocky terrains, such as the ones at Vulcano island.
Justification	This validation makes possible to qualify and quantify how suitable the prototype is for more realistic terrains.
Verification Method	Test, Analysis

The main objective of the Vulcano Summer School (VSS) was to carry out locomotion tests on more realistic terrains and share the project results with other students and researchers. Most of the participants had a background in geology or research in astronomy, physics (magnetic fields of certain rocks), or image processing, thus it was an excellent opportunity for the team to show some progress in the field of planetary exploration robotics. This knowledge complement formed a diverse atmosphere of science and engineering.

Table 5.2: Intermediate progress

Feature	Progress	Description
GUI	100%	It is now kept as a testing and calibration offline tool
Driving modes	100%	Fully functional Ackerman, Rotation, and Crabwalk
Data pipeline	60%	Partial integration of some high- and low-level modules
Antenna range info	100%	Antenna ready for testing within the 50m-coverage radius
Mapping	80%	Some first mapping tests performed with the camera
Autonomous Navigation	20%	Navigation concept only tested with the move_base ROS node

Based on the preparation for the VSS, Table 5.2 represents the achieved progress at that time. As can be seen, almost all tasks are completed, except

the autonomous navigation tuning and calibration. Thus, it was planned to focus more on the driving modes execution, and later, on the Navigation Stack.

5.1.1 Driving Modes In Real Environments

The three different driving modes, along with their equations already explained in Sec. 3.4.1, were successfully tested on three different terrains (Fig. 5.1): on the dried “Moon Lake”, on a “Martian” surface at La Fossa (top of Vulcano), and at the beach.



(a) Dried Moon Lake



(b) Martian Surface



(c) Beach

Figure 5.1: Terrains at Vulcano island

Results:

1. **At the dried Moon Lake:** the Rotation and Crabwalk modes performed flawlessly, basically without any mishaps, preventing the robot from getting stuck on the ground. No significant mechanical losses due to sliding were perceived, except for the Ackerman mode, which sometimes had trouble making a proper turn while moving forward at the same time. This was mostly due to the reduced force value of the servomotors that Eqs. 3.2 scaled down. This could easily be fixed by amplifying a bit (estimated 20%) of the top speeds achieved when rotating.
2. **At the “Martian Surface”:** as in the dried Moon Lake, the Ackerman mode presented some issues since on the Mars surface existed larger rocks. The Rotation mode had no issues at all, but the Crabwalk was only able to navigate paths where there were obstacles less than

6cm in diameter. The mechanics, together with the speed commanded for the tires could manage to stabilize the camera and adapt to the terrain as long as it could face smaller rocks than 6 cm. On one occasion, a 15cm rock almost causes breaks in the 3D-printed chassis of one of the bogies, and the robot began to “dig” in its own spot.

3. **At the Beach:** it can be highlighted that all driving modes worked 100% in this terrain. Certainly, rovers hardly would find this kind of viscous (wet) sand in space, or at least not in known places nearby. The surface seemed to “mold” to the tires, but at the same time, the tires adapted to the terrain. It was noted that, as long as the LRM traveled at a maximum of $3/4$ of its highest speed, the rover could be easily maneuvered in all directions.

5.1.2 Mapping of the Dried “Moon Lake”

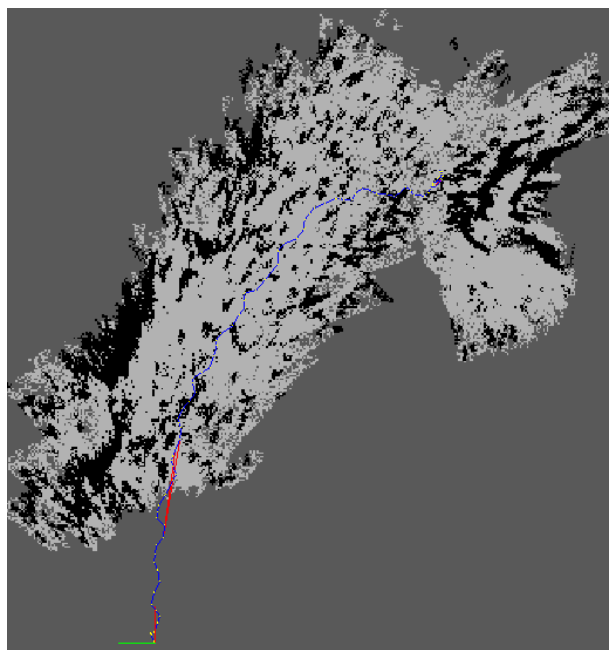


Figure 5.2: 2D Map of the dried Moon Lake

The LRM was manually operated to reach ca. 35m away from the WiFi antenna, first in a straight line, and then turning clockwise until coming face to face with a 1m-high rock.

This test was carried out by loading a Global Positioning System (GPS) weighing 800g, in order to also test how much extra load the robot could carry in case it was decided to attach a robotic arm on it in the future.

Due to security measures and the toxic gases from the sulfur that descended toward the place of the experiment, only a few experiments could be carried out. These tests helped to generate a 2D map of the area, which is shown in Fig. 5.2. Its initial

position and trajectory can also be observed.

The meaning of all the colors is explained below:

- Rover's initial position: coordinate system, **x on green**, and **y on red**
- Followed path: **blue-yellow**
- Explored area without obstacles: **light gray** (almost white)
- Explored area with obstacles: **black**
- Unexplored area: **dark gray**

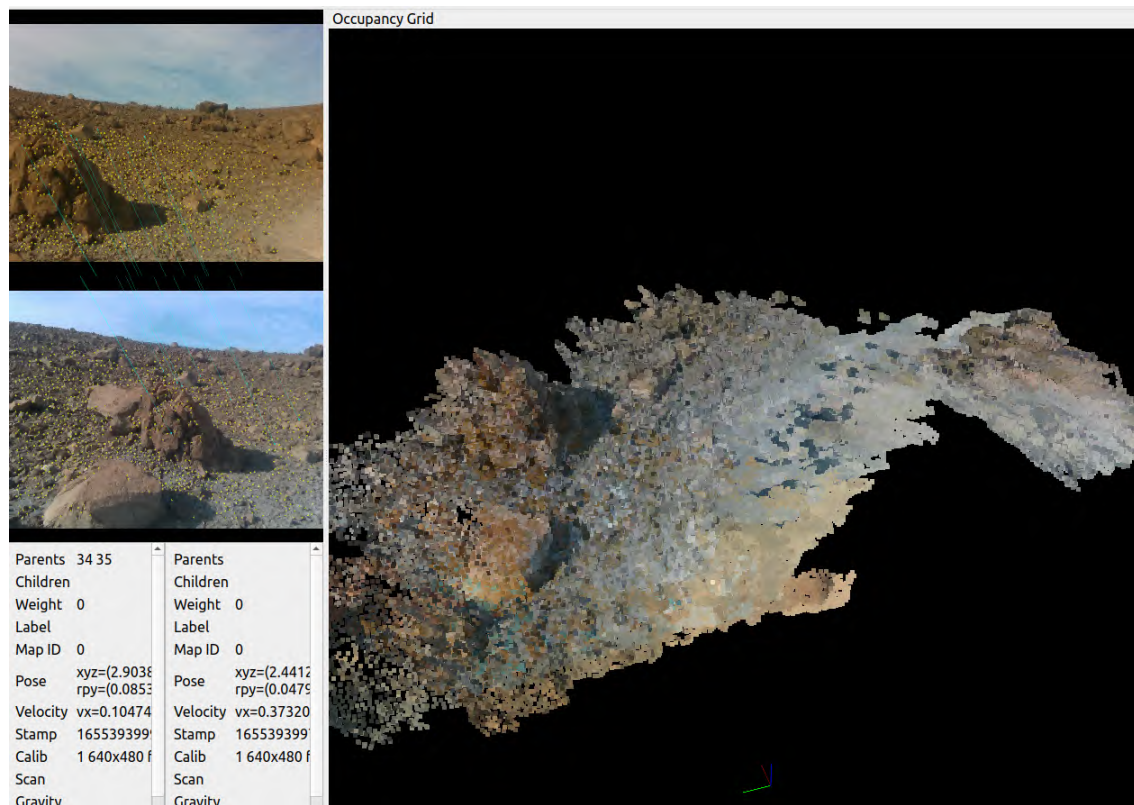


Figure 5.3: Occupancy Grid of the Moon dried Lake

It was also possible to generate the 3D map shown in Fig. 5.3, using the depth data, then converting it to a PointCloud, and finally, performing SLAM as explained in Sec. 3.4.3 to create the occupancy grid.

In the upper left of the image, the depth points that the D435i generates from the image stereoscopically can be appreciated; the top image is the right camera, and the bottom image is the left camera.

The correlation between the 2D and the 3D map matches, and the user can easily identify the place where the experiment was performed. Certainly, it took a lot of movement from the PanTilt servomotors and a long journey to really generate something recognizable. It should also be remembered that within the depth cameras in the market, the D435i is in an intermediate range, which means that its range of vision goes from 30 cm to 3 m, but with a surface accuracy of 2% when the objects are less than 2 m away and with good light conditions. This dried lake is a wide place with obstacles varying in size and concentration. Hence, and based on the above explanation, these results can be classified by the author as “acceptable”.

5.1.3 WiFi Connection Results

During the WiFi antenna experiments, the maximum working range was around 50 meters in radius. This means that the LRM can receive and transfer data within this radio. The data exchange could always be data chains, commands, PointClouds, video streaming, etc.

The steps of Appx. B.1 are effective in every launching mission, as long as the operating system or network updates do not change too much. In any case, one could always go to the Information Technology (IT) department of the RMC for some support.

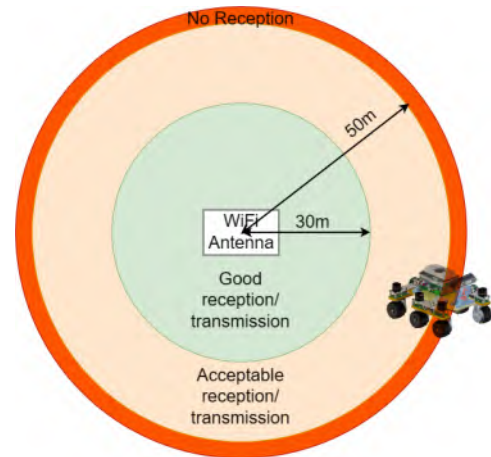


Figure 5.4: WiFi Antenna Range

5.1.4 Presentation

Finally, a video was made to show the rover driving on different terrains, and the team presented a poster (Fig. 5.5) showing the results obtained during the tests. This information was shared with the audience, where students, members of the DLR, and even supervisors of the ESA were present.



Figure 5.5: LRM's poster at the Vulcano Summer School

5.2 Festival der Zukunft

Another place where these rover concepts could be demonstrated was at the Festival der Zukunft at the Deutsches Museum, where a 3x4m environment was used as the testbed.

The setup that was presented during the three days of the exhibition is shown in Fig. 5.6. Three models were shown: the LRU (the big rover),

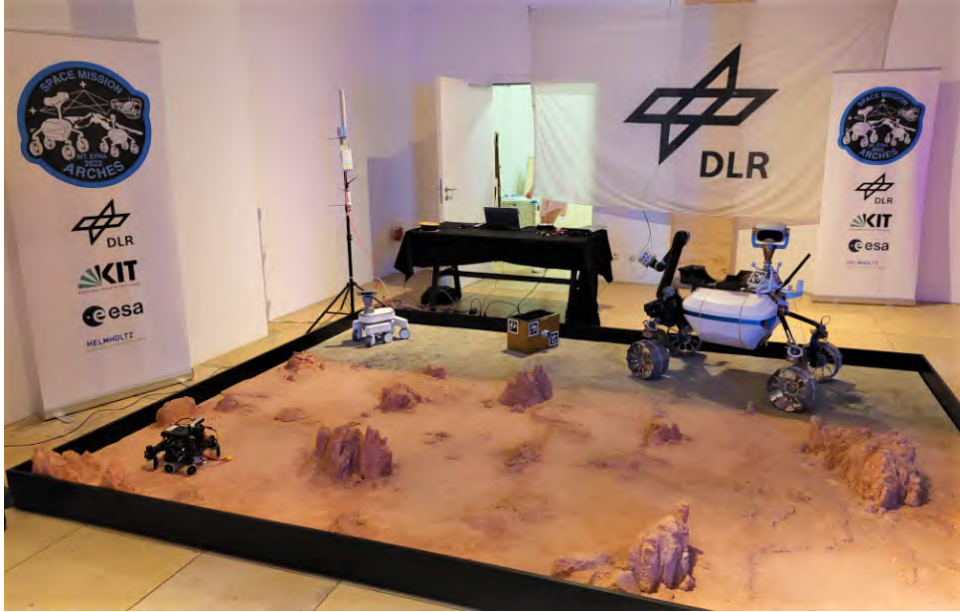


Figure 5.6: Testbed at the Deutsches Museum

the LRM black model used in this thesis (bottom left), and the new white chassis that is intended to be the new version of the LRM in a near future (the one in between).

5.2.1 Mapping - Deutsches Museum Testbed

The testbed mapping was done in about 15 minutes, taking care that the rover did not move too fast, as there was something called IMU drift. The camera's gyroscope, which measures rotational speed, has two complications: this drift and an integration error [21].

This is why another component is fused: the accelerometer, which measures linear acceleration (gravity vector) and helps to compensate the drift stability with some approaches from the state of the art. The camera faces some big challenges when generating accurate odometry if the movements are drastic. A more precise calibration would be the key to improve its performance, but even after carrying out some standalone tests, it was always best to perform slower maneuvers to ensure the stability of the self-localization of the LRM in 3D space. It was found that, after 70% of its maximum linear and angular velocities, which would mean $(0.7)0.115\text{m/s} = 0.0805\text{m/s}$ and $(7.532\text{ 1/m})0.0805\text{m/s} = 0.6\text{ rad/s}$ (rotation based on

Eq. (3.5)) respectively, the drift starts to get accumulated and therefore rises a self-localization problem. Operating under these values will ensure the successful performance of the rover.

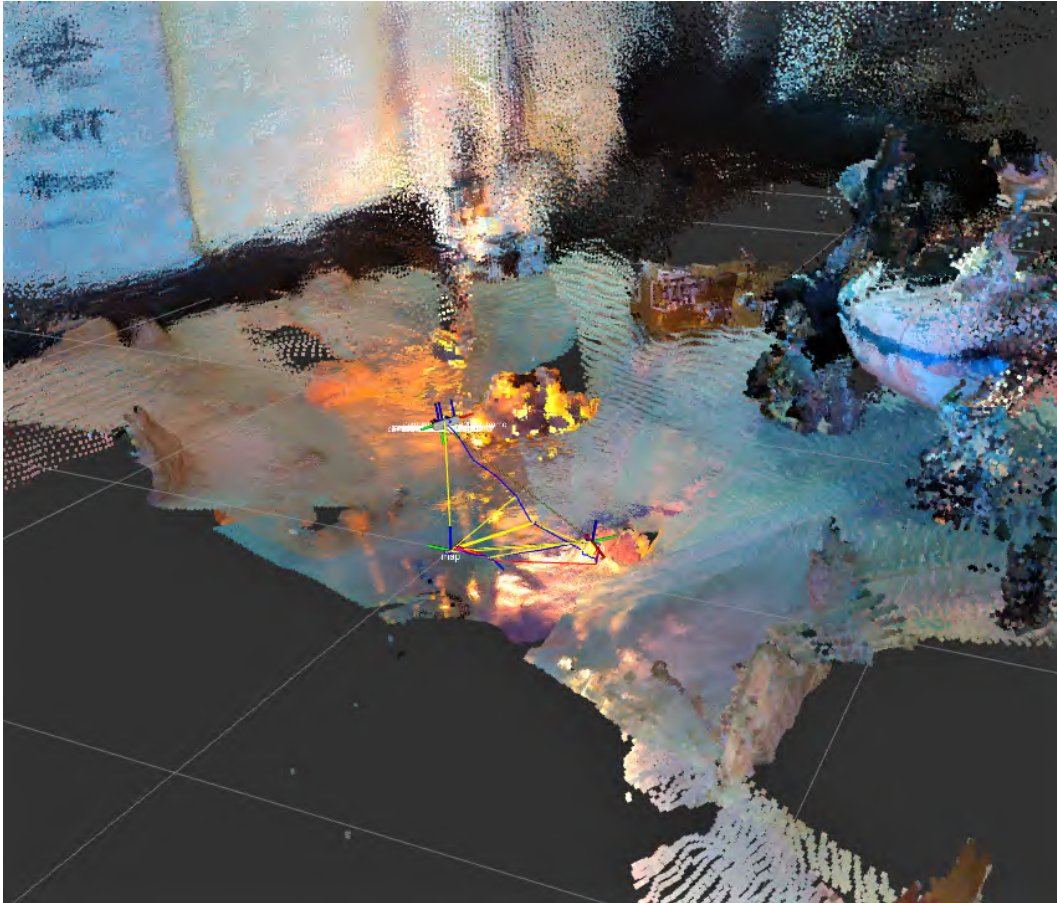


Figure 5.7: PointCloud of the Deutsches Museum's testbed

In Fig. 5.7, a large part of the LRU can be seen in the SLAM. Some people in the audience were able to identify some perspectives of the terrain by just looking at the displayed mapping on a big monitor. Some kids and young students were even able to control the LRM by just using the GamePad and watching the screen on the wall.

5.3 Space Demo Mission

In order to validate the operation of the systems and the mathematical theory, three experiments were carried out in a controlled environment on a $10 \times 5.5m^2$ testbed at the Planetary Exploration Laboratory (PEL). The

soil material was black sand (type of sand), somehow a similar terrain as it was in the VSS and the Festival der Zukunft, but this time with all systems and the autonomous navigation integrated.

Table 5.3: Validation Requirement 2

ID	VAR.02
Description	The LRM shall be tested on an artificial testbed that simulates a Lunar environment.
Justification	This validation will provide an opportunity to analyze more specific behaviors in a realistic controlled environment.
Verification Method	Test, Analysis

The results of such experiments, which include the generated maps, success rate, performance, and limitations, will be described in this section.

5.3.1 Indoor Evaluation Setup

Firstly, the initial and final positions of the LRM were defined. Some obstacles, the “moon rocks”, were placed relatively randomly, and the final setups for the three different experiments were defined.

First experiment: the Manual Mode with the GamePad was proven with completely random-placed “moon rocks” to be crossed in order to reach a final goal by only using the camera streaming and the map from the laptop. This means: without direct visual contact with the rover. During these first six attempts, the rover’s vision was pointing at an angle of 65 degrees out of phase to the target, with a final goal distance of 3.9 m. Fig. 5.10b shows the setup of this environment arrangement.

Second experiment: the first setup of autonomous navigation was tested. The total distance to the target was the same, 3.9m, but the initial pointing angle was set to 165 degrees out of phase to the final goal. The environmental arrangement was similar to the previous one.

Third experiment: this last test also had the goal of proving the autonomous navigation concept, but with a different arrangement: 3.53m

of the total distance to the target, and with 170 degrees out of phase. All objects were moved to different positions.

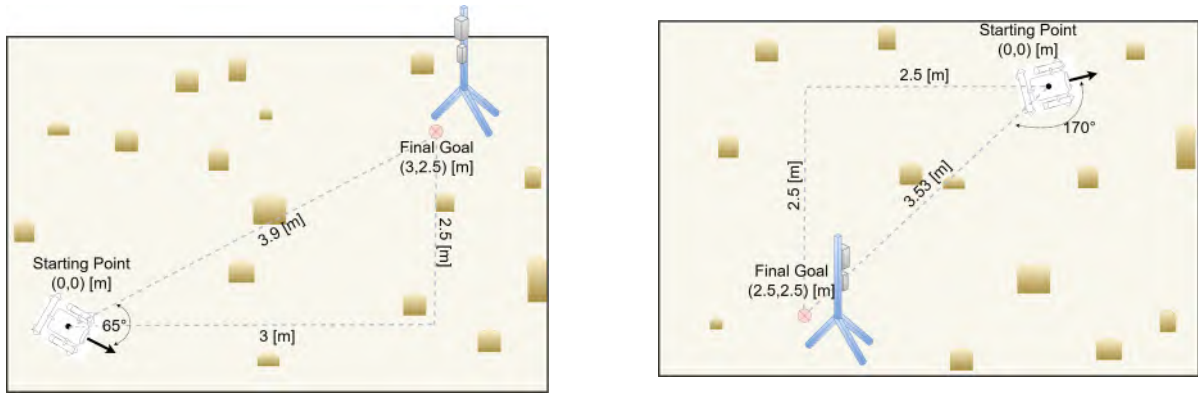
For the execution setup, the steps below were followed:

1. All the instrumentation of the rover was ensured to be properly connected, including the WiFi antenna, since this was pretended to be a completely remote-operated mission.
2. A fully charged 3.2 battery was connected to the main power input of the rover, and the NUC was powered on.
3. The rmc-lx0431 laptop was turned on, which was configured with the explanation of Sec. 3.3.
4. The WiFi station was placed in a stand, connected to a power source (it could be a battery or a direct socket). Then, it was turned on.
5. Once the WiFi connection has been secured, the series of commands of Appx. B were executed.
6. Once inside the LN Manager window, each group of processes was run from top to bottom by double-clicking or using the tick button.
7. When the ROS Visualization (RViz) was opened, and after ensuring the mapping was being collected, the goal was manually placed on the virtual map, also forcing the final direction the LRM was pretended to “park” to. With this, the LRM began to navigate autonomously.

In the sense of integration and validation, Table 5.4 indicates that all tasks have been completed.

Table 5.4: Final progress

Feature	Progress	Description
GUI	100%	It is now kept as a testing and calibration offline tool
Driving modes	100%	Fully functional Ackerman, Rotation, and Crabwalk
Data pipeline	100%	Full integration of high- and low-level processes
Antenna range info	100%	Antenna ready for testing within the coverage area
Mapping	100%	PointCloud, mapping, self-localization
Autonomous Navigation	100%	Integrated Navigation Stack with odometry and path planner



(a) Manual Mode & Autonomous Navigation 1

(b) Autonomous Navigation 2

Figure 5.8: Space Demo Mission setups

The mission was run six times per experiment setup, thus having a total of eighteen attempts for the whole mission. Fig. 5.9 shows a photo from the first environment arrangement. The blue tripod was established to be the final goal, and the rover can be seen at the left-bottom part of the image. There were two rocks in between that prevented the direct passage of the LRM towards its goal, so that a video could show the autonomous navigation performing.

The final success rate results can be analyzed in Fig. 5.10. The rover achieved 100% success in Manual Mode tests, which included only the use of the GamePad, the remote streaming of the camera, and the visualization of SLAM on the laptop.

The autonomous navigation experiments also had a high success rate, but considerably lower than Manual Mode. The results of the first autonomous mission are better than those of the second one. Although it is quite difficult to achieve 100% in all cases since mobile robot algorithms are still under development in many institutes nowadays, the author suggests some possibilities for accuracy improvement to reach the final goal in Sec. 5.3.3.



Figure 5.9: Space Demo Mission experiment

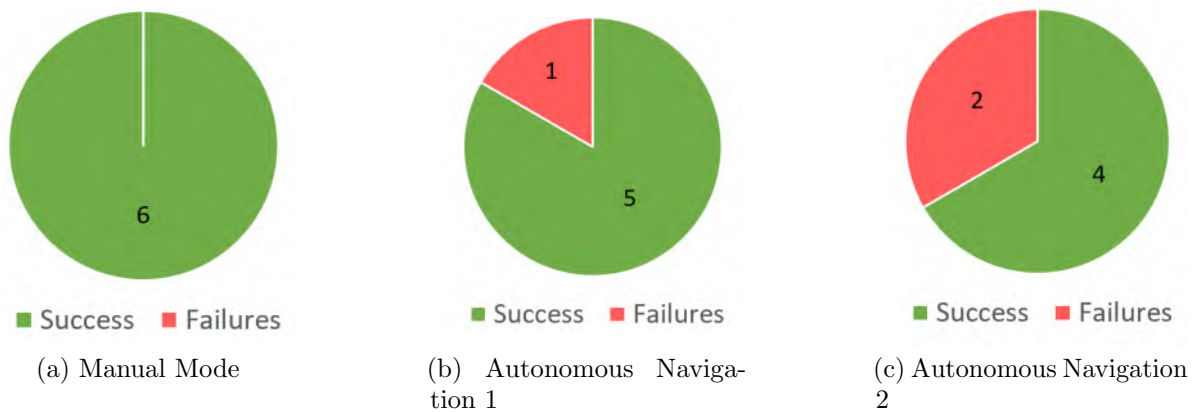


Figure 5.10: Experiment Validation Results

These experiments show an average success rate of 75%. Some authors such as [23] and [24], suggest that an autonomous navigation algorithm with all sensors fused, proper tuning, and accurate calculations, can begin to be considered effective from 71%-80% of the successful tests. The number of attempts is suggested to be as many as possible, and even more, if artificial intelligence is implemented. If this was the case, dozens of them could be needed.

Nonetheless, based on what author Chandra says in [22]:

“The discussion about autonomous operation shows that there is no single sensing solution that is sufficient. Multiple sensor data combined with the sensor fusion method is suggested for autonomous navigation and path planning. The challenge is that different autonomous operation requires a different level of localization precision, which further raises the selection of technique. Hence, sensor selection and sensing technology need research that considers power consumption, computational complexity,...”

All this analysis is relative since it also depends a lot on the navigation targets, the traveled distances, the context (if it is space-related, or navigation here on earth), the environment, and other variable factors.

5.3.2 Manual Mode

As it was done during the VSS tests, the driving modes did not have any critical problems, since this type of slightly rocky sand at the PEL testbed was quite similar to that on the dried “Moon Lake”, with certain characteristics also similar to those of the beach sand.

The maximum reached linear speeds of the tires (0.115 m/s), the entire body velocity (depending on the selected driving mode), and the small pauses that the rover had throughout the journey were solely based on the operator’s skills, pauses which lasted no more than 1.5 seconds.

It is interesting to note that the pipeline of LN and ROS, the laptop, and the WiFi antenna, generated a delay very similar to an operation on the Moon, which is an average of 2.56 seconds based on real experiments from the Apollo missions [20].

Taking these 2.5 seconds delay considerations into account and with no more than the laptop screen to run the remote mission, the final goal at (3,2.5)m was reached in 1:33 minutes or 93 seconds, a pretty good result, the author suggests.

This is not an absolute measure or completely proportional to other

missions from other rovers and therefore cannot be directly compared, as it depends very much on how adequate the GamePad commands were used by the operator to avoid obstacles.

The Manual mode concept was successfully validated, and there were no major collisions or delays between each drive mode change.

What was a bit difficult to understand for the operator at some points of the path, was the orientation of the robot, since when turning with the Rotation Mode, being quite fast, it was not possible to appreciate how much the LRM had turned, and the camera had to be moved with other commands for reconnaissance of the area.

In any case, this was not a problem. The carried out odometry by the RTAB-Map package and the display on Rviz helped this map perception, and it was possible to correlate the orientation of the robot with the image in the streaming, which was quite accurate to the reality: not greater than an offset of 5 degrees.

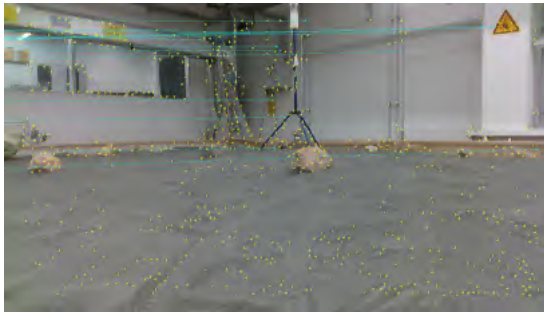
5.3.3 Autonomous Navigation

At it was described in Sec. 5.3.1, by using the same testbed setup for the first experiment, and a different one for the second, the autonomous navigation was validated.

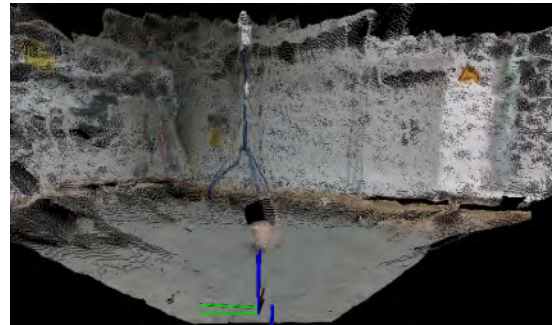
The difference between the Manual Mode and this controller is the absence of external intervention. The autonomous navigation only relies on the calculations that the algorithm provided internally from the NUC. The “Moon” delay does not affect the performance this time, since the data pipeline is performed locally. This setup is considered to have no more than 30 milliseconds in reaction delay. But this is only what the rover ‘sees’ because the user could notice the same 2.5 delay seconds.

However, if the rover made a mistake, the user could still intervene, but it would be much more difficult in a real mission since damage could occur even before aborting processes or pressing an emergency button. One

way to prevent this is by implementing a watchdog, which is an algorithm that, after a certain amount of time, somehow indicates to the system that something is going wrong with the process, and allows it to restart or reschedule the rover's trajectory, or even wait for further orders from the operator.



(a) Depth points and streaming



(b) Unique view

Figure 5.11: Instantaneous Pointcloud

Fig. 5.11a shows the depth points (similar to how it was done on the “Martian Surface” at Vulcano, while 5.11b literally represents how the robot “sees” in 3D space with an instantaneous the PointCloud.

Once the experiment was completed, the RTAB-Map visualizer was opened with another process in LN manager, and the 2D and 3D maps were displayed from the RTAB-Map database stored after the space mission.

The figure on the left is the direct streaming, while the one on the right was taken with the rtabmap-databaseViewer tool. A small zoom-out was made, and both the body (the center of mass of the robot) and the camera frames can be seen in the middle-bottom part.

In Fig. 5.12, in this generated 2D map, where it can be observed how the rover navigated using the planned path. The color labels are the following:

- Rover's initial position: coordinate system, **x on green**, and **y on red**
- Followed path: **blue-yellow**
- Explored area without obstacles: **light gray** (almost white)
- Explored area with obstacles: **black**
- Unexplored area: **dark gray**

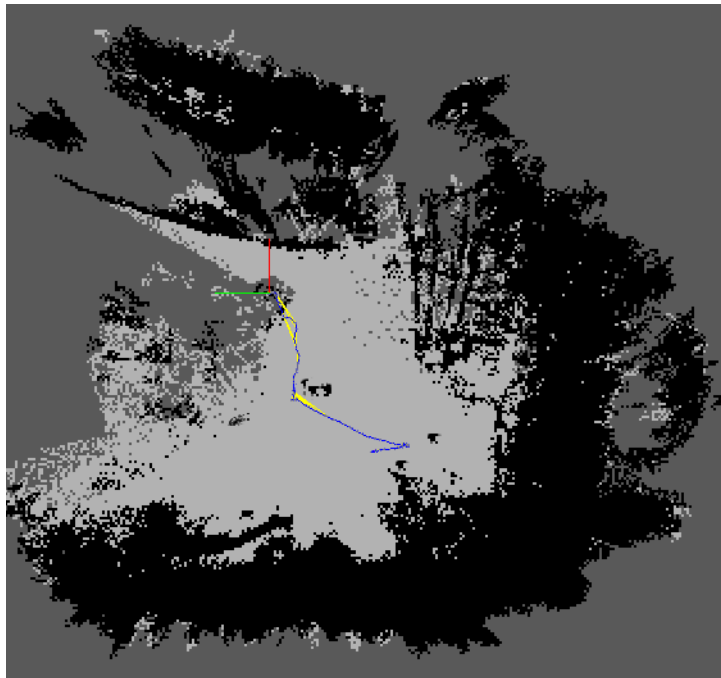


Figure 5.12: 2D Map and trajectory

The path planner, as explained in Sec. 3.4.3, uses an A* algorithm, where the robot's correction is generated. Such triangular approach depends on the present obstacles and the free path towards the final goal. The obstacle avoidance can be clearly noted by looking at the turning to the right (from the robot's perspective) and then adapting once more to the global planner.

Besides, a 3D map was generated and analyzed with the RTAB-Map visualizer. In Fig. 5.13 it is shown how the LRM clearly avoided the rock

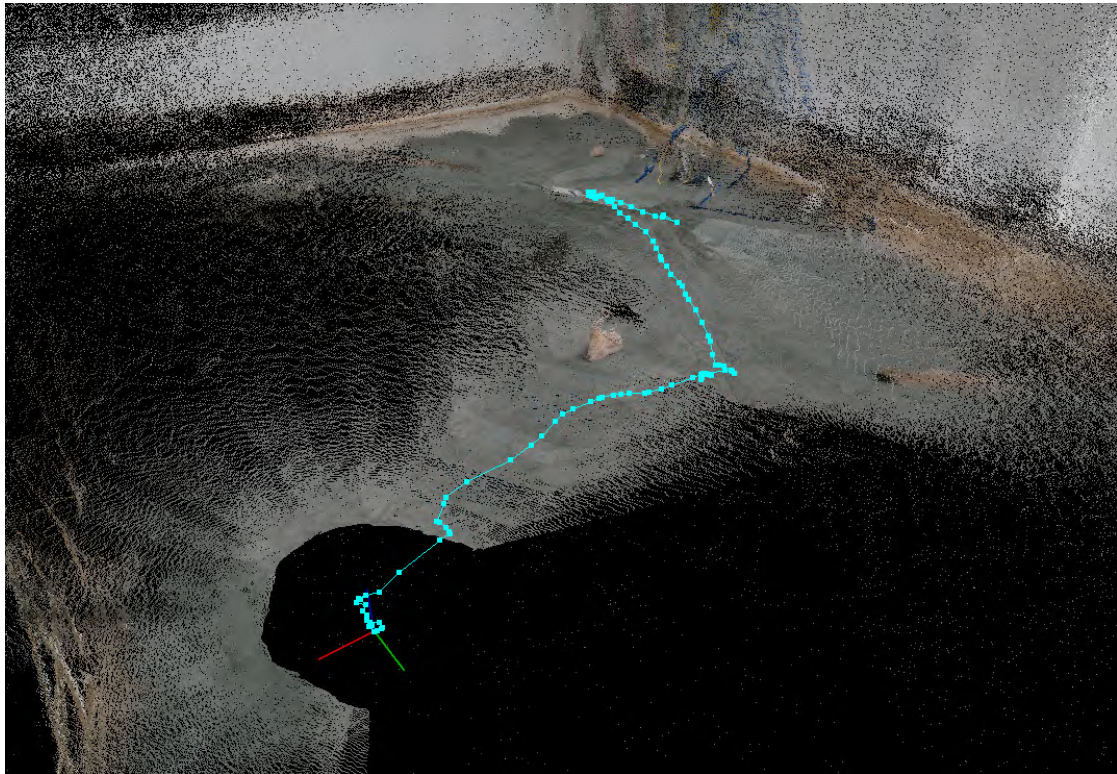


Figure 5.13: 3D Map, path planner with obstacle avoidance

that was on the straight path to its goal. The costmap of Fig. 5.14 at the second 45 of its trajectory defines whether there is an obstacle in front or not. In fact, it is categorizing the rock that is about to collide as a red area and leaves the free space as a pink area. After that assumption, the robot proceeded to recalculate the trajectory to the right, which was the most feasible path it found based on the explored map at that particular moment, a situation that not always is the best choice if the rover would know all the surroundings from the beginning, a decision which was one of the right ones in this case.

A small side note is that the drawn black circle at the beginning of its path represents the clockwise-Rotation mode to adjust the body steering towards the goal.

The setup and execution for each experiment took around an hour. More will be said about reaching a possible 100% future work in Sec. 6.1.

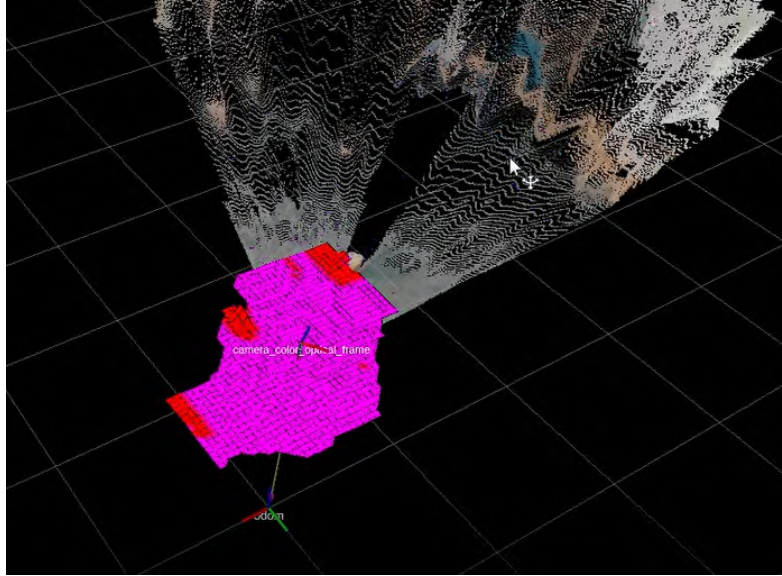


Figure 5.14: Costmap from the RMC Local Mapping

In the final video, two comments can be made about autonomous navigation:

Firstly, at minute 1:25, the rover began to rotate twice as if it did not locate itself on the map and did not know where to go. This would seem like an on-purpose exploration feature, but in reality, this phenomenon occurred due to the lack of good light conditions, the optimization of certain parameters, and the adjustment of the empirical driving modes conditioning.

There are many ways to correct the above, but two will be mentioned:

1. **Sensor fusion:** adding a LiDAR, improving the IMU integration (or another extra device), or even tracking the locomotion's odometry, could compensate for the limited field of view of the camera. Despite having good-quality image processing, it does not have as much resolution as other higher-priced cameras on the market, and it is quite sensitive to light conditions.
2. **Camera calibration:** the D435i camera already comes with an acceptable calibration for use by default; it can generate PointClouds

which can be transmitted within a short time period. However, some calibration tools, which were developed by Intel or by other members of the ROS community, have been explored, and many parameters and some experiments need to be reviewed in detail to take full advantage of their capabilities.

Secondly, some parameters such as the “step_height” and “slow_max” from the Navigation Stack package that represent the maximum delta-height that should not be recognized as an obstacle could be modified to adapt the algorithm to the dimensions of the LRM. It was noted that the free passage (half the distance between rock and rock) through which the rover could cross, could not be less than a meter. This happened because the tuning is very likely configured for the LRU, which has larger dimensions. Consequently, the rover “thinks” that it is bigger than it really is, so this perception must be modified.

In total, the first autonomous navigation experiment (which went better), took 2:51 minutes, or 171 seconds. This would mean that, compared to the Manual Mode, which lasted 93 seconds, the autonomy took 83.9% more time to reach its goal than a test by a direct user operation. Advantages? The LRM does all the traveling and decision-making by only consuming just under twice the time and energy as otherwise. Moreover, with more tests and better tuning, the LRM could perform obstacle avoidance and parking in a cleaner and more efficient way, although a little slower in average speed.

Regarding the accuracy to reach the final goal, the autonomous navigation experiments have shown an absolute error of less than 12 cm.

Of course, all these numbers can vary, since they depend on many factors.

Conclusions

As shown in Table 5.4, every feature of the rover that was planned to be implemented, has been completed. This means that a 100% functional prototype with autonomous navigation has been achieved for concept testing and future new development.

The locomotion's driving modes and conditioning **worked as developed**, especially using slowed-down velocities in some tests. It served as a benchmark for upcoming concept validations and improvements.

The concepts validated at the Vulcano Summer School and the Festival der Zukunft were very similar. One was done on more realistic terrains, and the other one was shown and explained to the general public, people who also interacted with the rover's interfaces. The objective of presenting the project as an integration of open-source modules for **learning and education purposes has been fulfilled**. Most people in the audience managed to understand the correlation between the software tools, the prototype, and the testbed.

All the connection and data exchange tests were successful since the key was the message standardization (GamePad controller, Simulink, rover communication, Navigation Stack, etc.). Both LN and ROS tools allowed the transfer and reception of data, thus being flexible in their operation and further module integration.

In matters related to quick tests, 15-20 minutes of setup are possibly classified as “normal”, but compared with the setup of a real demo mission

rover, the LRU, which was tested at Mount Etna, such setup could take even several hours. It can be established that the LRM is a very efficient platform to carry out basic to moderately-complex experiments to demonstrate planetary exploration principles.

The setting of changing driving modes explained in Sec. 3.4.2 was proven. The rover executed the driving modes continuously and without loops (a situation that had not been achieved in the first module integration attempt). In other words, the adaptation and integration of the program in Python `rover_communication` were adequate.

Using the percentages of the effectiveness of autonomous navigation explained in the analysis, its integration is considered within the range of “**beginning to be good**”. It can always be improved, but the integration and validation of all the modules have been a success.

The efficiency, the optimization, and other terms of deep analysis should be discussed since it was determined that the rover does not meet the necessary requirements to have a high Technology Readiness Level (TRL) value 4 as the LRU is with some -6 and 9 components.

6.1 Future Work

Great achievements have been made with this thesis, but much more remains to be done for future generations.

The autonomous navigation algorithm was adapted to the pipeline through mathematical conversions and standardization in system coordinates but was not deeply analyzed. A more detailed discussion of how efficient it is is beyond the scope of this thesis and could be looked up by other students.

6.1.1 Proposal for a Permanent Exhibition

After the presentation at the Festival der Zukunft, the possibility of proposing a setup for a permanent exhibition was discussed. For this, the following elements were explored:

Testbed

As it was done in the Deutsches Museum exhibition, a test bed similar to Fig. 5.6 would be needed. The dimensions may vary, but it is enough that per side they are approximately 4 m, being totally square or rectangular, but for assembly and weight purposes, a $3 \times 4 \text{ m}^2$ one is proposed. Obstacles can vary in size and distance from one another, but based on experimental results, it is suggested that there exist some that occupy no more than 15 cm^2 with a minimum distance of 70 cm among each.

AprilTag codes

The AprilTag codes are essentially codes that represent visual fiducial systems, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. The original idea is to use two or three of them in some corners of the test bed. This would make the rover have to cross the entire terrain avoiding obstacles and thus demonstrate all its traversability and driving modes. In addition, the plan is for the rover to be able to create a 3D PointCloud and display it somehow (tablet interface or computer), either in real-time or at the end of the mission.



Figure 6.1: April-tag code example [26]

Other tools

And of course, the GamePad or the controller being used at that moment (preferably a user-friendly tablet interface) with a single targeting option could be the essence of the exhibition. Certainly, other configurations can be used, such as looking for a battery or moving geological samples, but it would require a robotic arm, and it is believed that an integration point of that level is still a long way away.

6.1.2 New Pipeline Version

This subsection is best explained in a descriptive document sent to the LRM team by mail.dlr.de. It is about the restructuring of the data received by the Simulink model since this input is not a standardized vector data as it is the Twist vector. The team will need to go for any of the following two options:

1. Keep the current Simulink and GamePad program versions. This software transmits and receives the data message from Table 3.6, which is very specific. In order to integrate the Navigation with the full pipeline, the conditioning of autonomous navigation should be removed from the Rover Communication program and moved to the input of the Simulink model.
2. Integrate the conditioning and selection of management modes within the Simulink model, which would also lead to modifying the output of the GamePad code.

Currently, there are two functional but incomplete versions: one with the Manual Mode integrated with Simulink (no autonomous navigation), and the other one without Simulink but with all three driving modes and autonomous navigation conditioning implemented in the Rover Communication program. The goal in the future is to integrate both into a single standardized structure.

6.1.3 Migration to New Chassis

Other students are working on interface integration, like new controllers, or attaching a robotic arm as a payload. The new concept art representing the migration to the newly built design was developed during the Vulcano Summer School. The result was Fig. 6.2.

With this new model, it is expected that the mechanical stability will improve and the electronics will have a little more space for positioning, thus improving the ergonomic aspect. As a consequence, the camera stability might also increase.



Figure 6.2: New design migration concept [13]

Regarding the algorithms, do not forget to adapt the new dimensions in the calculations of the driving modes and the Python codes of the conditioning for autonomous navigation. Also, reviewing and modifying the equations of linear and rotational speeds would be required. The change should not be huge, but it is needed to achieve more accuracy.

tR

6.2 Work Contribution

Some final specifications about the documentation on GitHub are explained here, as well as a general analysis of the time investment on the project and the interactions with the public in Fig. 6.3.

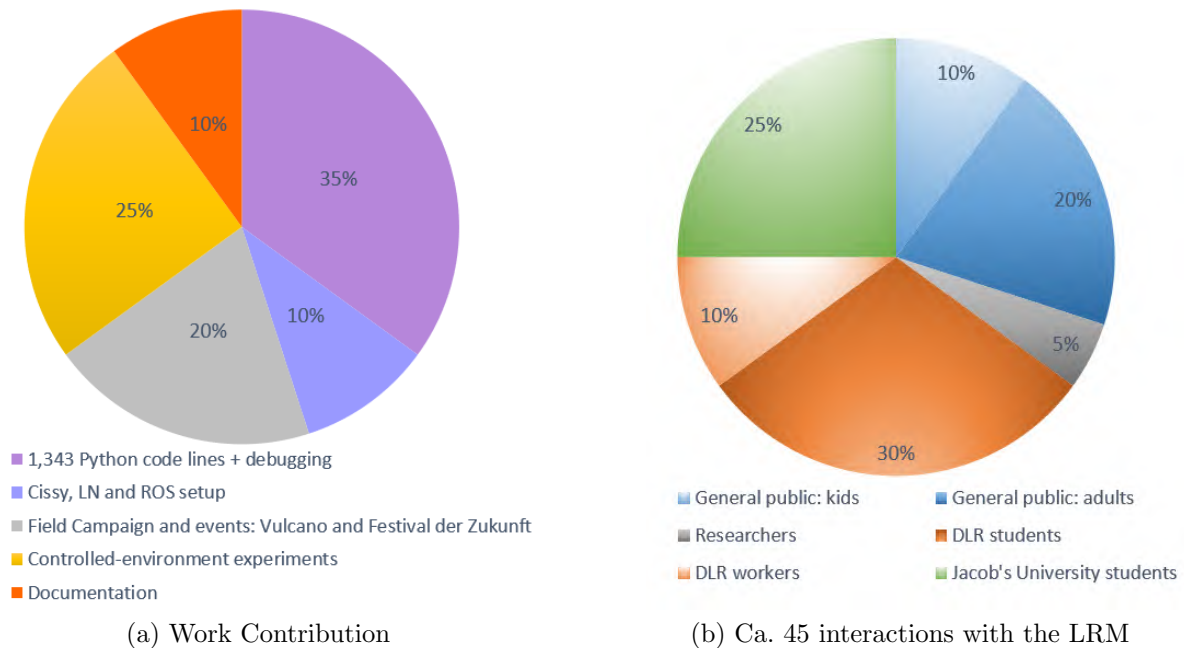


Figure 6.3: Work contribution statistics

6.2.1 GitHub Documentation

All relevant links are in the Appx. A. The documentation, together with the codes of the packages along with their respective README files are

stored on the RMC-GitHub repositories.

The manual [\[13\]](#) made by the LRM team is intended to complement this thesis in a more technical way and be used as a guide for future developments.

To make use of software tools such as LN Manager or Cissy, it is recommended to watch the video tutorials on the DLR Wiki and rely on the Mattermost platform for more efficient troubleshooting.

Most of the necessary processes here developed are also shown in the Appendices, in case it is necessary to go back to a functional version and restore the Simulink model or the manual and autonomous navigation modes.

Bibliography

- [1] **ExoMars Factsheet.** European Space Agency, 2022, recovered from: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Exploration/ExoMars/ExoMars_Factsheet, accessed on 14.06.2022.
- [2] **Mars Curiosity Rover.** NASA Science Mars Exploration Program, 2022, recovered from: <https://mars.nasa.gov/msl/home/>, accessed on 14.06.2022.
- [3] **Requirement Categories.** Argon Digital, 2022, recovered from: <https://argondigital.com/blog/product-management/>, accessed on 15.06.2022.
- [4] **Projekt Arches.** Helmholtz, 2022, recovered from: <https://www.arches-projekt.de/aktuelles/>, accessed on 15.06.2022.
- [5] **Festival der Zukunft.** 1E9 und Detusches Museum, 2022, recovered from: <https://www.festivalderzukunft.com/>, accessed on 15.06.2022.
- [6] Wedler, Armin, Irrgang Valentin; **Previous Manual of the LRM**, DLR (2021).
- [7] **Intel RealSense Depth Camera D435i.** Intel, 2022, recovered from: <https://www.intelrealsense.com/depth-camera-d435i/>, accessed on 16.06.2022.
- [8] **Intel NUC: Build It The Way You Want It.** Intel, 2022, recovered from: <https://www.intel.de/content/www/de/de/products/docs/boards-kits/nuc/nuc8i7beh-brief.html>, accessed on 16.06.2022.
- [9] **Gamepads.** Logitech, 2022, recovered from: <https://www.logitechg.com/de-de/products/gamepads.html>, accessed on 16.06.2022.
- [10] **Real-Time Appearance-Based Mapping.** Open robotics, 2021, recovered from: <http://introlab.github.io/rtabmap/>, accessed on 25.08.2022.
- [11] Min Wan Choi, J. S. Park, Bong Soo Lee and Man Hyung Lee, **The performance of independent wheels steering vehicle(4WS) applied Ackerman geometry.** 2008 International Conference on Control, Automation and Systems, 2008, pp. 197-202, doi: 10.1109/ICCAS.2008.4694549., recovered from: <https://ieeexplore.ieee.org/document/4694549>

-
- [12] **Ackerman Steering**. Robert Eisele, Computer Science & Machine Learning, 2022, recovered from: <https://www.xarg.org/book/kinematics/ackerman-steering/>, accessed on 30.08.2022.
- [13] Wedler, Armin, Ricardez Ortigosa, Adrian, Glückstadt Ludwig; **Official Manual of the LRM**, DLR (2022).
- [14] **XTRON LiPo Battery**. SLS, 2022, recovered from: https://www.stefansliposhop.de/en/batteries/sls-xtron:::173_120.html, accessed on 03.09.2022.
- [15] **FAULHABER SR-FLAT**. FAULHABER, 2022, recovered from: <https://www.faulhaber.com/de/produkte/serie/2619sr/>, accessed on 04.09.2022.
- [16] **TSR 2-2490**. TRACO POWER, 2022, recovered from: <https://www.tracopower.com/de/deu/model/tsr-2-2490>, accessed on 04.09.2022.
- [17] **ROS - Robot Operating System**. ROS.org, 2022, recovered from: <http://wiki.ros.org>, accessed on 06.09.2022.
- [18] **Two's Complement**. Wikipedia, 2022, recovered from: https://en.wikipedia.org/wiki/Two%27s_complement, accessed on 10.09.2022.
- [19] **tf**. ROS.org, 2022, recovered from: <http://wiki.ros.org/tf>, accessed on 23.12.2022.
- [20] **Communication Delays in Apollo Missions**. FlatEarth.ws, 2022, recovered from: <https://flatearth.ws/apollo-delay>, accessed on 03.01.2023.
- [21] **IMU Calibration and Drift**. Head tracker, 2022, recovered from: <https://headtracker.gitbook.io/head-tracker/getting-started/imu-calibration-and-drift>, accessed on 04.01.2023.
- [22] Chandra S., Rathin (2020). *Precise localization for achieving next-generation autonomous navigation: State-of-the-art, taxonomy and future prospects*, recovered from: <https://doi.org/10.1016/j.comcom.2020.06.007>.
- [23] Ibarra B., Natalia (2009). *Navegación autónoma de un robot con técnicas de localización y ruteo*. Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla. Recovered from: <https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/394/1/IbarraBMN.pdf>
- [24] Duoc Nguyen, H. (2023). *Efficient autonomous navigation for terrestrial insect-machine hybrid systems*. School of Mechanical & Aerospace Engineering, Singapore: Recovered from: <https://doi.org/10.1016/j.snb.2022.132988>

- [25] **Algorithms Used in Pathfinding and Navigation Meshes.** Corey Trevena, 2022, recovered from:
<https://www.cs.csustan.edu/~mmartin/teaching/CS4960S15/Corey%20Trevena%20-%20Pathfinding%20Algorithms%20in%20Navigational%20Meshes%20PDF.pdf>, accessed on 05.01.2023.
- [26] **Apriltag.** April Tag Robotics Laboratory, 2010, recovered from:
<https://april.eecs.umich.edu/software/apriltag#:~:text=AprilTag%20is%20a%20visual%20fiducial,tags%20relative%20to%20the%20camera>, accessed on 16.01.2023.

Relevant Links at DLR

Please note that one must be logged in on the DLR Intranet to get access to these websites, and that is why these links are not in the bibliography section.

Documentation:

- <https://wiki.robotic.dlr.de/LRM> - Official site of the LRM
- <https://www.overleaf.com/project/6242ec321176b869f45aabc5> - LRM 2022 Manual
- <https://wiki.robotic.dlr.de/Cissy> - Cissy

Documentation and GitHub packages

- https://rmc-github.robotic.dlr.de/moro/LRM_simulink - LRM Simulink
- https://rmc-github.robotic.dlr.de/moro/LRM_communication - LRM Communication
- https://rmc-github.robotic.dlr.de/moro/LRM_ln_msgdef - LRM ln message definition
- https://rmc-github.robotic.dlr.de/moro/LRM_simulink_ln_msgdef - LRM Simulink ln message definition
- https://rmc-github.robotic.dlr.de/moro/LRM_gamepad_controller - LRM GamePad controller
- <https://rmc-github.robotic.dlr.de/3rdparty/realsense-ros/tree/recipe/release/2.3.2> - realsense-ros for the intel d435i camera
- https://rmc-github.robotic.dlr.de/rica-ad/lrm_pan_tilt_tf_publisher - PanTilt Transform from LN to ROS
- <https://rmc-github.robotic.dlr.de/3rdparty/rtabmap> - RTAB-Map
- https://rmc-github.robotic.dlr.de/3rdparty/rtabmap_ros - RTAB-Map ROS
- https://rmc-github.robotic.dlr.de/moro/rmc_gbr_navigation - RMC GBR Navigation

- https://rmc-github.robotic.dlr.de/moro/rmc_local_mapping - RMC Local Mapping
- <https://rmc-github.robotic.dlr.de/common/rostopic2ln> - Rostopic2LN

Mission Instructions

B.1 Establishing The WiFi Connection

1. Turn on the rmc-lx0431 or any other available DLR laptop with 2,4/5GHz, and enter the “offline option”.
2. Enter the hard disk password, which is “lr...e”
3. Get access with your user and your cached password.
4. Connect the WiFi station configuration as explained in 3.3, and wait for a couple of minutes.
5. Connect the antenna module to the NUC and all other components as shown in Fig. 3.4. To see in detail where to connect the PanTilt servos, or other power/data buses, please check [\[13\]](#).
6. Grant access to the WiFi connection via the command:

```
# $ nmcli connection up rmc-ap0105-5-102-ap
```

which should appear in the available networks.
7. Enter the password requested, which starts with “su...l”.
8. Once connected, and once all LEDs shown in 3.5 are turned on, you can proceed to execute:

```
# $ ssh f_mobile@192.168.128.117
```

, which would be the fixed new IP address of the LRM.
9. As an alternative, if you want to run Rviz or other visual processes, it is always better to run the vnc network with:

```
# $ mrun vnc f_mobile@192.168.128.117
```

B.2 Running The Processes

Once inside, either via ssh or vnc, you can execute the command:

```
# $ mrun ws run
```

to start the LN Manager.

The processes from top to bottom to be launched for a demo mission, manual and autonomous, are:

1. `gamepad_control` : runs the node that maps the GamePad commands
2. `LRM_simulink` : launches the Simulink model
3. `rover_communication`: controls the connection and commands to the MainBoard
4. Tf group : `base_link_to_camera` and `lrm_pan_tilttf_publisher`: it does the transformations for the camera and body positions in the tf tree (for odometry)
5. `realsense_ros` (it will automatically run `roscore` and `rviz`)
6. Visual-Odometry group : `rtabmap_rgbodometry_params` and `rtabmap_rgbodometry`
7. Local-Mapping group : `rml_local_mapping_params` and `rml_local_mapping`
8. Navigation group: all nodes at once
9. SLAM group : `rtabmap_slam` and `rtabmap_slam_params` for creating a 3D-map of the explored area
10. `lrm1_rostopic2ln` : converts the ROS topics into LN topics

Optional:

1. `ros_shell` : to check `rostopics` or any other ROS features
2. `gamepad_control_wifi` : developed by another student, it is intended to work without the Bluetooth distance limitation and work with the same WiFi setup
3. `rtabmap_database` : opens the generated 2D and 3D maps IDE
4. `rtabmap_databaseVisualizer` : the same as above, but with more nice features in the RTAB-Map Visualizer

Remember to shut down the LRM every time it is disconnected, to avoid data corruption. To do this, it is necessary to know the **su root** password, which can be provided by the main project supervisor.

Simulink: Driving Modes

These Matlab scripts work only if the GamePad is the main controller. To integrate autonomous navigation, one of the following two options is required:

1. Use the code from Appx. F
2. Do what is explained in Sec. 6.1.2

C.1 Ackerman mode script

```
function [steering_angle, speed] = func(angle, body_vel)
quadrant = 0;           % from 1 to 4, 0 is undetermined, but
    initialized
% 1 | 4
%-----
% 2 | 3
k = 0.7;               % normal ackerman is abrupt, this is to smooth
    the data a bit
w = 0.0;               % initialization of body angular speed

% angle restriction for each quadrant to not more than 30
    respectively
if(angle >= 30 && angle <= 90)
    angle = 30;
elseif(angle >= 90 && angle <= 150)
    angle = 150;
elseif(angle >= 210 && angle <= 270)
    angle = 210;
elseif(angle >= 270 && angle <= 330)
    angle = 330;
end

% assignment of quadrants depending on provided angle
if(angle >= 0 && angle < 90)
```

```

    quadrant = 1;
elseif(angle >= 90 && angle < 180)
    quadrant = 2;
elseif(angle >= 180 && angle < 270)
    quadrant = 3;
elseif(angle >= 270 && angle < 360)
    quadrant = 4;
end

%% Output Values
steering_angle = zeros(6,1);
speed = zeros(6,1);

%% LRM Parameters
D = 0.085;           % distance to 1/2 Tread of vehicle
L = 0.102;           % distance to 1/2 Wheel base of vehicle

%% Ackermann Geometry
% For more information see: DOI: 10.1109/ICCAS.2008.4694549

if((angle > 2 && angle < 178) || (angle > 182 && angle < 358)
    ) % to prevent nan
    R = L/tand(angle); %truning radius vehicle
    w = body_vel/R;    %angular velocity vehicle

    r_fi = sqrt((R-D)^2 + L^2); %radius front inner
    r_fo = sqrt((R+D)^2 + L^2); %radius front outer
    r_ci = R-D;                %radius center inner
    r_co = R+D;                %radius center outer

    v_fi = r_fi*w;              %speed front inner
    v_fo = r_fo*w;              %speed front outer
    v_ci = r_ci*w;              %speed center inner
    v_co = r_co*w;              %speed center outer
    v_ri = v_fi;                %speed rear inner
    v_ro = v_fo;                %speed rear outer

    if(angle == 0)
        phi_fi = 0;
        phi_fo = 0;
        phi_ri = 0;
        phi_ro = 0;
    else

```

```
    phi_fi = atand(L/(R-D));           %steering angle front
        inner
    phi_fo = atand(L/(R+D));           %steering angle front
        outer
    phi_ri = phi_fi;                   %steering angle rear
        inner
    phi_ro = phi_fo;                   %steering angle rear
        outer
    % steering angle for center is zero
end

% wheel order in matlab: 1 = fr, 2 = cr, 3 = rr, 4 = rl,
    5 = cl, 6 = fl
% wheel order in matlab: 0 = rl, 1 = rr, 2 = fl, 3 = cl,
    4 = cr, 5 = fr

% wheel order in python:
% wheel order in python:

% depending on each quadrant, the angle logic is
different
if(quadrant == 1)
    steering_angle(1) = phi_fo;
    steering_angle(2) = 0;
    steering_angle(3) = 360-phi_ro;
    steering_angle(4) = 360-phi_ri;
    steering_angle(5) = 0;
    steering_angle(6) = phi_fi;
elseif(quadrant == 2)
    steering_angle(1) = phi_fi;
    steering_angle(2) = 0;
    steering_angle(3) = 360+phi_ri;
    steering_angle(4) = 360+phi_ro;
    steering_angle(5) = 0;
    steering_angle(6) = phi_fo;
elseif(quadrant == 3)
    steering_angle(1) = 360-phi_fi;
    steering_angle(2) = 0;
    steering_angle(3) = phi_ri;
    steering_angle(4) = phi_ro;
    steering_angle(5) = 0;
    steering_angle(6) = 360-phi_fo;
elseif(quadrant == 4)
```



```

steering_angle(1) = 360+phi_fo;
steering_angle(2) = 0;
steering_angle(3) = phi_ro;
steering_angle(4) = phi_ri;
steering_angle(5) = 0;
steering_angle(6) = 360+phi_fi;
end

% velocity multiplied by k to reduce it, because it is
% too abrupt
% Max vel is 15000, and it is almost always giving 15000
if(quadrant == 1)
    speed(6) = v_fi*k;
    speed(5) = v_ci*k;
    speed(4) = v_ri*k;
    speed(3) = v_ro*k;
    speed(2) = v_co*k;
    speed(1) = v_fo*k;
elseif(quadrant == 2)
    speed(6) = v_fo*k;
    speed(5) = -v_co*k;
    speed(4) = v_ro*k;
    speed(3) = v_ri*k;
    speed(2) = -v_ci*k;
    speed(1) = v_fi*k;
elseif(quadrant == 3)
    speed(1) = -v_fi*k;
    speed(2) = -v_ci*k;
    speed(3) = -v_ri*k;
    speed(4) = -v_ro*k;
    speed(5) = -v_co*k;
    speed(6) = -v_fo*k;
elseif(quadrant == 4)
    speed(1) = -v_fo*k;
    speed(2) = v_co*k;
    speed(3) = -v_ro*k;
    speed(4) = -v_ri*k;
    speed(5) = v_ci*k;
    speed(6) = -v_fi*k;
end
else
    for i=1:6
        steering_angle(i) = 0;    % if the angle is ~0, then

```

```
        don't do tangent, just do 0
    end

    % and the speed will only take the cosine (fwd and bckwrd
    % ) to make it more logic
    % and also, it is smoothed with the k factor
    speed(1) = body_vel*cosd(angle)*k;
    speed(2) = speed(1);
    speed(3) = speed(2);
    speed(4) = speed(3);
    speed(5) = speed(4);
    speed(6) = speed(5);
end

for i=1:6
```

C.2 Rotation mode script

```
function [steering_angle, speed] = func(body_vel)

% Rotation: Rover turns on the spot
% Positive body_vel (speed) = turn right
% Negative body_vel (speed) = turn left

%% LRM Parameters
D = 0.085;           %distance to 1/2 Tread of vehicle
L = 0.102;           %distance to 1/2 Wheel base of vehicle

r_f = sqrt(D^2+L^2); %radius front wheels to rover center
r_c = D;             %radius center wheels to rover center

w = body_vel/r_f;    %angular velocity
v_fl = body_vel;     %speed front left
v_cl = body_vel;     %speed center left
v_rl = body_vel;     %speed rear left

v_fr = -body_vel;    %speed front right
v_cr = -body_vel;    %speed center right
v_rr = -body_vel;    %speed rear right

% angles already set to rotate correctly
phi_fl = 360-60;     %steering angle front left
```

```
phi_cl = 0;           %steering angle center left
phi_rl = 60;          %steering angle rear left
phi_rr = 360-60;      %steering angle rear right
phi_cr = 0;           %steering angle center right
phi_fr = 60;          %steering angle front right
%steering angle for center is zero

%% Output Values
steering_angle = zeros(6,1);
speed = zeros(6,1);

steering_angle(1) = phi_fr;
steering_angle(2) = 0;
steering_angle(3) = phi_rr;
steering_angle(4) = phi_rl;
steering_angle(5) = 0;
steering_angle(6) = phi_fl;

speed(1) = v_fr;
speed(2) = v_cr;
speed(3) = v_rr;
speed(4) = v_rl;
speed(5) = v_cl;
speed(6) = v_fl;
```

C.3 Crabwalk mode script

```
function [steering_angle, speed] = func(angle, body_vel)

%% Output Valuesfunction [steering_angle, speed] = func(angle
    , body_vel)

%% Output Values
steering_angle = zeros(6,1);
speed = zeros(6,1);

if (angle < 0)
    angle = 360-angle;
end

steering_angle(1) = angle;
steering_angle(2) = angle;
```

```
steering_angle(3) = angle;
steering_angle(4) = angle;
steering_angle(5) = angle;
steering_angle(6) = angle;

speed(1) = body_vel;
speed(2) = body_vel;
speed(3) = body_vel;
speed(4) = body_vel;
speed(5) = body_vel;
speed(6) = body_vel;

% in crabwalk, all wheels are the same in speed and angle
for i=1:6
    steering_angle(i) = angle;
    speed(i) = body_vel;
end
steering_angle = zeros(6,1);
speed = zeros(6,1);

if (angle < 0)
    angle = 360-angle;
end

steering_angle(1) = angle;
steering_angle(2) = angle;
steering_angle(3) = angle;
steering_angle(4) = angle;
steering_angle(5) = angle;
steering_angle(6) = angle;

speed(1) = body_vel;
speed(2) = body_vel;
speed(3) = body_vel;
speed(4) = body_vel;
speed(5) = body_vel;
speed(6) = body_vel;

% in crabwalk, all wheels are the same in speed and angle
for i=1:6
    steering_angle(i) = angle;
    speed(i) = body_vel;
end
```

gamepad_controller code

This is the Gamepad_LRM.py code. Please, always uncomment the joystick filter (4 lines of code) which needs to be moved to the same indentation level as the “JOYSTICK FILTER” note.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 27 15:14:13 2022
@author: irrg_va and rica_ad
Description:
    Script to connect (auto reconnect) to "Logitech Wireless Gamepad F710", read input events
    and (process/) publish data to the ln-manager
    @note Logging levels can be changed by pressing LB, default is: WARNING
    @note DEBUG: show output of gamepad inputs
    @note INFO: show all connection details and mode changes
    @note WARNING: show critical connection details
    @note ERROR: not used (output nothing)
"""

import sys
import os
import struct
import select
import re # regular expression
import math
import array
import time
import timeit
from collections import namedtuple
# import numpy as np
import links_and_nodes as ln
# import matplotlib.pyplot as plt
import logging

# create subclass: "input_event_tuple" to be able to sort raw gamepad data
input_event_tuple = namedtuple("input_event", ["tv_sec", "tv_usec", "type", "code", "value"])

class input_event(input_event_tuple):
    """
    Returns raw gamepad data as "input_events", accessible via .type, .code, and .value
    ## Example
    -----{.py}
    a = input_event(data)
    a.value
    -----
    """
    def __str__(self):
        return "input_event(type=%s, code=%s, value=%d)" % (
            event_types_back.get(self.type, self.type),
            self.code,
            self.value)

class input_reader:
    """
    Class to handle connection and event reading from gamepad
    """
    def __init__(self):
        self.clnt = ln.client(sys.argv[0], sys.argv[1:]) # links and nodes client
        # self.port = self.clnt.publish("GamePad", "md_GamePad") # set up publisher
        self.input_event_format = "LLHHi"
        self.input_event_format_size = struct.calcsize(self.input_event_format)
        self.fd = "" # file descriptor for gamepad-path
        self.path = "" # selected gamepad-path
        self.gamepad_dir = "/dev/input/by-id/" # path to search for gamepads
        # old gamepad id = 16AA985F, new gamepad id = CA4D91F4
        self.connect_gamepad()
        self.ev_data = 0 # event values - data
```

```

def connect_gamepad(self):
    """
    Search for available logitech F710 gamepads and try to connect/reconnect to them
    """
    while (self.fd == ""):
        try:
            list_of_files = os.listdir(self.gamepad_dir) # list of files in the current directory
        except:
            logging.info("no gamepads connected")
            time.sleep(3)
            continue # try again
        gamepads = []
        for usb_dev in list_of_files:
            if usb_dev.startswith('usb-Logitech') and usb_dev.endswith(
                "event-joystick"): gamepads.append(usb_dev)
        if len(gamepads) == 0:
            logging.info("no gamepad found")
        elif len(gamepads) == 1:
            try:
                self.fd = os.open(self.gamepad_dir + gamepads[0], os.O_RDONLY)
                self.path = self.gamepad_dir + gamepads[0]
                logging.warning("gamepad: " + str(self.gamepad_dir + gamepads[0]) + " , connected")
            except:
                logging.info("could not connect to gamepad") # gamepad found but cant connect
                time.sleep(3)

def check_events(self, timeout=0.01):
    """!
    Read raw data from gamepad (if it is connected)
    """
    while True:
        try:
            rfd, _, _ = select.select([self.fd], [], [], timeout)
            if not rfd:
                return 0
            ev = os.read(self.fd, self.input_event_format_size) # read controller inputs
            self.ev_data = input_event(*struct.unpack(self.input_event_format, ev))
            had_events = True
            # logging.debug(("code: "+str(ev.code) + ", value: " + str(ev.value)))
            return ev
        except:
            pass
            logging.warning("Connection lost, trying to reconnect...")
            self.fd = ""
            self.connect_gamepad() # try to reconnect to gamepad

class gamepad_f710:
    """!
    Class to convert gamepad event values to commands that work with the lrm controller interface
    and publish data to links and nodes
    """
    def __init__(self):
        '''translate input types to input codes'''
        self.input_names = [
            "joy_left_lr",
            "joy_left_ud",
            "joy_right_lr",
            "joy_right_ud",
            #####
            "button_Y",
            "button_B",
            "button_A",
            "button_X",
            #####
            "joypad_lr",
            "joypad_ud",
            #####
            "button_lt",
            "button_rt",
            "button_lb",
            "button_rb",
        ]
        self.input_codes = [
            0,
            1,
            3,
            4,
            #####
            308,
            305,
            304,
            307,
            #####
            16,
            17,
            #####
            2,

```

```

5,
310,
311
]

'''ud = up/down, lr = left/right'''
self.joy_left_lr = 0 # left and right
self.joy_left_ud = 0 # up and down
self.joy_left_change = 0 # 1 if values have changed
self.joy_right_lr = 0 # left and right
self.joy_right_ud = 0 # up and down
self.joy_right_change = 0 # 1 if values have changed
self.joypad_ud_last = "none" # what arrow was pressed last
self.joypad_lr_last = "none" # what arrow was pressed last
self.speed_range = 100.0 # speeds from 0 - speed_range
self.emergency_stop = 0
self.frequency = 100 # publish frequency in Hz
'''values for ln'''
self.vel_x = 0
self.vel_y = 0
self.steering_angle = 0
self.drive_mode = 1 # default = 1 = ackermann steering
self.rover_mode = 90 # error reset mode
self.controller_on_off = 1 # en-/disable Bogie controller 1/0
self.pan = 139
self.tilt = 164
self.assume = 0
self.permission = 0
'''pan-/tilt'''
self.update_pan = 0
self.update_tilt = 0
self.last_update = 0
self.update_interval = 0.04 # seconds
'''calibration mode'''
self.prev_mode = 0 # previous selected mode
self.calib_enable = 0 # if calibration mode has been enabled
self.calib_interval = 0.1 # seconds (delay in between controller on/off and calibration)
self.calib_delay = 0 # to keep track of time
'''Links and Nodes connection'''
self.controller_num = 1 # controller channel number in simulink
clnt = ln.client(sys.argv[0], sys.argv[1:]) # create ln-client
self.portOUT = clnt.publish("Controller_1", "md_controller") # publisher client
#self.portOUT = clnt.publish("lrm1.gamepad_controller", "md_controller")
"""Logging"""
self.log_lvls = ["ERROR", "WARNING", "INFO", "DEBUG"]
self.log_lv = 1 # warning is default level
self.start_time = timeit.default_timer()

def event_mapping(self, ev):
    """
    Function to convert event values from gamepad to LRM commands
    joy_left_ud -> -
    joy_left_lr -> -
    joy_right_ud -> steering angle
    joy_right_lr -> steering angle
    joypad_ud -> tilt
    joypad_lr -> pan
    button_Y -> calibration mode
    button_B -> error reset mode
    button_A -> set permission for other controllers to get control
    button_X -> emergency button
    button_lt -> drive backwards
    button_rt -> drive forwards
    button_lb -> iterate logging levels
    button_rb -> change drive mode
    """
    if ev.code in self.input_codes:
        current_event = self.input_names[self.input_codes.index(ev.code)]
        # not used
        if current_event == "joy_left_ud":
            self.joy_left_ud = (ev.value * -1 / (2 ^ 15) * 0.1) # correctly scale joystick value
            self.joy_left_change = 1
            if self.joy_left_ud > 1: # filter noise
                logging.debug(("joy left ud: " + str(
                    round(self.constrain_angle(self.get_angle(self.joy_left_lr, self.joy_left_ud)), 0))))
        # not used
        elif current_event == "joy_left_lr":
            self.joy_left_lr = (ev.value * -1 / (2 ^ 15) * 0.1) # correctly scale joystick value
            self.joy_left_change = 1
            if self.joy_left_lr > 1: # filter noise
                logging.debug(("joy left lr: " + str(
                    round(self.constrain_angle(self.get_angle(self.joy_left_lr, self.joy_left_ud)), 0))))
        # steering angle y
        elif current_event == "joy_right_ud":
            self.joy_right_ud = (ev.value * -1 / (2 ^ 15) * 0.1) # correctly scale joystick value
            self.joy_right_change = 1
            if self.joy_right_ud > 1: # filter noise
                logging.debug(("joy right ud: " + str(
                    round(self.constrain_angle(self.get_angle(self.joy_right_lr, self.joy_right_ud)), 0))))
        # steering angle x

```

```

elif current_event == "joy_right_lr":
    self.joy_right_lr = (ev.value * -1 / (2 ^ 15) * 0.1) # correctly scale joystick value
    self.joy_right_change = 1
    if self.joy_right_lr > 1: # filter noise
        logging.debug(("joy right lr: " + str(
            round(self.constrain_angle(self.get_angle(self.joy_right_lr, self.joy_right_ud), 0))))
# tilt
elif current_event == "joypad_ud":
    if ev.value == -1:
        logging.debug("arrow up")
        self.joypad_ud_last = "up"
        self.update_tilt = 1
    if ev.value == 1:
        logging.debug("arrow down")
        self.joypad_ud_last = "down"
        self.update_tilt = 1
    if ev.value == 0:
        self.update_tilt = 0
        logging.debug("tilt value: " + str(self.tilt))
        logging.debug("arrow " + self.joypad_ud_last + " released")
# pan
elif current_event == "joypad_lr":
    if ev.value == -1:
        logging.debug("arrow left")
        self.joypad_ud_last = "left"
        self.update_pan = 1
    if ev.value == 1:
        logging.debug("arrow right")
        self.joypad_ud_last = "right"
        self.update_pan = 1
    if ev.value == 0:
        self.update_pan = 0
        logging.debug("pan value: " + str(self.pan))
        logging.debug("arrow " + self.joypad_ud_last + " released")
# calibration for rover wheels
elif current_event == "button_Y":
    if ev.value == 1:
        self.calib_enable = 1
        self.calib_delay = timeit.default_timer()
        self.prev_mode = self.rover_mode
        self.rover_mode = 120
        self.controller_on_off = 0
    elif ev.value == 0:
        self.calib_enable = 3
        self.calib_delay = timeit.default_timer()
        self.rover_mode = 120
        self.controller_on_off = 1
    logging.debug((current_event + str(ev.value)))
# error reset
elif current_event == "button_B":
    if ev.value == 1:
        self.rover_mode = 90 # error reset mode
        logging.info("Error reset (set mode: " + str(self.rover_mode) + ") enabled")
    elif ev.value == 0:
        self.rover_mode = 200 # normal (drive) mode
        logging.info("Error reset (set mode: " + str(self.rover_mode) + ") disabled")
    logging.debug((current_event + str(ev.value)))
# controller permission
elif current_event == "button_A":
    if ev.value == 1:
        if self.permission == 0:
            self.permission = 1
            logging.info("enabled - permission for other controllers to gain control")
        else:
            self.permission = 0
            logging.info("disabled - permission for other controllers to gain control")
    logging.debug((current_event + str(ev.value)))
# emergency button - no more data will be sent
elif current_event == "button_X":
    if ev.value == 1:
        if self.emergency_stop == 0:
            self.portOUT.packet.vel_x = 0
            self.portOUT.packet.steering_angle = 0
            # self.portOUT.packet.drive_mode = 1 # no need to change drive mode
            self.portOUT.packet.rover_mode = 90
            self.portOUT.packet.permission = 1
            self.portOUT.write() # publish
            self.emergency_stop = 1
            logging.warning("Emergency stop!!!")
        elif self.emergency_stop == 1:
            self.emergency_stop = 0
            logging.warning("Continue operation")
    logging.debug((current_event + str(ev.value)))
# rover speed (backward)
elif current_event == "button_lt":
    if ev.value > 0:
        self.vel_x = -((ev.value - 0.0) * float(self.speed_range)) / 255 / 100 + 0
    if self.vel_x > -0.4:
        self.vel_x = 0

```



```

        logging.debug("x_vel: " + str(self.vel_x))
        # logging.debug((current_event + str(ev.value)))
        # rover speed (forward)
        elif current_event == "button_rt":
            if ev.value > 0:
                self.vel_x = (((ev.value - 0.0) * float(self.speed_range)) / 255)/100 + 0
                if self.vel_x < 0.4:
                    self.vel_x = 0
                logging.debug("x_vel: " + str(self.vel_x))
                # logging.debug((current_event + str(ev.value)))
            # change drive modes 1=ackermann, 2 = PointTurn, 3 = CrabWalk
            elif current_event == "button_rb":
                if ev.value == 1:
                    self.drive_mode = (self.drive_mode + 1)
                    if self.drive_mode > 3:
                        self.drive_mode = 1
                    logging.info("drive_mode: " + str(self.drive_mode))
                    logging.debug((current_event + str(ev.value)))
            # change logging levels
            elif current_event == "button_lb":
                if ev.value == 1: # only on button press
                    self.log_lv = self.log_lv + 1
                    if len(self.log_lvs)-1 >= self.log_lv:
                        if self.log_lvs[self.log_lv] == "ERROR":
                            logging.getLogger().setLevel(logging.ERROR)
                            logging.error("logging level set to " + self.log_lvs[self.log_lv])
                        elif self.log_lvs[self.log_lv] == "WARNING":
                            logging.getLogger().setLevel(logging.WARNING)
                            logging.warning("logging level set to " + self.log_lvs[self.log_lv])
                        elif self.log_lvs[self.log_lv] == "INFO":
                            logging.getLogger().setLevel(logging.INFO)
                            logging.info("logging level set to " + self.log_lvs[self.log_lv])
                        elif self.log_lvs[self.log_lv] == "DEBUG":
                            logging.getLogger().setLevel(logging.DEBUG)
                            logging.debug("logging level set to " + self.log_lvs[self.log_lv])
                    else:
                        self.log_lv = 0
                        logging.getLogger().setLevel(logging.ERROR)
                        logging.error("logging level set to " + self.log_lvs[self.log_lv])

                logging.debug((current_event + str(ev.value)))
            elif current_event == "button_rb":
                logging.debug((current_event + str(ev.value)))

        '''convert to steering angle'''
        if self.joy_left_change == 1:
            self.joy_left_change = 0
            #if 1.0 - (timeit.default_timer() - self.start_time) < 0:
            #    logging.info("angle: " + str(self.steering_angle))
            #    self.start_time = timeit.default_timer()

        elif self.joy_right_change == 1: # compute steering angle from joystick values
            self.joy_right_change = 0
            #print("joy_right_lr: %d joy_right_ud: %d" %(self.joy_right_lr,self.joy_right_ud))

            # ----- JOYTICK FILTER by Adrian R. 2022 -----
            # this will restrict the angle of the GamePad to make it softer when reached the edges
            # UNCOMMENT THIS AT THIS LEVEL!

        #if abs(self.joy_right_lr) >= 230 or abs(self.joy_right_ud) >= 150:
        #    self.steering_angle = round(self.constrain_angle(self.get_angle(self.joy_right_lr, self.joy_right_ud)), 0)
        #elif (abs(self.joy_right_lr) + abs(self.joy_right_ud)) >= 300:
        #    self.steering_angle = round(self.constrain_angle(self.get_angle(self.joy_right_lr, self.joy_right_ud)), 0)

    def get_angle(self, y, x):
        """
        Convert a x and y pos of joystick into an angle (0 ->180 , 0 ->-180 )
        """
        return math.degrees(math.atan2(y, x)) # polar coordinates

    def constrain_angle(self, angle):
        """
        constrain steering angle
        """
        if angle > 180: angle = 360 - angle
        if angle < 0: angle = 360 + angle
        return angle

    def calibration_update(self):
        """
        This function handles the calibration process of the rover
        There are 3 stages to this:
        1. Turn Bogie controllers off (controller off = 120)
        2. Wait for wheels to be adjusted (calibration = mode 150)
        3. Turn Bogie controllers back on (controller on = 120)
        """
        if (self.calib_enable == 1) and ((timeit.default_timer()-self.calib_delay) > self.calib_interval):
            self.rover_mode = 150
            self.calib_enable = 2
            logging.info("Calibration (set mode: " + str(self.rover_mode) + ") enabled")

```

```

elif self.calib_enable == 2:
    pass # do nothing until Y-button is released
elif (self.calib_enable == 3) and ((timeit.default_timer()-self.calib_delay) > self.calib_interval):
    self.rover_mode = self.prev_mode
    self.calib_enable = 0
    logging.info("Calibration (set mode: " + str(self.rover_mode) + ") disabled")

def pan_tilt_update(self):
    """
    set values for pan and tilt (0-255 PWM)
    """
    update = 0 # 1 if pan tilt values change
    if self.update_tilt:
        if self.joyypad_ud_last == "up":
            if (timeit.default_timer() - self.last_update) > self.update_interval:
                self.tilt = self.tilt + 1
                self.last_update = timeit.default_timer()
                if self.tilt > 255:
                    self.tilt = 255
        elif self.joyypad_ud_last == "down":
            if (timeit.default_timer() - self.last_update) > self.update_interval:
                self.tilt = self.tilt - 1
                self.last_update = timeit.default_timer()
                if self.tilt < 0:
                    self.tilt = 0
    if self.update_pan:
        if self.joyypad_ud_last == "left":
            if (timeit.default_timer() - self.last_update) > self.update_interval:
                self.pan = self.pan + 1
                self.last_update = timeit.default_timer()
                if self.pan > 255:
                    self.pan = 255
        elif self.joyypad_ud_last == "right":
            if (timeit.default_timer() - self.last_update) > self.update_interval:
                self.pan = self.pan - 1
                self.last_update = timeit.default_timer()
                if self.pan < 0:
                    self.pan = 0
    return update

def send_data(self):
    """
    Write controller data to ln-manager port and publish topic
    """
    #self.vel_x = self.vel_x + 0.01
    #self.steering_angle = self.steering_angle + 1

    #if self.vel_x > 1.0:
    #    self.vel_x = 0
    #if self.steering_angle > 360:
    #    self.steering_angle = 0

    self.portOUT.packet.vel_x = self.vel_x
    self.portOUT.packet.vel_y = self.vel_y
    self.portOUT.packet.rotation = self.steering_angle
    self.portOUT.packet.drive_mode = self.drive_mode
    self.portOUT.packet.mode = self.rover_mode
    self.portOUT.packet.controller_on_off = self.controller_on_off
    self.portOUT.packet.Pan = self.pan
    self.portOUT.packet.Tilt = self.tilt
    self.portOUT.packet.Assume = self.assume
    self.portOUT.packet.Permission = self.permission
    self.portOUT.write() # publish

    # print variables hehe
    print("vel_x: %f vel_y: %f angle: %f drive_mode: %d rover_mode: %d pan: %d tilt: %d permission: %d"
          %(self.vel_x, self.vel_y, self.steering_angle, self.drive_mode, self.rover_mode,
            self.pan, self.tilt, self.permission))

def main(args):
    """
    main gamepad loop
    """
    clnt = ln.client("GamePad", args) # ln client
    ir = input_reader()
    gp = gamepad_f710()
    send_period = 0

    while True:
        if ir.check_events(timeout=0.01):
            gp.event_mapping(ir.ev_data)
        if gp.emergency_stop == 0:
            if send_period < timeit.default_timer():
                gp.send_data()
                send_period = timeit.default_timer() + (1.0 / gp.frequency)
            gp.pan_tilt_update()
            gp.calibration_update()

```

rover_communication with Simulink

This Rover.py code works only if the GamePad is the main controller. To integrate the autonomous navigation, one of the following two options is required:

1. Use the code from Appx. F
2. Do what is explained in Sec. 6.1.2

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Started on March 21 2022
@author: irrg_va
Finished on Jan 31 2023
@author: rica_ad
"""

# LIBRARIES
import logging
import threading
import time
import array
import sys
import serial
import serial.tools.list_ports
import glob
import struct
import numpy as np
import math
import links_and_nodes as ln
import time

# ----- Global Variables -----
wheelDiam_m = 0.0534          # wheel diameter in meters old rover (black)
robot_rotationDiam_m = 0.27  # it is the distance from the center to the corner wheels
                               # used to calculate Rotation mode velocity
body_w = 0.0                 # initializer of the body angular velocity

bodyV_max_m_s = 0.115        # from experiments 09.11.2022 in lab, meters per sec
wheelV_max_value = 15000     # found experimentally, maybe til 16000, but better set to this one

calibrated_pantilt = [139,164] # experimentally set to get a good mapping 09.11.2022
communication_time = 1./1000. # to give time in secs to the controller to command the velocity

mapping_bodyV_max_m_s = 0.05 # TODO: to cross-check the value
mapping_bodyW_max_rad_s = 0.25 # TODO: to cross-check the value
mapping_bodyW_min_rad_s = 0.025 # TODO: to cross-check the value

# *****
#                               Data Classes (@irrg_va)
# *****
class LRM_DataStruct:
    def __init__(self):
        self._lock = threading.Lock()

        self.errorextern = 0 # errorvalue
        self.seterror = 0    # errorclearvalue for bogies

        self.bogie00 = self.Bogie(0,0) # create all the bogies
        self.bogie01 = self.Bogie(0,1)
        self.bogie10 = self.Bogie(1,0)
        self.bogie11 = self.Bogie(1,1)
        self.bogie20 = self.Bogie(2,0)
        self.bogie21 = self.Bogie(2,1)
```

```

self.bogies =[]                                # put the bogies in a list
self.bogies.append(self.bogie00)
self.bogies.append(self.bogie01)
self.bogies.append(self.bogie10)
self.bogies.append(self.bogie11)
self.bogies.append(self.bogie20)
self.bogies.append(self.bogie21)

self.pantilt = self.PanTilt()                  # set the pantilt

class Bogie:
    def __init__(self, Bogie_ID, Wheel_ID):
        #set bogie data
        # default mode = normalmode
        self.mode = 200
        self.bogie_ID = Bogie_ID
        self.wheel_ID = Wheel_ID
        self.setangle = 0
        self.setspeed = 0
        #get bogie data
        self.realangle = 0
        self.realspeed = 0
        self.anglebogie = 0                    #only in bogiewheel X0 (X=0/1/2) -> see physical connection
        self.motorcurrentsteer = 0
        self.motorcurrentdrive = 0
        self.magencodesteerinc = 0             # magnetic encoder steer increments
        self.flagcounterdrive = 0
        self.errorintern = 0                   # gets send as ERRtmp
        self.crc_check = 0
        #status bogie
        self.sendData = True
        self.setControllerOnOff = 1
        #PID controller data
        self.PID_data = self.PID_Data()

class PID_Data:
    def __init__(self):
        # TODO: Note that these default values are also hard coded in the
        # bogie code and where determined purely experimentally (irrg_va)
        self.DP_i = 36 #Drive Parameter P inner loop
        # Drive PIDinit: 36, 4, 0,11,0,0, 65000 ,65000, 4, 2
        self.DI_i = 4 #Drive Parameter I inner loop
        self.DD_i = 0 #Drive Parameter D inner loop
        self.DP_o = 11 #Drive Parameter P outer loop
        self.DI_o = 0 #Drive Parameter I outer loop
        self.DD_o = 0 #Drive Parameter D outer loop
        self.SP_i = 4113 #Steer Parameter P inner loop
        # SteerPIDinit: 4113,0,0 ,8912,0,0, 65000 , 65000, 4, 1
        self.SI_i = 0 #Steer Parameter I inner loop
        self.SD_i = 0 #Steer Parameter D inner loop
        self.SP_o = 8912 #Steer Parameter P outer loop
        self.SI_o = 0 #Steer Parameter I outer loop
        self.SD_o = 0 #Steer Parameter D outer loop
        self.ScaleD_i = 4 #Drive Scaling inner loop
        self.ScaleS_i = 4 #Steer Scaling inner loop
        self.ScaleD_o = 2 #Drive Scaling outer loop
        self.ScaleS_o = 1 #Steer Scaling outer loop
        self.WD_o = 65000 #Drive Windup outer loop
        self.WD_i = 65000 #Drive Windup inner loop
        self.WS_o = 65000 #Steer Windup outer loop
        self.WS_i = 65000 #Steer Windup inner loop
        # list PID parameter for systematic access: S= Steer, D = Drive, I/i = Inner Loop,
        # O/o = Outer Loop #TODO:simplify parameter names
        self.list_S_I = [self.SP_i, self.SI_i, self.SD_i, self.ScaleS_i, self.WS_i]
        self.list_S_O = [self.SP_o, self.SI_o, self.SD_o, self.ScaleS_o, self.WS_o]
        self.list_D_I = [self.DP_i, self.DI_i, self.DD_i, self.ScaleD_i, self.WD_i]
        self.list_D_O = [self.DP_o, self.DI_o, self.DD_o, self.ScaleD_o, self.WD_o]
        self.list_S_IO = [self.list_S_I, self.list_S_O]
        self.list_D_IO = [self.list_D_I, self.list_D_O]
        self.list_SteerDrive = [self.list_S_IO, self.list_D_IO]

class PanTilt():
    def __init__(self):
        self.PanTiltset = 1
        self.Pan = calibrated_pantilt[0]
        self.Tilt = calibrated_pantilt[1]

# *****
# Communication Class and Functions (@irrg_va)
# =====
class Communication:
    def __init__(self):
        self.ser = 0
        self.selected_port = 0

    def connect(self):
        #get all the available ports and print them
        if sys.platform.startswith('win'):
            ports = ['COM%s' % (i + 1) for i in range(256)]

```

```
elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
    # this excludes your current terminal "/dev/tty"
    ports = glob.glob('/dev/tty[A-Za-z]*')
elif sys.platform.startswith('darwin'):
    ports = glob.glob('/dev/tty.*')
else:
    raise EnvironmentError('Unsupported platform')
result = []
for port in ports:
    try:
        s = serial.Serial(port)
        s.close()
        result.append(port)
    except (OSError, serial.SerialException):
        pass
for lauf in result :
    print(lauf)

self.selected_port = 0 # input("Select Port: "), can be user input, but better set to 0
try:
    self.ser = serial.Serial(result[int(self.selected_port)])
    if self.ser.isOpen():
        print(self.ser.portstr + " is enabled!\n")

        self.ser.baudrate = 230400
        self.ser.timeout = 2
        self.ser.bytesize = serial.EIGHTBITS
        self.ser.parity = serial.PARITY_NONE
        self.ser.stopbits = serial.STOPBITS_ONE
    except:
        print("could not connect to"+result[self.selected_port])

def disconnect(self):
    try:
        self.ser.close()
        print("disconnected" + "(COM" + str(self.selected_port) + ")")
    except:
        print("Error with COM-Port connection!")

def createmessage(self, datastruct):
    datastruct._lock.acquire()
    #bogie modes
    NormalMode = 200
    ErrorSetMode = 90
    CalibrationMode = 150
    ControllerOnOff = 120
    ControllerSet = 80

    message = bytearray(0)
    message.append(146) #select maintobogie protocol
    message.append(0) #preserve the first two bytes for the message length
    message.append(0)

    #select mode dependent message for bogies
    for bogie in range(len(datastruct.bogies)): #for every bogiewheel (6)
        if datastruct.bogies[bogie].sendData == True:
            if datastruct.bogies[bogie].mode == NormalMode:
                sendNormalMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].setangle,
                                datastruct.bogies[bogie].setspeed,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ErrorSetMode:
                sendErrorSetMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.seterror,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == CalibrationMode:
                sendCalibrationMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ControllerOnOff:
                sendControllerOnOff(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].setControllerOnOff,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ControllerSet:
                sendControllerSet(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].PID_data,
                                datastruct.errorextern)
        else:
            print("Error: no/wrong mode selected")
            raise ValueError
    else:
        message.append(0)

    if datastruct.pantilt.PanTiltset == 1:
        message.append(1)
```

```

        message.append(130)
        message.append(6)
        message.append(datastruct.pantilt.Pan) #pan
        message.append(datastruct.pantilt.Tilt) #tilt
        message.append(0) #Error
        message.append(0) #CRC
    else:
        message.append(1)

    #last bits no data (for now)
    message.append(0) #TODO: IMU
    message.append(0) #Middle Board
    message.append(0) #Error
    message.append(0) # TODO: CRC

    #determine message length
    message[1] = ((len(message) & 255))
    message[2] = ((len(message) >>8))

    datastruct._lock.release()
    return message #return finished message

def SendData(self, data): #send LRM data
    try:
        self.ser.flushInput()
        for i in data:
            self.ser.write(struct.pack('>B', i))
            #print(i, end=" ")
        print("")
    except:
        print("Error: Attempting to use a port that is not open")

def ReadData(self): #returns body message
    print("")

# =====
#                               ModeHandling Functions (@irrg_va)
# =====
def BogieModusSort(DataBogie,bogie,datastruct):
    datastruct._lock.acquire()
    NormalMode = 200
    ErrorSetMode = 90
    CalibrationMode = 150
    ControllerOnOff = 120
    ControllerSet = 80
    i = 0
    modus = DataBogie[i] & 0xfe
    i += 2 #skipp mode + length
    if NormalMode == modus:
        datastruct.bogies[bogie].realangle = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].realspeed = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].anglebogie = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentsteer = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].magencodersteerinc = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].flagcounterdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ErrorSetMode == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        #datastruct.errorextern = DataBogie[i] #TODO: should be errorextern
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif CalibrationMode == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ControllerOnOff == modus:
        #ControllerOnOff same Data as with NormalMode
        datastruct.bogies[bogie].realangle = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].realspeed = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].anglebogie = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentsteer = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].magencodersteerinc = DataBogie[i] | (DataBogie[i+1] << 8)

```

```

        i += 2
        datastruct.bogies[bogie].flagcounterdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ControllerSet == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    else:
        logging.info("Error unknown mode received: " + str(modus))
        datastruct._lock.release()

# =====
#                               SendModes Functions (@irrg_va)
# =====
def sendNormalMode(msg, wheel, angleint, speedint, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(200+wheel)
    msg.append(8)
    msg.append(angleint & 255) #angle low byte
    msg.append(angleint >> 8) #angle high byte
    msg.append(speedint & 255) #speed low byte
    msg.append(speedint >> 8) #speed high byte
    msg.append(ErrorExtern)
    msg.append(0)

def sendErrorSetMode(msg, wheel, SetError, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(90+wheel)
    msg.append(6)
    msg.append(SetError)
    msg.append(0)
    msg.append(ErrorExtern)
    msg.append(0)

def sendCalibrationMode(msg, wheel, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(150+wheel)
    msg.append(4)
    msg.append(ErrorExtern)
    msg.append(0)

def sendControllerOnOff(msg, wheel, Controller_OnOff, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(120+wheel)
    msg.append(5)
    msg.append(Controller_OnOff) #1=controller on, 0=controller off
    msg.append(ErrorExtern)
    msg.append(0)

# =====
#                               User Functions (@rica_ad)
# =====
def setangle(bogiewheel, angle, datastruct, com):
    # set the error first before commanding angle
    datastruct.bogies[bogiewheel].mode=90
    com.SendData(com.createmessage(datastruct))
    time.sleep(communication_time)

    # set the angle
    datastruct.bogies[bogiewheel].mode=200
    datastruct.bogies[bogiewheel].setangle = angle
    com.SendData(com.createmessage(datastruct))
    time.sleep(communication_time)

def setspeed(bogiewheel, speed, datastruct, com):
    # set the error first before commanding speed
    datastruct.bogies[bogiewheel].mode=90
    com.SendData(com.createmessage(datastruct))
    showbogiedata(datastruct)
    time.sleep(communication_time)

    # set the speed
    datastruct.bogies[bogiewheel].mode=200
    datastruct.bogies[bogiewheel].setspeed = speed
    com.SendData(com.createmessage(datastruct))
    showbogiedata(datastruct)
    time.sleep(communication_time)

def setstop(datastruct, com, delay):
    # it will only stop the speed, the angles keep the same value
    for j in range (delay):
        for i in range (6):
            datastruct.bogies[i].setspeed = 0
            com.SendData(com.createmessage(datastruct))
            time.sleep(communication_time)

```

```
#def showbogiedata(datastruct):
#     print("bogienum:", datastruct.bogies[i].bogie_ID, end=" ")
#     print("wheelnum:", datastruct.bogies[i].wheel_ID, end=" ")
#     print("setangle:", datastruct.bogies[i].setangle, end=" ")
#     print("setspeed:", datastruct.bogies[i].setspeed, end=" ")
#     print("pan:", datastruct.pantilt.Pan, end=" ")
#     print("tilt:", datastruct.pantilt.Tilt, end=" ")

#     print("v2 ",speed," a2: ",angle,"|v0 ",speed_wheels[1]," a0: ",angle_wheels[0])
#     print("v3 ",speed," a3: ",angle_wheels[0],"|v1 ",speed_wheels[1]," a1: ",angle_wheels[0])
#     print("v5 ",speed," a5: ",angle_wheels[0],"|v4 ",speed_wheels[1]," a4: ",angle_wheels[0])

#     print("realangle:", datastruct.bogies[i].realangle, end=" ")
#     print("realspeed:", datastruct.bogies[i].realspeed, end=" ")
#     print("anglebogie:", datastruct.bogies[i].anglebogie, end=" ")
#     print("currentsteer:", datastruct.bogies[i].motorcurrentsteer, end=" ")
#     print("currentdrive:", datastruct.bogies[i].motorcurrentdrive, end=" ")
#     print("magencsteerinc:", datastruct.bogies[i].magencodersteerinc, end=" ")
#     print("flagcounterdrive:", datastruct.bogies[i].flagcounterdrive, end=" ")
#     print("errorintern:", datastruct.bogies[i].errorintern, end=" ")
#     print("crc_check:", datastruct.bogies[i].crc_check)

# =====
#                               Threads Functions
# =====

def stopcom():
    global ComOnOff
    ComOnOff = False

def startcom():
    global ComOnOff
    ComOnOff = True
    ISR_com = threading.Thread(target=ISR_COM, args=(1,))
    ISR_com.start()

# =====
#                               MAIN Function
# =====

def main(args):
    clnt = ln.client(sys.argv[0], sys.argv[1:])    # links and nodes client
    # for the gamepad msgs (Controller 1)
    port_gamepad = clnt.subscribe("Controller_1", "md_controller")
    # for the Simulink model msgs
    port_Bogie_Wheel = ["","","","","",""]
    port_Bogie_Wheel[0] = clnt.subscribe("DataIN_Bogie0_Wheel0", "md_Simulink_to_LN_Wheel")
    port_Bogie_Wheel[1] = clnt.subscribe("DataIN_Bogie0_Wheel1", "md_Simulink_to_LN_Wheel")
    port_Bogie_Wheel[2] = clnt.subscribe("DataIN_Bogie1_Wheel0", "md_Simulink_to_LN_Wheel")
    port_Bogie_Wheel[3] = clnt.subscribe("DataIN_Bogie1_Wheel1", "md_Simulink_to_LN_Wheel")
    port_Bogie_Wheel[4] = clnt.subscribe("DataIN_Bogie2_Wheel0", "md_Simulink_to_LN_Wheel")
    port_Bogie_Wheel[5] = clnt.subscribe("DataIN_Bogie2_Wheel1", "md_Simulink_to_LN_Wheel")
    # for the navigation msgs (Controller 2)
    port_navigation = clnt.subscribe("lrm1.cmd_vel_autonomy", "gen/geometry_msgs/Twist")
    # for the pantilt msgs (to keep it separated for ROS for the tf tree)
    portOUT_PanTilt = clnt.publish("lrm1.pantilt", "md_PanTilt")

    print("Rover Communication initialized...")

    # Initialization of some functions for the communication structure and connection
    lrm = LRM_DataStruct()
    c=Communication()
    c.connect()

    # Set to error, like a reset, to start the normal mode afterwards
    for i in range(6):
        lrm.bogies[i].mode=90
    c.SendData(c.createmessage(lrm))
    time.sleep(communication_time)

    # Set normal mode, ready to be commanded by gamepad or navigation stack
    for j in range(20):
        for i in range(6):
            lrm.bogies[i].mode=200
            lrm.bogies[i].setangle = 0
        lrm.pantilt.Pan = calibrated_pantilt[0]
        lrm.pantilt.Tilt = calibrated_pantilt[1]
        c.SendData(c.createmessage(lrm))
        time.sleep(communication_time)
    setstop(lrm,c,10)

# =====
#                               MAIN LOOP
# =====

while True:
    # obtain the gamepad parameters
    port_gamepad.read()
    #angle = int(port_gamepad.packet.rotation)
    #body_vel = float(port_gamepad.packet.vel_x)
    #rover_mode = int(port_gamepad.packet.mode)
```



```

pan = int(port_gamepad.packet.Pan)
tilt = int(port_gamepad.packet.Tilt)
drive_mode = int(port_gamepad.packet.drive_mode)
assume = int(port_gamepad.packet.Assume)
permission = int(port_gamepad.packet.Permission)

# initializing values gotten from Simulink
steering_angle = np.zeros(6)
steering_angle_deg = np.zeros(6)
speed = np.zeros(6)
speed_m_s = np.zeros(6)
rover_mode = np.zeros(6)

# initialization of navigation variables
nav_vx = 0.0 # linear velocity in x
nav_vy = 0.0 # linear velocity in y
nav_wz = 0.0 # angular velocity in z

# initialization of calculated variables
body_w = 0.0 # body angular velocity (calculated, not provided)
body_vel = 0.0
wheelV_m_s = np.zeros(6)
wheelV_value = np.zeros(6)
wheelW_rad_s = np.zeros(6)
wheelAngle_degrees = np.zeros(6)

# reception of values from Simulink
for i in range(6):
    port_Bogie_Wheel[i].read()
    steering_angle[i] = int(port_Bogie_Wheel[i].packet.setAngle)
    speed[i] = float(port_Bogie_Wheel[i].packet.setMeterPerSecond)
    rover_mode[i] = int(port_Bogie_Wheel[i].packet.setMode)

    # auxiliary variables needed for technical issues
    speed_m_s[i] = float(speed[i]*bodyV_max_m_s)
    steering_angle_deg[i] = int(steering_angle[i])

# Already, the calculated speed from the Simulink model
for i in range(6):
    wheelV_m_s[i] = speed_m_s[i] # From the "Simulink model"
    wheelAngle_degrees[i] = steering_angle_deg[i] # From the "Simulink model"
    wheelV_m_s[i] = float(wheelV_m_s[i]) # Transformed it to float, coding purposes
    wheelW_rad_s[i] = wheelV_m_s[i]/(wheelDiam_m/2) # Conversion to w for each wheel

    # Conversion from e.g. 1.0 to 15000 (and float to int)
    wheelV_value[i] = int(wheelV_m_s[i]*wheelV_max_value/bodyV_max_m_s)

    # Restriction values
    if wheelV_value[i] > wheelV_max_value:
        wheelV_value[i] = wheelV_max_value
    elif wheelV_value[i] < -wheelV_max_value:
        wheelV_value[i] = -wheelV_max_value

    # If the value is negative, binary-complement (value to positive)
    # since the functions can't receive negative values
    if wheelV_value[i] < 0:
        wheelV_value[i] = int(bin(2**16+int(wheelV_value[i]))[-16:],2)

    # Restricting values to dump the printing part for a nicer view
    if abs(wheelW_rad_s[i]) < 0.1:
        wheelW_rad_s[i] = 0.0

    if drive_mode == 2:
        # ----- Formulas to calculate the body angular velocity -----
        # body_w = (Vr_robot-Vl_robot/robot_rotatonDiam_m) or 2V/Drobot
        # V = w*r, meaning that: Vr_robot or Vl_robot = wheelR_rad_s*(wheelD_m/2)
        # then: body_w = 2*wheelR_rad_s*(wheelD_m/2)/robot_rotationDiam_m
        # finally:
        body_w = wheelW_rad_s[0]*wheelDiam_m/robot_rotationDiam_m

# printed values
print("W2: %d rad/s %d W5: %d rad/s %d \n" \
      "W3: %d rad/s %d W4: %d rad/s %d \n" \
      "W0: %d rad/s %d W1: %d rad/s %d \n" \
      "drive_mode: %d rover_mode: %d pan: %d tilt: %d assume: %d permission: %d\n" \
      "From GamePad / Navigation Stack: \n" \
      "nav_vx: %0.3f m/s nav_vy: %0.3f m/s nav_wz: %0.3f rad/s \n" \
      "Calculated:\n" \
      "body_vel: %0.3f m/s body_w: %0.3f rad/s\n" \
      "%(wheelV_value[2],wheelAngle_degrees[2],wheelV_value[5],wheelAngle_degrees[5],\n" \
      "wheelV_value[3],wheelAngle_degrees[3],wheelV_value[4],wheelAngle_degrees[4],\n" \
      "wheelV_value[0],wheelAngle_degrees[0],wheelV_value[1],wheelAngle_degrees[1],\n" \
      "drive_mode,rover_mode[0],pan,tilt,assume,permission,\n" \
      "nav_vx,nav_vy,nav_wz,body_vel,body_w))

# writing into the Main Board
for i in range(6):
    lrm.bogies[i].setspeed = int(wheelV_value[i])
    lrm.bogies[i].setangle = int(wheelAngle_degrees[i])

```

```
lrm.pantilt.Pan = int(pan)
lrm.pantilt.Tilt = int(tilt)

# PanTilt implementation for rostopic2ln, ln2ros
portOUT_PanTilt.packet.Pan = int(pan)
portOUT_PanTilt.packet.Tilt = int(tilt)
portOUT_PanTilt.write()

# Writing data
c.SendData(c.createmessage(lrm))
time.sleep(communication_time)

# Have fun! :)
```

rover_communication no Simulink

This Rover.py code works for both controllers: the GamePad and the autonomous navigation.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Started on March 21 2022
@author: irrg_va
Finished on Jan 31 2023
@author: rica_ad
"""

# LIBRARIES
import logging
import threading
import time
import array
import sys
import serial
import serial.tools.list_ports
import glob
import struct
import numpy as np
import math
import links_and_nodes as ln
import time

# ----- Global Variables -----
wheelDiam_m = 0.0534          # wheel diameter in meters old rover (black)
robot_rotationDiam_m = 0.27  # it is the distance from the center to the corner wheels
                                # used to calculate Rotation mode velocity
body_w = 0.0                 # initializer of the body angular velocity

bodyV_max_m_s = 0.115        # from experiments 09.11.2022 in lab, meters per sec
wheelV_max_value = 15000     # found experimentally, maybe til 16000, but better set to this one

calibrated_pantilt = [139,164] # experimentally set to get a good mapping 09.11.2022
communication_time = 1./1000. # to give time in secs to the controller to command the velocity

mapping_bodyV_max_m_s = 0.05  # TODO: to cross-check the value
mapping_bodyW_max_rad_s = 0.25 # TODO: to cross-check the value
mapping_bodyW_min_rad_s = 0.025 # TODO: to cross-check the value

# *****
#                               Data Classes (@irrg_va)
# *****
class LRM_DataStruct:
    def __init__(self):
        self._lock = threading.Lock()

        self.errorextern = 0      # errorvalue
        self.seterror = 0        # errorclearvalue for bogies

        self.bogie00 = self.Bogie(0,0)    # create all the bogies
        self.bogie01 = self.Bogie(0,1)
        self.bogie10 = self.Bogie(1,0)
        self.bogie11 = self.Bogie(1,1)
        self.bogie20 = self.Bogie(2,0)
        self.bogie21 = self.Bogie(2,1)
        self.bogies = []              # put the bogies in a list
        self.bogies.append(self.bogie00)
        self.bogies.append(self.bogie01)
        self.bogies.append(self.bogie10)
        self.bogies.append(self.bogie11)
        self.bogies.append(self.bogie20)
        self.bogies.append(self.bogie21)

        self.pantilt = self.PanTilt()    # set the pantilt
```

```

class Bogie:
    def __init__(self, Bogie_ID, Wheel_ID):
        #set bogie data
        # default mode = normalmode
        self.mode = 200
        self.bogie_ID = Bogie_ID
        self.wheel_ID = Wheel_ID
        self.setangle = 0
        self.setspeed = 0
        #get bogie data
        self.reangle = 0
        self.realspeed = 0
        self.anglebogie = 0          #only in bogiewheel X0 (X=0/1/2)
        self.motorcurrentsteer = 0
        self.motorcurrentdrive = 0
        self.magencodesteerinc = 0 # magnetic encoder steer increments
        self.flagcounterdrive = 0
        self.errorintern = 0         # gets send as ERRtemp
        self.crc_check = 0
        #status bogie
        self.sendData = True
        self.setControllerOnOff = 1
        #PID controller data
        self.PID_data = self.PID_Data()

class PID_Data:
    def __init__(self):
        # TODO: Note that these default values are also hard coded in the
        # bogie code and where determined purely experimentally (irrg_va)
        self.DP_i = 36 #Drive Parameter P inner loop
        # Drive PIDinit: 36, 4, 0,11,0,0, 65000 ,65000, 4, 2
        self.DI_i = 4 #Drive Parameter I inner loop
        self.DD_i = 0 #Drive Parameter D inner loop
        self.DP_o = 11 #Drive Parameter P outer loop
        self.DI_o = 0 #Drive Parameter I outer loop
        self.DD_o = 0 #Drive Parameter D outer loop
        self.SP_i = 4113 #Steer Parameter P inner loop
        # SteerPIDinit: 4113,0,0 ,8912,0,0, 65000 , 65000, 4, 1
        self.SI_i = 0 #Steer Parameter I inner loop
        self.SD_i = 0 #Steer Parameter D inner loop
        self.SP_o = 8912 #Steer Parameter P outer loop
        self.SI_o = 0 #Steer Parameter I outer loop
        self.SD_o = 0 #Steer Parameter D outer loop
        self.ScaleD_i = 4 #Drive Scaling inner loop
        self.ScaleS_i = 4 #Steer Scaling inner loop
        self.ScaleD_o = 2 #Drive Scaling outer loop
        self.ScaleS_o = 1 #Steer Scaling outer loop
        self.WD_o = 65000 #Drive Windup outer loop
        self.WD_i = 65000 #Drive Windup inner loop
        self.WS_o = 65000 #Steer Windup outer loop
        self.WS_i = 65000 #Steer Windup inner loop
        # list PID parameter for systematic access: S= Steer, D = Drive, I/i = Inner Loop,
        # O/o = Outer Loop #TODO:simplify parameter names
        self.list_S_I = [self.SP_i, self.SI_i, self.SD_i, self.ScaleS_i, self.WS_i]
        self.list_S_O = [self.SP_o, self.SI_o, self.SD_o, self.ScaleS_o, self.WS_o]
        self.list_D_I = [self.DP_i, self.DI_i, self.DD_i, self.ScaleD_i, self.WD_i]
        self.list_D_O = [self.DP_o, self.DI_o, self.DD_o, self.ScaleD_o, self.WD_o]
        self.list_S_IO = [self.list_S_I, self.list_S_O]
        self.list_D_IO = [self.list_D_I, self.list_D_O]
        self.list_SteerDrive = [self.list_S_IO, self.list_D_IO]

class PanTilt():
    def __init__(self):
        self.PanTiltset = 1
        self.Pan = calibrated_pantilt[0]
        self.Tilt = calibrated_pantilt[1]

# *****
# Communication Class and Functions (@irrg_va)
# =====
class Communication:
    def __init__(self):
        self.ser = 0
        self.selected_port = 0

    def connect(self):
        #get all the available ports and print them
        if sys.platform.startswith('win'):
            ports = ['COM%s' % (i + 1) for i in range(256)]
        elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
            # this excludes your current terminal "/dev/tty"
            ports = glob.glob('/dev/tty[A-Za-z]*')
        elif sys.platform.startswith('darwin'):
            ports = glob.glob('/dev/tty.*')
        else:
            raise EnvironmentError('Unsupported platform')
        result = []
        for port in ports:
            try:

```

```

        s = serial.Serial(port)
        s.close()
        result.append(port)
    except (OSError, serial.SerialException):
        pass
for lauf in result :
    print(lauf)

self.selected_port = 0 # input("Select Port: "), can be user input, but better set to 0
try:
    self.ser = serial.Serial(result[int(self.selected_port)])
    if self.ser.isOpen():
        print(self.ser.portstr + " is enabled!\n")

        self.ser.baudrate = 230400
        self.ser.timeout = 2
        self.ser.bytesize = serial.EIGHTBITS
        self.ser.parity = serial.PARITY_NONE
        self.ser.stopbits = serial.STOPBITS_ONE
    except:
        print("could not connect to"+result[self.selected_port])

def disconnect(self):
    try:
        self.ser.close()
        print("disconnected" + " (COM" + str(self.selected_port) + ")")
    except:
        print("Error with COM-Port connection!")

def createmessage(self, datastruct):
    datastruct._lock.acquire()
    #bogie modes
    NormalMode = 200
    ErrorSetMode = 90
    CalibrationMode = 150
    ControllerOnOff = 120
    ControllerSet = 80

    message = bytearray(0)
    message.append(146) #select maintobogie protocol
    message.append(0) #preserve the first two bytes for the message length
    message.append(0)

    #select mode dependent message for bogies
    for bogie in range(len(datastruct.bogies)): #for every bogiewheel (6)
        if datastruct.bogies[bogie].sendData == True:
            if datastruct.bogies[bogie].mode == NormalMode:
                sendNormalMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].setangle,
                                datastruct.bogies[bogie].setspeed,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ErrorSetMode:
                sendErrorSetMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.seterror,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == CalibrationMode:
                sendCalibrationMode(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ControllerOnOff:
                sendControllerOnOff(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].setControllerOnOff,
                                datastruct.errorextern)
            elif datastruct.bogies[bogie].mode == ControllerSet:
                sendControllerSet(message,
                                datastruct.bogies[bogie].wheel_ID,
                                datastruct.bogies[bogie].PID_data,
                                datastruct.errorextern)
        else:
            print("Error: no/wrong mode selected")
            raise ValueError
    else:
        message.append(0)

    if datastruct.pantilt.PanTiltset == 1:
        message.append(1)
        message.append(130)
        message.append(6)
        message.append(datastruct.pantilt.Pan) #pan
        message.append(datastruct.pantilt.Tilt) #tilt
        message.append(0) #Error
        message.append(0) #CRC
    else:
        message.append(1)

    #last bits no data (for now)

```

```

message.append(0) #TODO: IMU
message.append(0) #Middle Board
message.append(0) #Error
message.append(0) # TODO: CRC

#determine message length
message[1] = ((len(message) & 255))
message[2] = ((len(message) >>8))

datastruct._lock.release()
return message #return finished message

def SendData(self, data): #send LRM data
try:
    self.ser.flushInput()
    for i in data:
        self.ser.write(struct.pack('>B', i))
        #print(i, end=" ")
    print("")
except:
    print("Error: Attempting to use a port that is not open")

def ReadData(self): #returns body message
print("")

# =====
#                               ModeHandling Functions (@irrg_va)
# =====
def BogieModusSort(DataBogie,bogie,datastruct):
    datastruct._lock.acquire()
    NormalMode = 200
    ErrorSetMode = 90
    CalibrationMode = 150
    ControllerOnOff = 120
    ControllerSet = 80
    i = 0
    modus = DataBogie[i] & 0xfe
    i += 2 #skipp mode + length
    if NormalMode == modus:
        datastruct.bogies[bogie].realangle = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].realspeed = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].anglebogie = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentsteer = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].magencodersteerinc = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].flagcounterdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ErrorSetMode == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        #datastruct.errorextern = DataBogie[i] #TODO: should be errorextern
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif CalibrationMode == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ControllerOnOff == modus:
        #ControllerOnOff same Data as with NormalMode
        datastruct.bogies[bogie].realangle = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].realspeed = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].anglebogie = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentsteer = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].motorcurrentdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].magencodersteerinc = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].flagcounterdrive = DataBogie[i] | (DataBogie[i+1] << 8)
        i += 2
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]
    elif ControllerSet == modus:
        datastruct.bogies[bogie].errorintern = DataBogie[i]
        i += 1
        datastruct.bogies[bogie].crc_check = DataBogie[i]

```

```

    else:
        logging.info("Error unknown mode received: " + str(modus))
        datastruct._lock.release()

# =====
#                               SendModes Functions (@irrg_va)
# =====
def sendNormalMode(msg, wheel, angleint, speedint, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(200+wheel)
    msg.append(8)
    msg.append(angleint & 255) #angle low byte
    msg.append(angleint >> 8) #angle high byte
    msg.append(speedint & 255) #speed low byte
    msg.append(speedint >> 8) #speed high byte
    msg.append(ErrorExtern)
    msg.append(0)

def sendErrorSetMode(msg, wheel, SetError, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(90+wheel)
    msg.append(6)
    msg.append(SetError)
    msg.append(0)
    msg.append(ErrorExtern)
    msg.append(0)

def sendCalibrationMode(msg, wheel, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(150+wheel)
    msg.append(4)
    msg.append(ErrorExtern)
    msg.append(0)

def sendControllerOnOff(msg, wheel, Controller_OnOff, ErrorExtern):
    msg.append(1) #sendData == True
    msg.append(120+wheel)
    msg.append(5)
    msg.append(Controller_OnOff) #1=controller on, 0=controller off
    msg.append(ErrorExtern)
    msg.append(0)

# =====
#                               User Functions (@rica_ad)
# =====
def ackerman(angle, body_vel):
    quadrant = 0 # from 1 to 4, 0 is undetermined, but initialized
    # 1 | 4
    # ----
    # 2 | 3
    k = 0.7 # normal ackerman is abrupt, this is to smooth the data a bit
    w = 0.0 # initialization of body angular speed

    # Output Values initialization
    steering_angle = np.zeros(6)
    speed = np.zeros(6)

    # angle restriction for each quadrant to not more than 30 respectively
    if angle >= 30 and angle < 90:
        angle = 30
    elif angle >= 90 and angle <= 150:
        angle = 150
    elif angle >= 210 and angle < 270:
        angle = 210
    elif angle >= 270 and angle <= 330:
        angle = 330

    # assignment of quadrants depending on provided angle
    if angle >= 0 and angle < 90:
        quadrant = 1
    elif angle >= 90 and angle < 180:
        quadrant = 2
    elif angle >= 180 and angle < 270:
        quadrant = 3
    elif angle >= 270 and angle < 360:
        quadrant = 4

    # LRM Parameters
    D = 0.085 # distance to 1/2 Tread of vehicle in m
    L = 0.102 # distance to 1/2 Wheel base of vehicle in m

    # Ackerman Geometry
    # For more information see: DOI: 10.1109/ICCAS.2008.4694549

    # if the angle is between these angles, calculate mathematically:
    if (angle > 2 and angle < 178) or (angle > 182 and angle < 358): #to prevent nan
        R = L/math.tan(math.radians(angle)) #turning radius vehicle
        w = body_vel/R # body angular velocity

```

```

r_fi = math.sqrt((R-D)*(R-D)+L*L)    #radius front inner
r_fo = math.sqrt((R+D)*(R+D)+L*L)    #radius front outer
r_ci = R-D                            #radius center inner
r_co = R+D                            #radius center outer

v_fi = r_fi*w                         #speed front inner
v_fo = r_fo*w                         #speed front outer
v_ci = r_ci*w                         #speed center inner
v_co = r_co*w                         #speed center outer
v_ri = v_fi                           #speed rear inner
v_ro = v_fo                           #speed rear outer

if angle == 0:
    phi_fi = 0
    phi_fo = 0
    phi_ri = 0
    phi_ro = 0
else:
    phi_fi = math.degrees(math.atan(L/(R-D)))    #steering angle front inner
    phi_fo = math.degrees(math.atan(L/(R+D)))    #steering angle front outer
    phi_ri = phi_fi                             #steering angle rear inner
    phi_ro = phi_fo                             #steering angle rear outer
    #steering angle for center is zero

# wheel order in matlab: 1 = fr, 2 = cr, 3 = rr, 4 = rl, 5 = cl, 6 = fl
# wheel order in matlab: 0 = rl, 1 = rr, 2 = fl, 3 = cl, 4 = cr, 5 = fr

# depending on each quadrant, the angle logic is different
if quadrant == 1:
    steering_angle[0] = 360-phi_ri # before phi_fo in matlab
    steering_angle[1] = 360-phi_ro # before 0 in matlab
    steering_angle[2] = phi_fi     # before 360-phi_ro in matlab
    steering_angle[3] = 0          # before 360-phi_ri in matlab
    steering_angle[4] = 0
    steering_angle[5] = phi_fo     # before phi_fi in matlab
elif quadrant == 2:
    steering_angle[0] = 360+phi_ro # before phi_fi in matlab
    steering_angle[1] = 360+phi_ri # before 0 in matlab
    steering_angle[2] = phi_fo     # before 360+phi_ri in matlab
    steering_angle[3] = 0          # before 360+phi_ro in matlab
    steering_angle[4] = 0
    steering_angle[5] = phi_fi     # before phi_fo in matlab
elif quadrant == 3:
    steering_angle[0] = phi_ro     # before 360-phi_fi in matlab
    steering_angle[1] = phi_ri     # before 0 in matlab
    steering_angle[2] = 360-phi_fo # before phi_ri in matlab
    steering_angle[3] = 0          # before phi_ro in matlab
    steering_angle[4] = 0
    steering_angle[5] = 360-phi_fi # before 360-phi_fo in matlab
elif quadrant == 4:
    steering_angle[0] = phi_ri     # before 360+phi_fo in matlab
    steering_angle[1] = phi_ro     # before 0 in matlab
    steering_angle[2] = 360+phi_fi # before phi_ro in matlab
    steering_angle[3] = 0          # before phi_ri in matlab
    steering_angle[4] = 0
    steering_angle[5] = 360+phi_fo # before 360+phi_fi in matlab

# velocity multiplied by k to reduce it, because it is too abrupt
# max vel is 15000, and it was almost always giving 15000
# TODO: maybe change this in matlab, this was done in python
if quadrant == 1:
    speed[0] = v_ri*k
    speed[1] = v_ro*k
    speed[2] = v_fi*k
    speed[3] = v_ci*k
    speed[4] = v_co*k
    speed[5] = v_fo*k
elif quadrant == 2:
    speed[0] = v_ro*k
    speed[1] = v_ri*k
    speed[2] = v_fo*k
    speed[3] = -v_co*k
    speed[4] = -v_ci*k
    speed[5] = v_fi*k
elif quadrant == 3:
    speed[0] = -v_ro*k
    speed[1] = -v_ri*k
    speed[2] = -v_fo*k
    speed[3] = -v_co*k
    speed[4] = -v_ci*k
    speed[5] = -v_fi*k
elif quadrant == 4:
    speed[0] = -v_ri*k
    speed[1] = -v_ro*k
    speed[2] = -v_fi*k
    speed[3] = v_ci*k
    speed[4] = v_co*k
    speed[5] = -v_fo*k
else:

```



```

    for i in range(6):
        steering_angle[i] = 0 # if the angle is 0, then don't do tangent, just do 0

    # and the speed will only take the cosine (fwd and bckwr) to make it more logic
    # and also, it is smoothed with the k factor
    speed[0] = body_vel*math.cos(math.radians(angle))*k

    for i in range(5):
        speed[i+1] = speed[0] # since all wheels are going the same direction, assign it

    for i in range(6):
        steering_angle[i] = abs(steering_angle[i]) # negatives don't exist, only 0-360

    return steering_angle, speed, w

def rotation(body_vel):
    D = 0.085 # distance to 1/2 Tread of vehicle in m
    L = 0.102 # distance to 1/2 Wheel base of vehicle in m
    r_f = math.sqrt(D*D+L*L) # Pitagoras
    r_c = D
    w = body_vel/r_f
    v_fl = body_vel
    v_cl = body_vel
    v_rl = body_vel
    v_fr = -body_vel
    v_cr = -body_vel
    v_rr = -body_vel

    # angles already set to rotate correctly
    phi_fl = 360-60
    phi_cl = 0
    phi_rl = 60
    phi_rr = 360-60
    phi_cr = 0
    phi_fr = 60

    # Output Values initialization
    steering_angle = np.zeros(6)
    speed = np.zeros(6)

    # TODO: maybe change this in matlab, this was done in python
    steering_angle[0] = phi_fr
    steering_angle[1] = phi_fl
    steering_angle[2] = phi_rr
    steering_angle[3] = 0
    steering_angle[4] = 0
    steering_angle[5] = phi_rl

    # TODO: maybe change this in matlab, this was done in python
    speed[0] = v_fr
    speed[1] = v_fl
    speed[2] = v_rr
    speed[3] = v_cr
    speed[4] = v_cl
    speed[5] = v_rl

    return steering_angle, speed

def crabwalk(angle, body_vel):
    # Output Values initialization
    steering_angle = np.zeros(6)
    speed = np.zeros(6)

    if angle < 0:
        angle = 360-angle # if the angle decreases more, then just add the 360

    # in crabwalk, all wheels are the same in speed and angle
    for i in range(6):
        steering_angle[i] = angle
        speed[i] = body_vel

    return steering_angle, speed

def setangle(bogiewheel, angle, datastruct, com):
    # set the error first before commanding angle
    datastruct.bogies[bogiewheel].mode=90
    com.SendData(com.createmessage(datastruct))
    time.sleep(communication_time)

    # set the angle
    datastruct.bogies[bogiewheel].mode=200
    datastruct.bogies[bogiewheel].setangle = angle
    com.SendData(com.createmessage(datastruct))
    time.sleep(communication_time)

def setspeed(bogiewheel, speed, datastruct, com):
    # set the error first before commanding speed
    datastruct.bogies[bogiewheel].mode=90
    com.SendData(com.createmessage(datastruct))

```

```

showbogiedata(datastruct)
time.sleep(communication_time)

# set the speed
datastruct.bogies[bogiewheel].mode=200
datastruct.bogies[bogiewheel].setspeed = speed
com.SendData(com.createmessage(datastruct))
showbogiedata(datastruct)
time.sleep(communication_time)

def setstop(datastruct,com,delay):
    # it will only stop the speed, the angles keep the same value
    for j in range (delay):
        for i in range (6):
            datastruct.bogies[i].setspeed = 0
            com.SendData(com.createmessage(datastruct))
            time.sleep(communication_time)

#def showbogiedata(datastruct):
#    print("bogienum:", datastruct.bogies[i].bogie_ID, end=" ")
#    print("wheelnum:", datastruct.bogies[i].wheel_ID, end=" ")
#    print("setangle:", datastruct.bogies[i].setangle, end=" ")
#    print("setspeed:", datastruct.bogies[i].setspeed, end=" ")
#    print("pan:", datastruct.pantilt.Pan, end=" ")
#    print("tilt:", datastruct.pantilt.Tilt, end=" ")

#    print("v2 ",speed," a2: ",angle,"|v0 ",speed_wheels[1]," a0: ",angle_wheels[0])
#    print("v3 ",speed," a3: ",angle_wheels[0],"|v1 ",speed_wheels[1]," a1: ",angle_wheels[0])
#    print("v5 ",speed," a5: ",angle_wheels[0],"|v4 ",speed_wheels[1]," a4: ",angle_wheels[0])

#    print("realangle:", datastruct.bogies[i].realangle, end=" ")
#    print("realspeed:", datastruct.bogies[i].realspeed, end=" ")
#    print("anglebogie:", datastruct.bogies[i].anglebogie, end=" ")
#    print("currentsteer:", datastruct.bogies[i].motorcurrentsteer, end=" ")
#    print("currentdrive:", datastruct.bogies[i].motorcurrentdrive, end=" ")
#    print("magencsteerinc:", datastruct.bogies[i].magencodersteerinc, end=" ")
#    print("flagcounterdrive:", datastruct.bogies[i].flagcounterdrive, end=" ")
#    print("errorintern:", datastruct.bogies[i].errorintern, end=" ")
#    print("crc_check:", datastruct.bogies[i].crc_check)

# =====
#                               Threads Functions
# =====

def stopcom():
    global ComOnOff
    ComOnOff = False

def startcom():
    global ComOnOff
    ComOnOff = True
    ISR_com = threading.Thread(target=ISR_COM, args=(1,))
    ISR_com.start()

# =====
#                               MAIN Function
# =====

def main(args):
    clnt = ln.client(sys.argv[0], sys.argv[1:])    # links and nodes client
    # for the gamepad msgs (Controller 1)
    port_gamepad = clnt.subscribe("Controller_1", "md_controller")
    # for the navigation msgs (Controller 2)
    port_navigation = clnt.subscribe("lrm1.cmd_vel_autonomy", "gen/geometry_msgs/Twist")
    # for the pantilt msgs (to keep it separated for ROS for the tf tree)
    portOUT_PanTilt = clnt.publish("lrm1.pantilt", "md_PanTilt")

    print("Rover Communication initialized...")

    # Initialization of some functions for the communication structure and connection
    lrm = LRM_DataStruct()
    c=Communication()
    c.connect()

    # Set to error, like a reset, to start the normal mode afterwards
    for i in range(6):
        lrm.bogies[i].mode=90
        c.SendData(c.createmessage(lrm))
        time.sleep(communication_time)

    # Set normal mode, ready to be commanded by gamepad or navigation stack
    for j in range(20):
        for i in range(6):
            lrm.bogies[i].mode=200
            lrm.bogies[i].setangle = 0
            lrm.pantilt.Pan = calibrated_pantilt[0]
            lrm.pantilt.Tilt = calibrated_pantilt[1]
            c.SendData(c.createmessage(lrm))
            time.sleep(communication_time)
        setstop(lrm,c,10)

```

```
# =====
#                               MAIN LOOP
# =====
while True:
    # obtain the gamepad parameters
    port_gamepad.read()
    angle = int(port_gamepad.packet.rotation)
    body_vel = float(port_gamepad.packet.vel_x)
    pan = int(port_gamepad.packet.Pan)
    tilt = int(port_gamepad.packet.Tilt)
    drive_mode = int(port_gamepad.packet.drive_mode)
    rover_mode = int(port_gamepad.packet.mode)
    #controller_on_off = port_gamepad.packet.controller_on_off
    assume = int(port_gamepad.packet.Assume)
    permission = int(port_gamepad.packet.Permission)

    # initialization of navigation variables
    nav_vx = 0.0 # linear velocity in x
    nav_vy = 0.0 # linear velocity in y
    nav_wz = 0.0 # angular velocity in z

    # initialization of calculated variables
    body_w = 0.0 # body angular velocity (calculated, not provided)
    steering_angle = np.zeros(6)
    speed = np.zeros(6)
    wheelV_m_s = np.zeros(6)
    wheelV_value = np.zeros(6)
    wheelW_rad_s = np.zeros(6)
    wheelAngle_degrees = np.zeros(6)

    # By default, green button A from gmpd is set to 0
    if permission == 0:
        # -----> GAMEPAD COMMANDING
        body_vel = body_vel*bodyV_max_m_s # Value coming from GamePad, 0.0 to 1.0
        if drive_mode == 1:
            steering_angle, speed, body_w = ackerman(angle,body_vel)
        elif drive_mode == 2:
            steering_angle, speed = rotation(body_vel)
            body_vel = 0 # because it is rotation and the rover doesn't have any vx,vy
        # ----- Formulas to calculate the body angular velocity -----
        # body_w = (Vr_robot-Vl_robot/robot_rotatonDiam_m) or 2V/Drobot
        # V = w*r, meaning that: Vr_robot or Vl_robot = wheelR_rad_s*(wheelD_m/2)
        # then: body_w = 2*wheelR_rad_s*(wheelD_m/2)/robot_rotationDiam_m
        # finally:
        body_w = wheelW_rad_s[0]*wheelDiam_m/robot_rotationDiam_m
        elif drive_mode == 3:
            steering_angle, speed = crabwalk(angle,body_vel)
    else:
        # -----> NAVIGATION STACK COMMANDING
        # obtain the navigation parameters
        port_navigation.read()
        nav_vx = float(port_navigation.packet.linear.x)
        nav_vy = float(port_navigation.packet.linear.y)
        nav_wz = float(port_navigation.packet.angular.z)

        # Conditioning of each driving mode from the navigation stack
        if nav_vx != 0 and nav_vy == 0 and nav_wz != 0:
            # Ackerman mode
            drive_mode = 1
        elif nav_vx == 0 and nav_vy == 0 and nav_wz != 0:
            # Rotation mode
            drive_mode = 2
        elif nav_vx != 0 and nav_vy != 0 and nav_wz == 0:
            # Crab mode
            drive_mode = 3

        # Locomotion calculation
        if drive_mode == 1:
            # quadrant analysis based on the directions in ROS/Rviz:
            # +x
            # +y -y
            # -x
            angle = nav_wz*180 # 180 is a K that I found to transform the wz into angle data
            if angle < 0:
                angle = angle+360 # if the angle is negative, just add 360
            # restricting the hardware linear velocity capabilities to map properly
            s = 0.5 # to smooth a bit more the translation, to restrict the hardware capabilities
            body_vel = nav_vx*s
            steering_angle, speed, body_w = ackerman(angle,body_vel)
        elif drive_mode == 2:
            s = 0.2 # to smooth a bit more the rotation, to restrict the hardware capabilities
            nav_wz_smooth = nav_wz*s

            # to dump the navigation command to a min value, if not, the rotation gets stuck
            if nav_wz_smooth > 0 and nav_wz_smooth < mapping_bodyW_min_rad_s:
                nav_wz_smooth = mapping_bodyW_min_rad_s
            elif nav_wz_smooth < 0 and nav_wz_smooth > -mapping_bodyW_min_rad_s:
                nav_wz_smooth = -mapping_bodyW_min_rad_s
```

```

    body_w = nav_wz_smooth
    steering_angle, speed = rotation(body_w)
    body_vel = 0 # because it is rotation and the rover doesn't have any vx nor vy
elif drive_mode == 3:
    s = 0.45 # to smooth a bit more the translation, to restrict the hardware capabilities
    body_vel = math.sqrt(pow(nav_vx,2)+pow(nav_vy,2))*s # Pitagoras
    angle = int(math.degrees(math.atan(nav_vy/nav_vx))) # vy/vx

    # quadrant analysis to get the proper angle direction
    if nav_vy > 0 and nav_vx < 0:
        angle = angle + 180
    elif nav_vy < 0 and nav_vx < 0:
        angle = angle + 180
    elif nav_vy < 0 and nav_vx > 0:
        angle = angle + 360

    steering_angle, speed = crabwalk(angle,body_vel)
    # body_vel is the same as the commanded from the wheels
    body_w = 0 # there is no body angular velocity since there is no rotation in crabwalk

# Already, the calculated speed from the "Simulink model" in m/s
for i in range(6):
    wheelV_m_s[i] = speed[i] # From the "Simulink model"
    wheelAngle_degrees[i] = steering_angle[i] # From the "Simulink model"
    wheelV_m_s[i] = float(wheelV_m_s[i]) # Transformed it to float, coding purposes
    wheelW_rad_s[i] = wheelV_m_s[i]/(wheelDiam_m/2) # Conversion to w for each wheel

# Conversion from e.g. 1.0 to 15000 (and float to int)
wheelV_value[i] = int(wheelV_m_s[i]*wheelV_max_value/bodyV_max_m_s)

# Restriction values
if wheelV_value[i] > wheelV_max_value:
    wheelV_value[i] = wheelV_max_value
elif wheelV_value[i] < -wheelV_max_value:
    wheelV_value[i] = -wheelV_max_value

# If the value is negative, binary-complement (value to positive)
# since the functions can't receive negative values
if wheelV_value[i] < 0:
    wheelV_value[i] = int(bin(2**16+int(wheelV_value[i]))[-16:],2)

# Restricting values to dump the printing part for a nicer view
if abs(wheelW_rad_s[i]) < 0.1:
    wheelW_rad_s[i] = 0.0

if drive_mode == 2:
    # ----- Formulas to calculate the body angular velocity -----
    # body_w = (Vr_robot-Vl_robot/robot_rotatonDiam_m) or 2V/Drobot
    # V = w*r, meaning that: Vr_robot or Vl_robot = wheelR_rad_s*(wheelD_m/2)
    # then: body_w = 2*wheelR_rad_s*(wheelD_m/2)/robot_rotationDiam_m
    # finally:
    body_w = wheelW_rad_s[0]*wheelDiam_m/robot_rotationDiam_m

# printed values
print("W2: %d rad/s %d W5: %d rad/s %d \n" \
      "W3: %d rad/s %d W4: %d rad/s %d \n" \
      "W0: %d rad/s %d W1: %d rad/s %d \n" \
      "drive_mode: %d rover_mode: %d pan: %d tilt: %d assume: %d permission: %d\n" \
      "From GamePad / Navigation Stack: \n" \
      "nav_vx: %0.3f m/s nav_vy: %0.3f m/s nav_wz: %0.3f rad/s \n" \
      "Calculated:\n" \
      "body_vel: %0.3f m/s body_w: %0.3f rad/s\n" \
      "%(wheelV_value[2],wheelAngle_degrees[2],wheelV_value[5],wheelAngle_degrees[5], \
      wheelV_value[3],wheelAngle_degrees[3],wheelV_value[4],wheelAngle_degrees[4], \
      wheelV_value[0],wheelAngle_degrees[0],wheelV_value[1],wheelAngle_degrees[1], \
      drive_mode,rover_mode,pan,tilt,assume,permission, \
      nav_vx,nav_vy,nav_wz,body_vel,body_w))

# writing into the Main Board
for i in range(6):
    lrm.bogies[i].setspeed = int(wheelV_value[i])
    lrm.bogies[i].setangle = int(wheelAngle_degrees[i])
lrm.pantilt.Pan = int(pan)
lrm.pantilt.Tilt = int(tilt)

# PanTilt implementation for rostopic2ln, ln2ros
portOUT_PanTilt.packet.Pan = int(pan)
portOUT_PanTilt.packet.Tilt = int(tilt)
portOUT_PanTilt.write()

# Writing data
c.SendData(c.createmessage(lrm))
time.sleep(communication_time)

# Have fun! :)

```

pantilt tf publisher code

This is the `pan_tilt_tf_publisher.py` code. This serves as a communication between the links and nodes world, to the ROS world to see the odometry of the d435i camera based on the movement of the servos (only coming from the gamepad so far, but intended to be received from the Simulink model too).

```
#!/usr/bin/env python

from __future__ import print_function

import math
import numpy as np
import threading

import rospy
# import tf
import tf2_ros
import tf_conversions
import geometry_msgs.msg

import links_and_nodes as ln

class FramePublisher:
    def __init__(self):
        rospy.init_node('lrm_pan_tilt_tf_publisher')

        #self.br = tf.TransformBroadcaster()
        self.br = tf2_ros.TransformBroadcaster()

        self.cint = ln.client("lrm_pan_tilt_tf_publisher")
        # where the gamepad msg is stored:
        self.port_gamepad = self.cint.subscribe("Controller_1", "md_controller")

        # to receive the msg 'paralelly':
        self.ln_sub = threading.Thread(target=self.ln_subscriber)
        self.ln_sub.daemon = True
        self.ln_sub.start()
        print("ready")

        rate = rospy.Rate(30)

    def ln_subscriber(self):
        first = True
        while not rospy.is_shutdown():
            self.port_gamepad.read()
            if first:
                print("got first ln message.")
                first = False
            self.send_transform(self.port_gamepad.packet)

    def send_transform(self, pkt):
        pan_angle = np.deg2rad(int(port_gamepad.packet.Pan))
        tilt_angle = np.deg2rad(int(port_gamepad.packet.Tilt))
        t = geometry_msgs.msg.TransformStamped()
        t.header.stamp = rospy.Time.now()
        t.header.frame_id = "base_link"
        t.child_frame_id = "lrm_camera_mount"
        t.transform.translation.x = 0.112 # red axis, taken from CAD
        t.transform.translation.y = 0 # green axis
        t.transform.translation.z = 0.2 # blue axis, taken from CAD
        q = tf_conversions.transformations.quaternion_from_euler(0, tilt_angle, pan_angle)
        t.transform.rotation.x = q[0]
        t.transform.rotation.y = q[1]
        t.transform.rotation.z = q[2]
```

```
        t.transform.rotation.w = q[3]
        self.br.sendTransform(t)

        rate.sleep()

if __name__ == '__main__':
    try:
        node = FramePublisher()
    except rospy.ROSInterruptException:
        pass
```