

---

# M.R.F.A.M.E.

Mars Robot For Autonomous Mapping and  
Exploration

---



ROB 10  
Group 1053

Robotics  
Aalborg University



# AALBORG UNIVERSITY

## STUDENT REPORT

### Robotics

Aalborg University  
Department of Electronic Systems  
Fredrik Bajers Vej 7B  
DK-9220 Aalborg  
<http://www.robotics.aau.dk>

**Title:**

M.R.F.A.M.E

**Theme:**

Mobile Robots

**Project Period:**

Spring 2023

**Project Group:**

ROB 1053

**Participant(s):**

Alexandru Mihai Smau  
Bogdan Stefan Miron

**Supervisor(s):**

Simon Bøgh

**Copies:** 0

**Page Count:** 63

**Date of Completion:**

June 2, 2023

**Abstract:**

This master thesis explores the uses of Visual Simultaneous Localization and Mapping (VSLAM) techniques deployed on the Martian and Lunar terrains simulated in Isaac Sim Omniverse. The physical setup consists of a rover based on ExoMy Rover developed by ESA, and a RealSense D435i camera. The research begins by conducting a review of the state-of the art and then focuses on a comparison between RTAB-Map, a state-of-the-art SLAM algorithm and its competitor, developed by NVIDIA, Elbrus.

The solution is tested using a simulated Lunar terrain provided by the University of Luxembourg, and a simulated warehouse provided by NVIDIA. The results discuss the applicability of the algorithms and enters a discussion about what aspects of the presented algorithms functioned well and where they lacked in terms of performance.

# Preface

Group ROB1053, Aalborg University, June 2, 2023

---

Alexandru Mihai Smau  
<asmau21@student.aau.dk>

---

Bogdan-Stefan Miron  
<bmiron21@student.aau.dk>





# Acronyms and abbreviations

Acronym	Definition
AAU	Aalborg University
BRIEF	Binary Robust Independent Elementary Features
BT	Behaviour Tree
CAD	Computer Aided Design
CAN	Controller Area Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DDS	Data Distribution Services
EDL	Entry-Descent-Landing
EOL	End of Life
ERC	European Rover Challenge
ESA	European Space Agency
F2F	Frame To-Frame
F2M	Frame To-Map
GFTT	Good Features to Track
GPS	Global Positioning System
GPU	Graphics Processing Unit
GTSAM	Georgia Tech Smoothing and Mapping
IMU	Inertial Measurement Unit
ISRU	In-Situ Resource Utilization
LTM	Long-Term Memory
MAV	Mars Ascent Vehicle
MFR	Mars Fetch Rover
MSR	Mars Sample Return
NASA	National Aeronautics and Space Administration
NAV2	Navigation2
OBC	On-Board Computer
PC	Personal Computer
QR	Quick Response
RAM	Random Access Memory
RGB-D	Red Green Blue Depth
ROI	Region of Interest
ROS	Robotic Operating System
SLAM	Simultaneous Localization and Mapping
SMS	Short Message Service
TCP	Transmission Control Protocol
TRN	Transcriptional Regulatory Network
UDP	User Datagram Protocol

UE	Unreal Engine
URDF	Unified Robotics Description Format
USD	Universal Scene Description
VIO	Visual Inertial Odometry
VO	Visual Odometry
VRAM	Video Random Access Memory
VSLAM	Visual Simultaneous Localization and Mapping
WM	Working Memory
XML	Extensible Markup Language
iSAM	Incremental Smoothing and Mapping

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Problem Analysis</b>	<b>3</b>
2.1	Communication with Earth . . . . .	4
2.2	Entry, Descent and Landing . . . . .	5
2.3	Martian Terrain and Climate . . . . .	6
2.4	Correct Localization of the Robot . . . . .	6
<b>3</b>	<b>Technical analysis</b>	<b>8</b>
3.1	SLAM . . . . .	8
3.2	Comparison Between Different SLAM Approaches . . . . .	9
3.3	QR Landmark Recognition . . . . .	16
3.4	Robot Operating System . . . . .	17
3.5	NVIDIA Isaac Sim . . . . .	18
3.6	Navigation2 . . . . .	19
<b>4</b>	<b>Implementation</b>	<b>22</b>
4.1	Rover Description . . . . .	22
4.2	Simulation Environment . . . . .	28
4.3	Software and Hardware Setup . . . . .	29
4.4	Detection Stack . . . . .	33
4.5	SLAM Stack . . . . .	36
<b>5</b>	<b>Testing</b>	<b>39</b>
5.1	Requirements . . . . .	39
5.2	Delimitation . . . . .	41
5.3	Tests . . . . .	42
<b>6</b>	<b>Results</b>	<b>48</b>
6.1	Simulation Environment Results . . . . .	48
6.2	Perception and Control Results . . . . .	48
<b>7</b>	<b>Discussion and Conclusion</b>	<b>50</b>
7.1	Discussion . . . . .	50
7.2	Conclusion . . . . .	52

<b>Bibliography</b>	<b>54</b>
.1 Isaac Sim Setup . . . . .	59
.2 Testing . . . . .	61

# 1 - Introduction

Mobile robots are capable of replacing people in many fields. Applications include surveillance, planetary exploration, emergency rescue operations, reconnaissance, intervention in extreme environments, medical care, as well as many other industrial and nonindustrial applications [1].

Space exploration, here defined as the physical exploration of the outer space, fully begun in 1957 with the successful launch of Sputnik 1, the first man-made satellite launched into orbit. After that, both the United States of America (USA) and the Soviet Union have used a combination of orbiters and impactors to study the properties and composition of the Moon and various other celestial bodies. However, both approaches had their limitations. The former could only offer low resolution data over a large area, while the latter could offer high precision data but, limited to the impacted site.

To explore a larger area on distant celestial bodies, the use of either a manned or robotic mission was required. While a human can perform very complex tasks and experiments, the resources and technological requirements needed for their safety makes this approach challenging, both from an engineering and a financial perspective. It is for this reason that robotic rovers are preferred.

The first rovers (starting with the Soviet Union's Lunokhod 1), were teleoperated. This is permitted for nearby celestial bodies as the latency is within acceptable safety margins. A radio communication between Earth and the Moon is around 2.6 seconds. However, on Mars the latency is between 3 and 22 minutes [2]. This lag would force operators on Earth to either issue only small movement commands or risk the rover getting stuck in an unseen obstacle such as sand dunes. Both options incur a time cost for the mission which may not be acceptable. This issue raises the need for an autonomous, vision based system that can localise, map and navigate in dangerous terrains.

The goal of this project is to create an autonomous rover for planetary space exploration based on the ExoMy Mars rover technology from the European Space Agency. To complete the exploration stack, the rover must have a variety of actuators, sensors, and software as Stachniss et al said : *"Exploration is the task of guiding a vehicle in such a way that it covers the environment with its sensor"* [3]. The extraction

of data of the explored area will provide more answers to the questions that the humanity has.

**Initial Problem Formulation**

*How can a mobile robot use its sensors to map and subsequently navigate its environment?*

## 2 - Problem Analysis

There are currently only two celestial bodies onto which unmanned rovers have been deployed: the Moon and Mars. The missions for these rovers have historically been a combination of surveying the surface in order to provide high quality images of it and carrying out measurements using its probes to determine the physical characteristics of its environment.

A shift from this paradigm occurred with NASA's Mars 2020 mission, where the Perseverance rover collected soil samples and left the sealed tubes in designated caching areas for them to be collected and returned to Earth by a subsequent mission. The reason for this approach is that it is unfeasible to equip the rover with the equipment necessary to perform a detailed biological analysis of the samples. Such an attempt would incur adding extra weight and power requirements. Moreover, there is a high chance that the equipment itself could be damaged during the transit between Earth and Mars.

NASA in collaboration with ESA have thus created the Mars Sample Return (MSR) mission which aims to bring the samples back to Earth. This consists of a Mars Fetch Rover (MFR) and a Mars Ascent Vehicle (MAV). The former has the task of gathering all the samples from the caches left by Perseverance and load them onto the MAV.

Apart from official missions announced by NASA and ESA, the Defense Advanced Research Projects Agency (DARPA) has shown interest in developing robots capable of exploring underground lava tunnels on Earth and, potentially, with applications for Mars. In 2021 NASA participated in DARPA's Subterranean Challenge thus implying that research is being done for this use case [4].

Lastly, NASA's Artemis mission has the final goal of sending humans to the Moon again. To achieve this, In-Situ Resource Utilization (ISRU) will have to be used. This lends itself perfectly to robotic automation where repetitive and potentially dangerous tasks can be delegated to a robot such as regolith collection and transport to a processing plant.

The main problems that were encountered with Mars rovers were: communication with the base on Earth, landing of the robot, terrain navigation and the correct

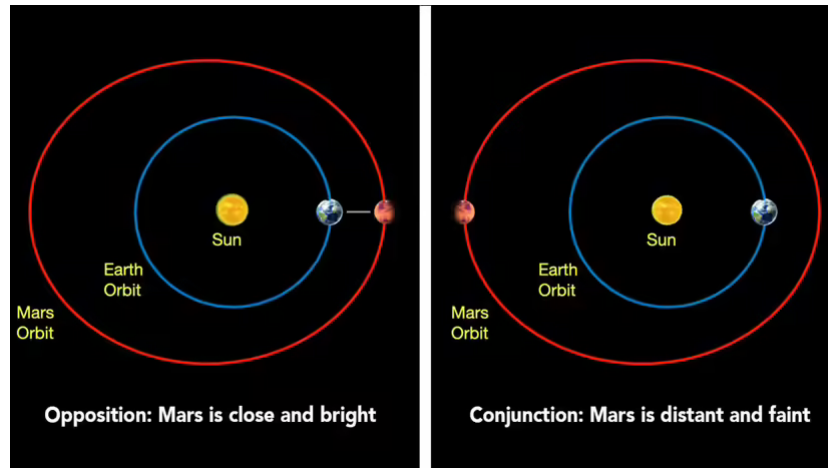


Figure 2.1: Presentation of a Solar Conjunction between Mars and Earth [6]

localization of the robot. These problems will be discussed further in the following chapters and given solutions on how to solve them. Once each problem gets a solution the exploration of the extraterrestrial environment will encounter fewer difficulties.

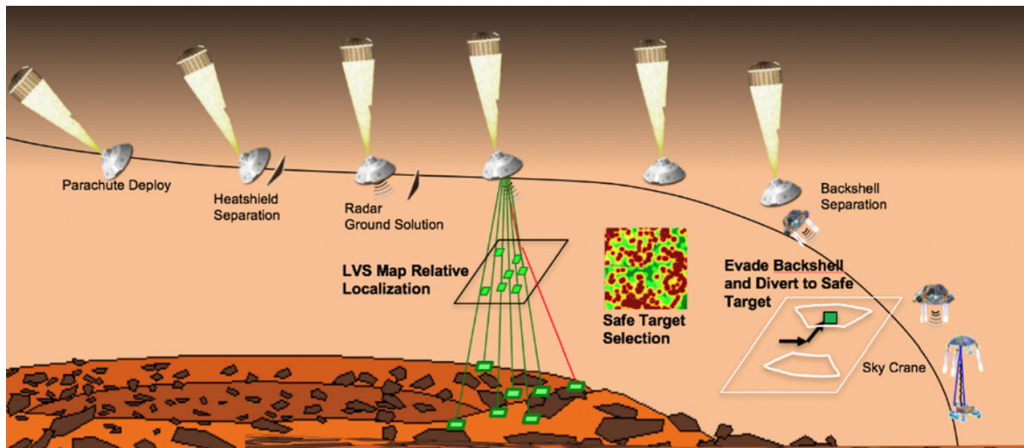
## 2.1 Communication with Earth

Over long distances, such as Mars - Earth, the teleoperations between a rover and a human can have a delay between 6 and 20 minutes. Such delays have a significant effect on real time operations. A way to counter this problem is to make the vehicle autonomous. [2].

The radio link between the base on Earth and the rover on Mars can be impeded by solar conjunction. Solar conjunction is the time where the Sun is between the two planets and expels hot, ionized gas from its corona. This gas combined with the Sun's strong electromagnetic radiation jams the radio signal for two weeks. This phenomena occurs once every two years. A representation of this can be seen in Figure 2.1.

During that period of two weeks, the rover is not receiving any command from the engineers and it stays on stand by. NASA engineers are preparing for the next conjunction, 11 November - 25 November 2023, a method to make the rover still collect data and continue its mission by by automating some of its tasks [5].





**Figure 2.2:** Presentation of different stages which the rover has during the landing on Mars using the TRN method [7]

## 2.2 Entry, Descent and Landing

According to the NASA's data, only 40 percent of the missions from any space agency are successful. A big problem being the landing phase, or how NASA named it, entry-descent-landing (EDL) phase. This phase consists of making sure the rover enters in Mars' atmosphere and lands without any issues on its terrain.

To land on the Mars surface the rover would need to enter into the martian atmosphere, descent at a controlled speed to a chosen safe spot, and then land on the chosen spot. This is done by an aeroshell which penetrates the upper atmosphere. At 10km altitude, parachutes deploy denoting the descent stage, reducing the velocity of the rover and prepare it for the landing phase. Twenty seconds after the parachutes are deployed, the detection hardware becomes operational and analyzes the terrain. An illustration of the EDL phase can be observed in Figure 2.2 [7].

In 2020, the rover named Perseverance used a new method named Terrain-Relative-Navigation (TRN). The Perseverance rover uses this method to gauge its location as it descends into the Martian atmosphere with its parachute. As a result, the rover can locate itself in relation to the ground with an accuracy of at least 40 meters. TRN works as follows: during the descent the rover takes pictures of the terrain and compares it to the orbital map which is already stored in the memory. After some fast comparisons of different landmarks between pictures and the orbital map the rover chooses the safest zone where it can reach [7].

## 2.3 Martian Terrain and Climate

The Martian environment imposes geological challenges such as slopes, cliffs, sand and rocks of different sizes. These environmental factors makes it difficult to land and traverse the terrain. Also, since the surface of Mars is covered by martian dust and the climate of this planet it is prone to sand storms, the work environment for the rover will be challenging [8]. There are also wind gusts and the dust, being very small and slightly electrostatic, can stick, corrode and even damage the equipment of the robot. Hence the mission can be jeopardized [9].

Dust storms are not so frequent. The occurrence of a big sandstorm is approximately once every three Mars years, equivalent of 5 and a half Earth years [9]. Hence, during that time the rover cannot do its tasks and the chances of losing equipment, data or even the rover itself are high.

## 2.4 Correct Localization of the Robot

Quality of the localization knowledge has a direct impact on how rover activities are planned and refined into commands for execution on the vehicle.

In order to preform task correctly, the rover's control system needs know the location and the orientation of the vehicle precisely. The rovers cannot use GPS so they are equipped with IMU and encoders to estimate the location. For example, Sojuren rover, during Mars Pathfinder mission used wheel and heading sensors [10]. Since the terrain of Mars is mainly described by rocks, slopes, and canyons full of dust, the encoders used by Sojuren were not sufficient for the mission. Due to this type of environment, the rover's wheels can get easily stuck in sand, dust or any kind of obstacles and provide erroneous information of it's position.

Usually mars rovers are also equipped with a panoramic camera which is connected to the control system. This camera is used to find the location of the Sun during the cycle of one martian day, named sol, leading to a much more refined determination of the rover's attitude. To further improve localisation, a stereo camera was also added to create 3D XYZ maps of the readily discernible nearby topography. In order to direct the scientific instruments to the points of interest with a high precision, the 3D keypoints obtained from stereo vision were used. Vision based navigation was proven to be reliable method of reducing the localisation error which otherwise would increase over time. [10].

Two mitigating options are available in this situation: either the instrument can be articulated in a restraining manner to increase its effective field of view and correct for

localization error, or a new stereo imagery can be acquired from the new position, sent back to Earth, and wait until the next sol to estimate a new accurate location. Both of this options require more usage of bandwidth and also it may be a problem communication with Earth as stated in Chapter 2.4 [10].

### **Final Problem Formulation**

Now, that it has been given a proper understanding of the major challenges that can happen during an extraterrestrial mission, the scope of the project will be narrowed to a final problem formulation.

*How can a mobile robot's navigation be used in spatial exploration to provide efficient data of the explored area such that can be autonomous and provide a precise localisation?*

## 3 - Technical analysis

This chapter will present different methods and techniques in order to answer the question at the end of Chapter 2.4. The scope of the project has been narrowed such that the rover can have a better understanding of its surroundings, for example 3D mapping, be autonomous and provide an accurate localisation of its position in real-time.

### 3.1 SLAM

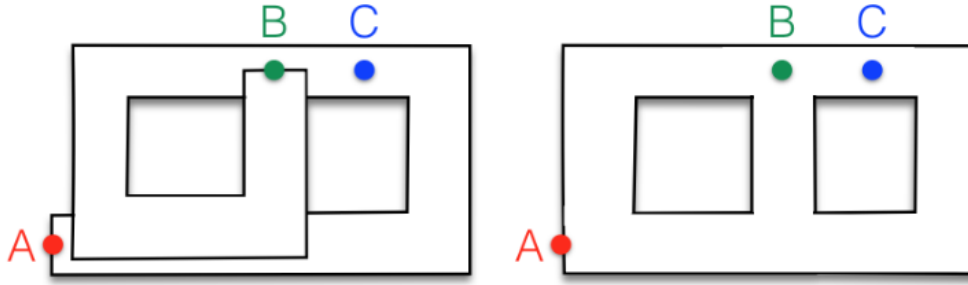
The paper presented by Cesar Cadena et al., titled *Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age* [11], follows to answer multiple questions regarding SLAM, but one of them is: Do robots need SLAM?

To answer the question, a definition of SLAM is needed in order to understand its use. SLAM represents the ability of a robot or an autonomous vehicle to localize itself and create a map of its surroundings.

The map is essential in order to support other tasks, such as path planning or provide the human operator a visualization of the robot. Also, the map provides a better understanding of the state of the robot. Scenarios where the robot uses GPS makes the utilization of SLAM redundant. Hence, the popularity of SLAM algorithms came within the absence of GPS [11].

Without a map, dead-reckoning would invariably drift over time. However, with a map containing a set of recognisable landmarks, the robot can "reset" its localization error by returning to previously visited locations (a process known as loop-closure).

When performing odometry, a robot views the world as an "infinite corridor" while ignoring loop closures. The robot is informed via a loop-closure event that this "corridor" keeps crossing itself, see Figure 3.1. The benefit of loop-closure now becomes apparent. By locating loop-closures, the robot is able to comprehend the true topology of its environment and discover shortcuts between locations [11].



**Figure 3.1:** Left: map built from odometry. The map is homotopic to a long corridor that goes from the starting position A to the final position B. Points that are close in reality (e.g., B and C) may be arbitrarily far in the odometric map. Right: map build from SLAM. By leveraging loop closures, SLAM estimates the actual topology of the environment, and “discovers” shortcuts in the map. [11]

Hence, the author concluded that SLAM is indeed needed for autonomous robots lacking external localization service, such as GPS. The versatility of SLAM with different tasks, from cleaning robots to self-driving cars and even military exploration robots make it a topic worthy of research.

## 3.2 Comparison Between Different SLAM Approaches

### 3.2.1 RTAB-Map vs ORB-SLAM3 vs OpenVSLAM

Merzlyakov et al [12] presented in their paper a comparison between several SLAM algorithms on 3 datasets: KITTI, EuRoC MAV, TUM RGB-D. The difference between those 3 can be seen in Figure 3.3. The algorithms which are proposed to analyze can be seen in Figure 3.2. In order to compare the algorithms, a baseline requirement is needed for further selection. This baseline consists of: loop-closures, re-localization, pure localization support, RGB-D and Stereo cameras.

Those criteria are set in order to make sure that the compared solution can map a space in a way that is globally consistent and can localise within this map over the course of several days, weeks, or even years [12]. Based on these arguments, the authors selected three algorithms for the comparison: ORB-SLAM3, OpenVSLAM, and RTAB-Map.

The setup used for their testing consists of a PC which uses an Ubuntu 18.04 OS and an Intel Core i5-6600 4-core CPU operating at 3.30GHz with 8 GB of RAM memory [12].

The algorithms are tested on the three datasets in the modes described below:

VSLAM	Type	Mono	Stereo	RGB-D	Fisheye	IMU	Pure Localization	Re-Localization	Loop Closing	Dev.status
ORB-SLAM3 [2]	SLAM	✓	✓	✓	✓	✓	✓	✓	✓	Sep 2020
OpenVSLAM [3]	SLAM	✓	✓	✓	✓	-	✓	✓	✓	Active <sup>1</sup>
RTABMap [4]	SLAM	-	✓	✓	✓	-	✓	✓	✓	Active
Kimera [6]	VIO/SLAM	✓ <sup>2</sup>	✓ <sup>2</sup>	-	-	✓	-	-	✓	Active
VINS-Fusion [7], [8]	VIO	✓ <sup>2</sup>	✓	-	✓	✓	✓	✓	✓	Oct 2019
SVO [9], [10]	VO	✓	✓	-	✓	-	-	✓	-	May 2017
DSO [11]	VO	✓	✓	-	✓	-	-	-	-	Dec 2018
LSD-SLAM [12]	SLAM	✓	✓	-	-	-	-	✓	✓	Dec 2014
PL-SLAM [13]	SLAM	-	✓	-	-	-	-	-	✓	Nov 2018
OKVIS [14]	VIO	✓ <sup>2</sup>	✓ <sup>2</sup>	-	-	✓	-	-	-	Jul 2016

Figure 3.2: Different SLAM Algorithms [12]

Dataset	Sensors			GT	Environment			Platform
	Stereo	RGB-D	IMU		Small Indoor	Large Indoor	Outdoor	
EuRoC MAV	✓		✓	✓	✓	✓		Drone
TUM RGB-D		✓		✓	✓	✓		Robot, Hand
KITTI	✓			✓ <sup>1</sup>			✓	Car

Figure 3.3: Different datasets used to compare SLAM Algorithms [12]

- EuRoC: ORB-SLAM3 w/monocular, stereo,monocular-inertial, and stereo-inertial; OpenVSLAM w/ monocular, stereo; RTAB-Map w/ stereo.
- TUM RGB-D: All methods w/ RGB-D.
- KITTI: ORB-SLAM3 w/ monocular and stereo; Open-VSLAM w/ monocular and stereo; RTAB-Map w/ stereo. [12]

It has been found out, after testing the SLAM algorithms on these 3 datasets that:

- Indoor environments with stereo cameras **OpenVSLAM** and **ORB-SLAM3** with inertial fusion performed the best.
- Indoor environments with RGB-D cameras **OpenVSLAM** and **RTAB-Map** performed well, even in low-feature conditions.
- Outdoors environment **OpenVSLAM** had the highest robustness and accuracy.

### 3.2.2 Kimera: Visual Inertial Odometry and Mesher

Kimera [13] is an open source library for real-time metric-semantic SLAM. Its inputs are a mono/stereo camera feed and the data stream of an IMU. The algorithm is able to output a local 3D pose estimation and create a 3D mesh of its surrounding environment. The architecture comprises of 4 modules, each having their own thread: Kimera-VIO, Kimera-Mesher, Kimera-Semantics and Kimera-PGMO (Pose Graph and Mesh Optimisation) .

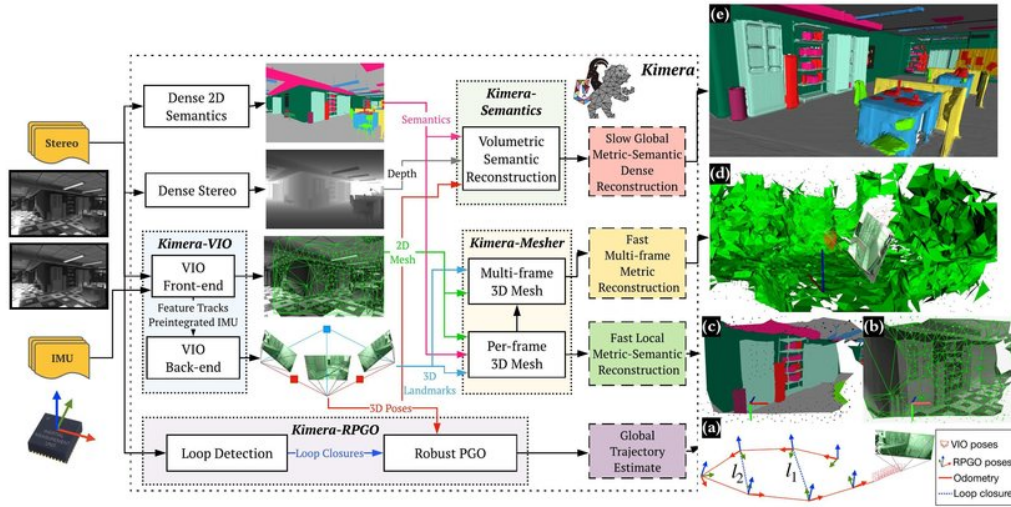


Figure 3.4: Kimera architecture [13]

Kimera-VIO is used for accurate, IMU-rate pose (state) estimation. This module is itself split into two threads. A front-end thread that integrates the IMU measurements and performs the vision algorithm. This algorithm detects Shi-Tomasi corners [14] and tracks them across the frames using a Lucas-Kanade tracker [15]. These corners are also matched in the stereo image pair using a 3-point RANSAC [16]. From the corner pairs detected in the stereo pair, the relative position of the camera can be inferred. From the movement of these corners between each frame pair, the velocity in 3D of the camera can thus be calculated. The back-end thread performs a fixed-lag smoother which combines the velocity from the IMU and the velocity from the vision component to obtain a pose estimate. This step is based on GTSAM (Georgia Tech Smoothing and Mapping) [17] and uses iSAM2 (Incremental Smoothing and Mapping). This is a variation of landmark based SLAM in which the Shi-Tomasi corners are used as the landmarks.

The Kimera-Mesher thread is tasked with creating 3D meshes of the environment based on the data from the Kimera-VIO. It creates a local, per-frame 3D mesh which is fast to compute and is used for obstacle avoidance, and a multi-frame 3D mesh of the global environment (Figure 3.4.d). The local mesh is created from the Shi-Tomasi corners detected by the VIO front-end (Figure 3.4.b).

Kimera-Semantics has two tasks. The first is to optimise the global 3D mesh using RGB-D information from the stereo camera. This is done using the Voxblox library developed by Oleynikova et. al. [18]. The second task is to semantically annotate the global mesh based on the 2D semantic labels obtained from the images. The

2D semantic labels can be obtained with classic deep neural network algorithms. The authors of Kimera have used Mask-RCNN [19].

Lastly, the Kimera-PGMO is tasked with detecting loop-closures and then updating its position and optimising the global mesh. A loop-closure occurs when the robot sees previously detected landmarks or assesses that it is a location it has been before. At this time, it is possible to reduce the accumulated error at this location and reduce the error of each previous known location in the loop.

### 3.2.3 Elbrus Stereo Visual SLAM

Elbrus or Nvidia VSLAM [20] is a method created by Nvidia for estimating the position of a robot, by two core technologies. The two technologies are: Visual Odometry (VO) and Simultaneous Mapping and Localization (SLAM). These two approaches are useful in environments where GPS can not be used or is noisy, such as indoors or spatial surroundings.

#### Architecture of Elbrus

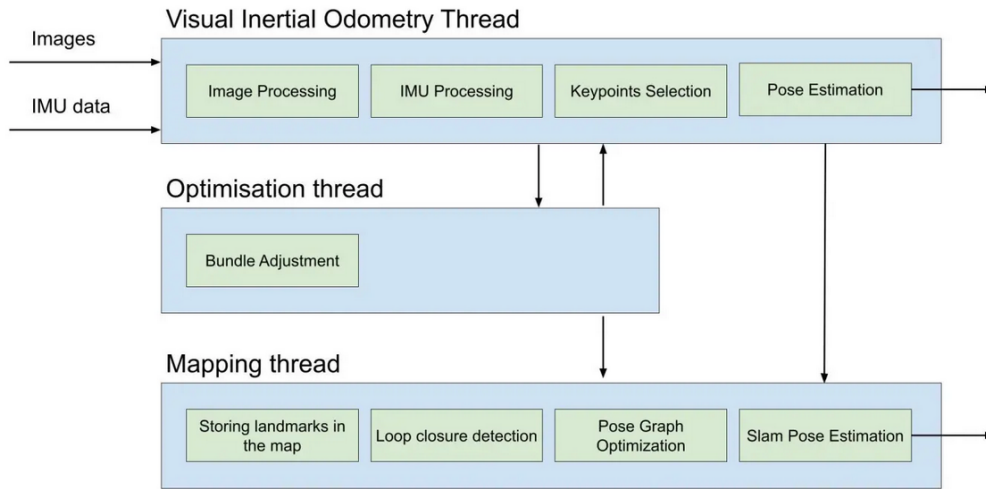
The architecture of Nvidia VSLAM [20] is operated by two modules, working as a two-tiered system, VIO and SLAM. Its architecture can be seen in Figure 3.5.

The VIO is a technique for determining a camera's position in relation to its starting position. This method is iterative in that it evaluates two sequential input frames (stereo pairs) at each iteration. It identifies a group of keypoints on both frames. The ability to predict the transition and relative rotation of the camera between frames comes from matching the keypoints in these two sets.

The method of SLAM is based on the VIO predictions. By utilising the knowledge of previously observed portions of a trajectory, it seeks to enhance the accuracy of VIO estimations. When the current scene has been seen before, the camera moves in a loop, which is detected, and an additional optimisation technique is done to fine-tune previously obtained positions. Those actions resulting in improving the accuracy uses up extra resources in the process. The results can be seen in Figure 3.6a. SLAM use is an option that can be turned off.

Elbrus [20] can use IMU measurements in addition to visual data. When the VO is unable to estimate a position, for as when there is poor lighting or a long solid surface in front of a camera, it immediately switches to IMU. In circumstances of poor visual conditions, using an IMU usually results in a significant performance improvement. In order for the IMU to work in concordance with the VO, the robot's sensor has to be calibrated first in a plane, horizontal surface. This





**Figure 3.5:** Elbrus' architecture. The process of communication between VIO thread and SLAM thread. The data is gathered by cameras and IMU, corrected by the optimisation thread and enhanced by the SLAM thread. Bundle adjustment is part of the optimisation thread and is in charge of refining a visual reconstruction to produce jointly optimal structure and viewing parameter estimates. [20]

requirement can be a disadvantage for in an environment such as Mars or Moon, since mainly the terrain there is composed by bumps, craters and steep plains. With VGA resolution, Elbrus [20] offers real-time tracking performance of more than 60 FPS. The algorithm achieves a **localization drift of 1% and an orientation inaccuracy of 0.003 degrees/meter of motion for the KITTI benchmark**. This can be seen in the Figure 3.6b .

### 3.2.4 RTAB\_MAP

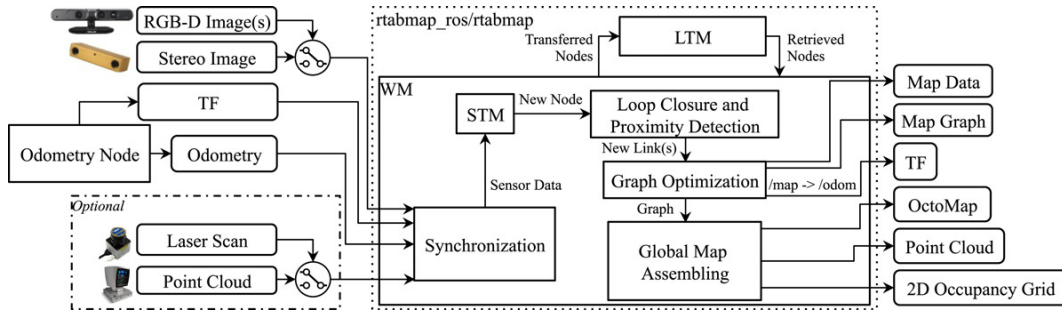
RTAB-Map is an open source SLAM library created by Mathieu Labbé et. al [RTAB-Map] which is designed to be deployed in large environments over a long period of time. Originally designed as a standalone C++ library, since 2013 it has been made open-source and provides a ROS wrapper. It is designed to be highly modular, allowing a large combination of sensors: RGB-D, stereo images, laser scan, point cloud and IMU. Moreover, it provides a robust loop-closure algorithm which, when combined with an efficient short-term long-term memory management system, requires fewer hardware resources. This make it ideal for use on limited resource hardware such as embedded platforms.

The main architecture of RTAB-Map is shown in figure 3.7. The required inputs are video feed which can be either stereo or RGB-D, a transformation frame which cor-



(a) The difference between the simple odometry of El-data. The final results being: localization drift of 1% brus and it's SLAM. Showing that the SLAM enhances and an orientation inaccuracy of 0.003 degrees/meter of the output of odometry, being closer to ground truth motion

**Figure 3.6:** Results of the tests for Nvidia SLAM. Subfigure a, differences between odometry and SLAM. Subfigure b, the test results on the KITTI data. Red path being the ground truth and the blue path is the real-time trajectory reconstruction from the stereo camera. [20]



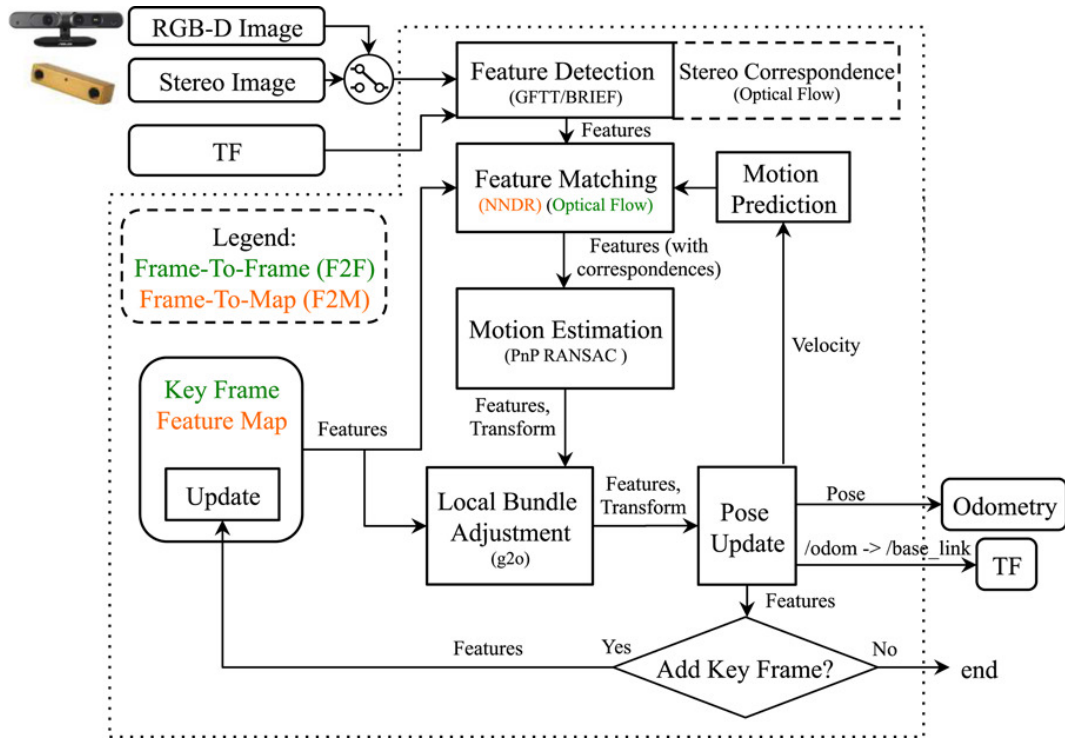
**Figure 3.7:** The RTAB-MAP main node architecture. The required inputs are a video feed (stereo or RGB-D) and a transformation frame (TF) that describes the location of the sensors with respect to the base of the robot. The standard output is Map Data which contains the built map [RTAB-Map].

relates the sensor's position with respect to the robot's base and lastly and odometry source. Additional sensors can be incorporated such as laser scanners, point clouds and IMU (not shown in Figure 3.7, but described in the ROS implementation launch files). The outputs are a corrected odometry topic, the generated Map Data containing a Map Graph with a compressed subset of the sensor data and a Map Graph that is devoid of sensor data. Optionally a 3D (OctoMap) occupancy grid, a dense 3D Point Cloud and a 2D occupancy grid can also be generated.

Similar to Kimera (Chapter 3.2.2), the SLAM approach is that of creating a graph representing the sequential robot poses and their related sensor information and then optimise this chain if a loop-closure is detected (Mathieu Labbé et. al. [21]). However, over a large environment such as outdoors, the graph would grow to such an extent that it cannot be easily processed with limited hardware. To mit-

igate this, RTAB-Map uses a memory management system that has a working memory (WM) and a long-term memory (LTM). The WM holds all the nodes that are most relevant to the robot's current position. WM can also be interpreted as a local "map". Once the robot has moved to a further position and certain nodes are no longer relevant, they are stored in the LTM. This allows the localisation and loop closure algorithm to only use the nodes that are strictly necessary, thus improving efficiency.

The odometry provided by RTAB-Map can be as simple as using the wheel encoders and IMU or it can use more advanced methods such as Lidar or vision. The visual odometry provided ([Scaramuzza et. al [22]) has two approaches: Frame-To-Frame (F2F) and Frame-To-Map (F2M). The first compares consecutive frames and builds a sequence of features from them. This is useful when a new map is built or when a previously made map is expanded. The second approach compares the the new visual frame with one from a database (map). This is useful when solving the "kidnapped robot" problem where the robot is placed in a known environment but unknown initial position.



**Figure 3.8:** RTAB-Map visual odometry diagram for stereo or RGB-D nodes. The F2F and F2M odometry approaches are color-coded in green for the former and red for the latter. [RTAB-Map].

The architecture of the visual odometry can be observed in Figure 3.8. The components of the visual odometry are: feature detection and matching, motion prediction and estimation, local bundle adjustment, pose update and lastly key frame and feature map update. RTAB-Map can use any feature detectors available in the OpenCV library however, the authors have chosen to use the GoodFeaturesToTrack (GFTT) by Shi et. al. [23] as it is more robust over different light intensities and image sizes.

Feature matching differs between F2M and F2F. For the former a descriptor is required in order to match current frame features against those stored in a database. The descriptor used is BRIEF [24] because its fast computation times. Using the descriptors, the matching is then done using nearest neighbour search (Lowe et al. [25]). For F2F, the features are tracked using optical flow.

The features matching across frames allows the algorithm to calculate the direction and velocity. From this, a prediction can be made on where the same feature will be in the subsequent frame. This helps narrow down the feature tracking area improving the efficiency.

The Local Bundle Adjustment and the Pose Update mechanisms in Figure 3.8 have the purpose of refining the calculated motion and outputting the odometry of the cameras. This odometry is then correlated with the transformation frames which, in the end, will output the location of the base of the robot.

### 3.3 QR Landmark Recognition

In the paper presented by P. Kurnar et al [26] it is shown the use case of detecting landmarks, such as QR codes, in the navigation of the mobile robots. The implemented solution consisted in using the class provided by OpenCV for detecting QR codes. The detected QR codes are further tracked by a Kalman Filter. The results provided by them were encouraging but not sufficient.

A similar approach has been realised by H. Zhang et al [27] for indoor navigation. The method used for detecting and tracking the QR codes was done by combining two libraries, OpenCV for image processing and ZBar to decode and track the bar codes. The setup for their experiment consisted on: a mobile robot equipped with a laser and an industrial camera pointed to the ceiling to detect the QR codes.

The height of the ceiling in the experiment done by authors has a measurement of 2.45m. The size of the QR codes also played an important role in detection. QR codes smaller than 0.12x0.12m could not be detected. Results of the test are

Category	ROS 1	ROS 2
Network Transport	Bespoke protocol built on TCP/UDP	Existing standard (DDS), with abstraction supporting addition of others
Network Architecture	Central name server ( <i>roscore</i> )	Peer-to-peer discovery
Platform Support	Linux	Linux, Windows, macOS
Client Libraries	Written independently in each language	Sharing a common underlying C library ( <i>rcl</i> )
Node vs. Process	Single node per process	Multiple nodes per process
Threading Model	Callback queues and handlers	Swappable executor
Node State Management	None	Lifecycle nodes
Embedded Systems	Minimal experimental support ( <i>roserial</i> )	Commercially supported implementation (micro-ROS)
Parameter Access	Auxiliary protocol built on XMLRPC	Implemented using service calls
Parameter Types	Type inferred when assigned	Type declared and enforced

Figure 3.9: Differences between ROS 1 and ROS 2 features [ros2].

promising, resulting in a **mean error of 6cm on X axis**, a **mean error of 6cm in Y axis** and a **maximum error at X-Y of 13cm**.

Implementing the QR codes in the navigation stack of the robot, made it to move with a speed of 2.5m/s while performing SLAM and not losing track of the detection [27].

### 3.4 Robot Operating System

Robot Operating System [28] (ROS) is an open source software platform intended for the development and deployment of robotic solutions. This platform provides the developer with a wide range of libraries and services that allows them to develop, configure and maintain their application in a highly modular fashion. In broad terms, ROS works as a publisher-subscriber system, with the underlying communication protocols hidden from the user.

The project began in 2007 with the introduction of what is now known as ROS 1. The last version of ROS 1 is Noetic and its end-of-life (EOL) is scheduled for 2025. ROS 1, while very capable, lacks features that are required for modern applications and industry in general, such as real-time support and a more robust and reliable network middleware. To address this, the developers decided that instead of adding the necessary patches, it would be more suitable to re-design the entire system from the ground up. This new version is known as ROS 2 [28] and its main

underlying differences are represented in Figure 3.9.

From a user's perspective the following have to be taken into account when changing from ROS 1 to ROS 2. There is no longer a central node (*roscore*), making the system fully distributed. The user has the ability to specify a domain number isolating the topics, publishers and subscribers to only that domain. This could be used for instance if there are several robots, each having the same control software. In this case each robot will have its own domain id, thus preventing the overwriting of topics with the same name. Moreover, launcher files are no longer written as an XML file. They are now written as a Python script allowing far greater flexibility and ease of configuration. Hence, this framework will provide a better communication between the hardware of the robot and the code.

### 3.4.1 Action Client-Server

Actions are a new addition in ROS 2. This provides a new way of communicating between nodes via requests-responses by **actions**. Those actions are goal oriented, asynchronous and the communications between **services** and **clients** can be stopped anytime [28]. As previously stated, actions are divided in two types: services and clients.

Services are represented as a request-response communication from clients and providing back feedback to them. Throughout the process, it controls how the action is carried out and handles state changes. The action server can handle numerous objectives at once and prioritizes them according to their importance [28].

Clients represent the component that transmits action goals and get results and feedback. By getting regular updates from the server, clients may ask for the execution of an action and keep track of its progress and, if needed the respective action can be revoked [28].

Hence, the action server and client in ROS 2 provide a standardized method to manage time-consuming activities with goal-oriented communication and feedback mechanisms, which makes it possible to design robust and scalable systems.

## 3.5 NVIDIA Isaac Sim

NVIDIA Isaac Sim [29] is a robotics simulation environment created by the NVIDIA Corporation. It provides a powerful physics engine, photo-realistic environment and a source of high quality sensor data such as IMU and RGBD camera feeds. It also supports a ROS 1 and ROS 2 interface which facilitates the development of a

robotic control stack agnostic of the sensor's data source. Regarding the control of a virtual robot, Isaac Sim exposes to the user the concept of *Articulation Controller* which can be interfaced via a ROS 2 bridge. This controller can be applied to any articulation (joint link) between two Prims and can be set to either a angular velocity or an absolute position. A Prim (short for "primitive") is any basic object in Isaac Sim such as a wheel, an axle or the body frame.

The concept of Prims comes from the Universal Scene Description (USD) markup language developed by Pixar Studios [30]. This is a standard that has been adopted by most major 3D physics engines such as Unity or Unreal Engine. In fact, Isaac Sim allows its users to import an environment created in Unreal Engine or Unity. This may be useful as Isaac Sim does not have many tools for 3D modelling and manipulation, instead focusing more on the creation of virtual robots.

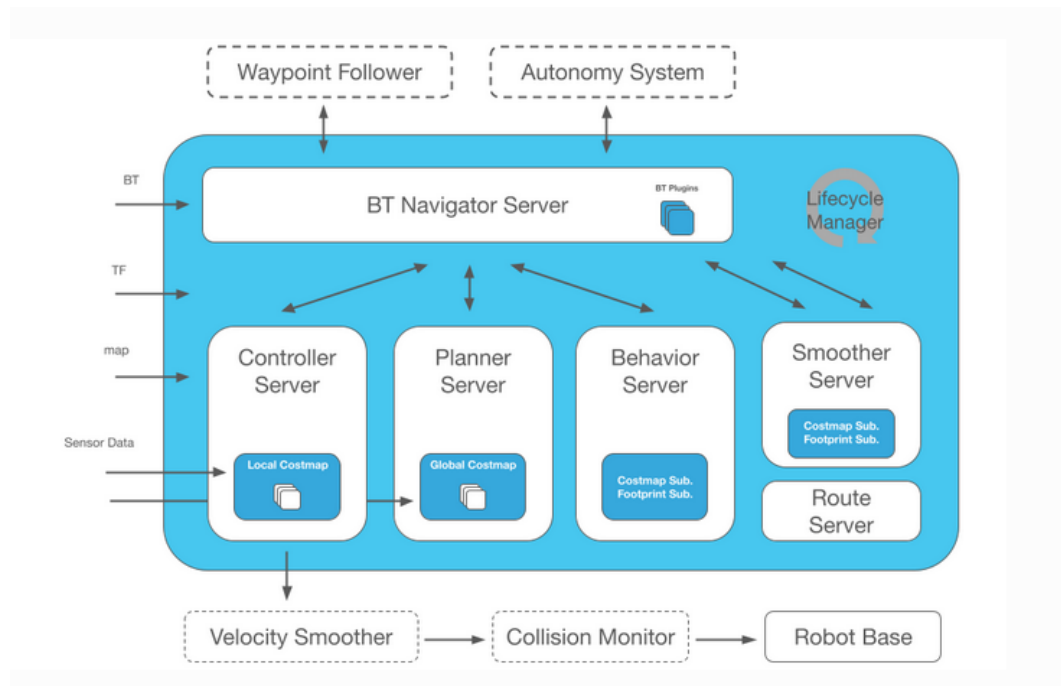
The reason why Isaac Sim is preferable to other robotic simulation environments such as Gazebo [31] is its highly realistic environment generation and quality of sensor data. The real-time rendering of a realistic environment is useful for running computer vision algorithms such as visual odometry and human detection.

## 3.6 Navigation2

Navigation2 or the short form known in the ROS community, Nav2, is the navigation for mobile robots. It is the successor of ROS Navigation Stack. Nav2 is the professionally supported spiritual successor of the ROS Navigation Stack. This project seeks to find a safe way to have a mobile robot to move to complete complex tasks through many types of environments and classes of robot kinematics, such as differential, omnidirectional, Ackermann, legged and custom. Not only can it move from Point A to Point B, but it can have intermediary poses, and represent other types of tasks like object following and more [32].

In the paper presented by S. Manceski [33] et al it is shown, more in depth, the upgrades that ROS Navigation received in order to become Navigation2. The new addition to the Nav2 stack are the behaviour trees which are in charge of path planning, control and recovery tasks. Another feature that makes Nav2 stand out is the flexibility around different types of robots. This stack can be used for any type of mobile robot for a large variety of environments and applications [33].

Navigation2 was built with the idea of being used in an industrial environment. As seen in Figure 3.9, ROS 2 now uses DDS, an upgrade from TCP/UDP and Life Cycle Nodes.



**Figure 3.10:** The design system of the Navigation2 stack [32].

The design of the Nav2 system Figure 3.10, in order to work work requires Behaviour Trees. BT Navigator Server is the highest component and entry level of the server which hosts the BT provided to implement navigation behaviours. BT Navigator Server can communicate with other servers using Action-Client. Planners and controllers are the core of a navigation problem. Recoveries are used to get the robot out of a bad situation or attempt to deal with various forms of issues to make the system fault-tolerant. Smoothers can be used for improving the path planning of the robot [34]. The four action servers of the Nav2 stacks are: controller, planner, behavior and smoother.

The main task of Nav2's planner is to create the most optimal path from a current position to a given goal. The path can also be known as a route, depending on the nomenclature and algorithm selected. Two canonical examples are computing a plan to a goal (e.g. from current position to a goal) or complete coverage (e.g. plan to cover all free space) [34].

Planners can be written to:

- Compute shortest path



- Compute complete coverage path
- Compute paths along sparse or predefined routes [34] [34]

The controller server, or by its former name in ROS1 the local planner, has the task to identify the dynamic obstacles which cannot be seen in the global map.

Controllers can be written to

- Follow a path
- Interact with an object
- Board in a ramp/elevator [34]

The behavior server contains a costmap subscriber to the local costmap, receiving real-time updates from the controller server, to compute its tasks. This is done to obtain information via sensors on how the robot can be reset to a controllable state. In case of not-being controllable, the behavior server may communicate with the operators via different: emails, SMS or messages [34].

Additional path refining is frequently advantageous since the requirements for an ideal path are typically lowered in comparison to reality. In order to do this, smoothers have been developed. These tools are normally in charge of minimizing path raggedness and smoothing abrupt rotations, but they may also be used to increase distance from barriers and expensive locations because they have access to a global environmental representation [34].

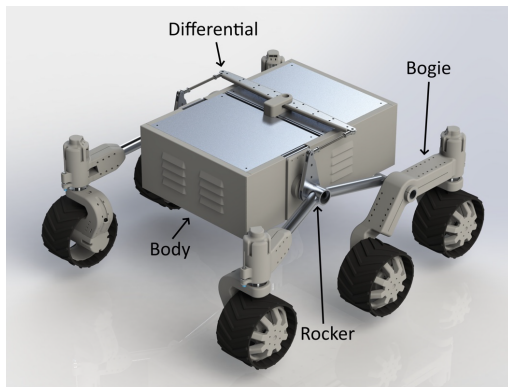
Receiving a path and sending back an enhanced version of it is the general duty of a smoother in Nav2. However, there are several smoothers that may be registered in this server due to the variety of input pathways, improvement criteria, and methodologies available [34].

## 4 - Implementation

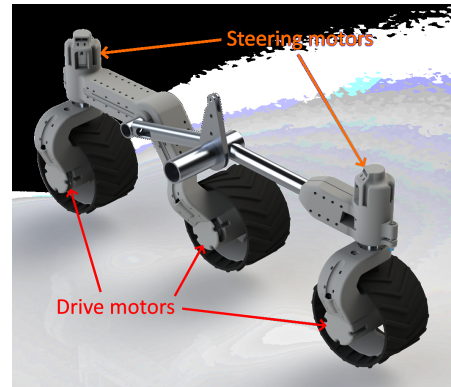
This chapter will present the solution to the question asked in Chapter 2.4 based on the technical analysis preformed in Chapter 3. Hence, a navigation stack, a detection stack and a SLAM stack will be developed such that the robot will be autonomous, provide accurate localisation of itself and provide a better understanding of its surroundings. ROS 2 will be used to support the interaction between the aforementioned stack and the simulated environment.

### 4.1 Rover Description

The rover used for this thesis is inherited from the ROB7 team project at AAU comprised of A. Mortensen et. al. [35]. The model is represented in Figure 4.1a and is based on the ExoMy Rover [36], developed by the European Space Agency (ESA). It has a 6-wheel Ackerman drive however, only the front and rear wheel pair can be steered. the Middle pair's direction is fixed.



(a) Rover model with its main components.



(b) Motor bogie, with 3 drive motors and 2 steering motors on the front and rear.

**Figure 4.1:** Rover model [35].

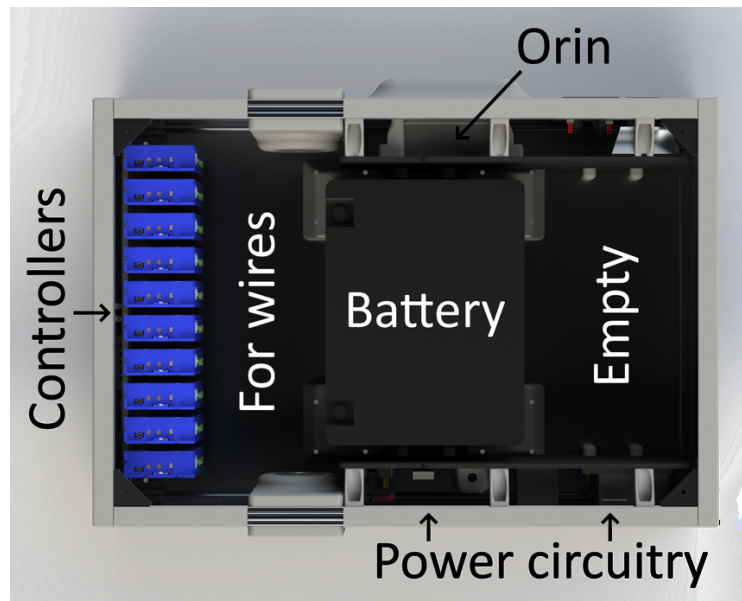


Figure 4.2: Location of components inside the body of the rover [35].

#### 4.1.1 Mechanical Design

The rover is comprised of 6 components: a body, a differential, two rockers and two boogies (Figure 4.1a). The body houses the battery, on-board computer (OBC), motor drivers, as well as the charging and voltage conversion circuitry. The designers wanted the center of mass to be as close to the geometrical center of the robot as possible. To this end, they placed the heaviest component, the battery, as close to the center of the body (Figure 4.2). The rockers are attached to the body by an articulated joint. They each carry half of the body's weight. Onto each rocker there are further two components attached. The first are the two front drives and the second are the rear boogies that support the middle and rear drives. Part of the rocker-boogie suspension is the differential. This component keeps the body pitched parallel to the ground and transfers most of its weight to the rocker-boogie assembly.

#### 4.1.2 Electrical Design

The power source is a 24 V battery, capable of sustaining approximately 5h of continuous drive. There are 10 DB59 motors from Nanotec, each with a 1:62 high-torque planetary gearbox attached. 6 of the motors are used for driving the rover and have been equipped with a Hall-sensor to monitor their angular velocity. The other 4 of the motors are used for steering and they have been equipped with high-resolution encoders to measure their absolute position. The motors and their

sensors are interfaced with the OBC via CAN motor drivers.

The OBC is a 64GB NVIDIA Orin [37], a powerful embedded platform running Ubuntu 20.04. Its Ampere GPU is capable of running computing heavy algorithms usually used for computer vision such as CNNs. Furthermore, the Arm Cortex-A78AE CPU has 12 cores which can be used for branching algorithms such as the ROS 2 navigation and mapping stack.

The sensors propose to be used for this project are Intel RealSense D435i stereo cameras. The technical details of the camera can be seen in the Table 4.1

Intel RealSense Depth Camera D435i Specifications	
<b>Baseline:</b>	50mm
<b>Depth Technology:</b>	Active IR Stereo
<b>Field of View Diagonal:</b>	95°
<b>Field of View Horizontal:</b>	87°
<b>Field of View Vertical:</b>	58°
<b>Horizontal Resolution:</b>	1280 pixels
<b>Vertical Resolution:</b>	800 pixels
<b>Max. Depth Frame Rate at full resolution:</b>	30 fps
<b>Max. Depth Resolution:</b>	1280 x 720
<b>Max. RGB Framerate at full resolution:</b>	60 fps
<b>Max. RGB Resolution:</b>	1920 x 1080
<b>Maximum Range:</b>	0.4m to over 10m
<b>Minimum Depth Distance:</b>	200mm
<b>IMU:</b>	Yes

**Table 4.1:** Hardware specifications of the Intel RealSense Depth Camera D435i [38]

### 4.1.3 Kinematic Model

The kinematic model of the rover is a variation of an Ackerman steering. The variation lies in the fact that the steering is also done with the rear motor pair, not just with the front (like a normal car) as seen in Figure 4.1.3.

To perform a turning maneuver, the rover driver has to control the steering angles for  $W_1, W_2, W_5, W_6$  (Figure 4.3a) and the angular velocities of all the wheels. The constraint is that the turning point must be on the  $W_3, W_4$  axis. The turning angles of each of the wheels will thus be the tangent to the circle formed from the center  $P$  to the wheel. As a consequence to the aforementioned, constraint the the angles

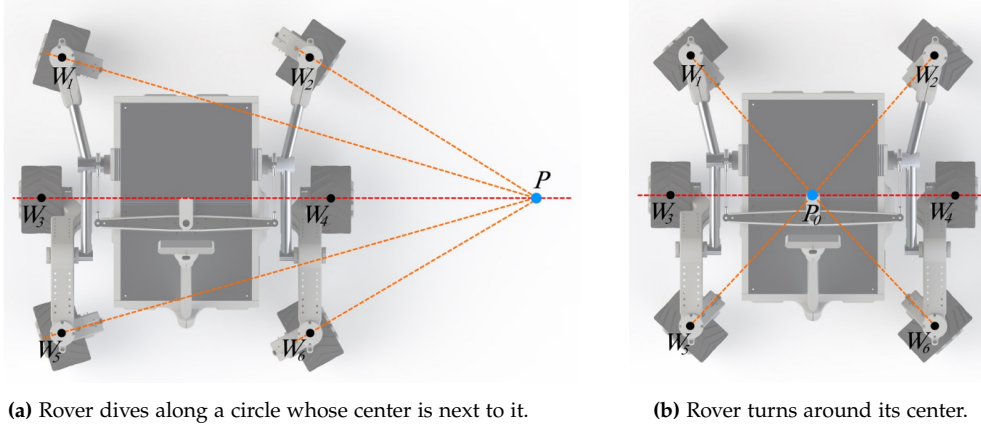


Figure 4.3: Rover turning model described in [35].

for  $W_3$  and  $W_4$  will always be  $0^\circ$ .

The command from ROS 2 will be a linear velocity in the x-axis,  $v_x$  and a angular velocity against the z-axis,  $\theta_z$ . From this, the turning circle's radius has to be calculated. One approach is to calculate how long it takes to travel a quarter of a circle given  $v_x$  ( $\frac{m}{s}$ ) and  $\theta_z$  ( $\frac{rad}{s}$ ) (Equations 4.1 and 4.2).

$$t * \theta_z = \frac{\pi}{2} \quad (4.1)$$

$$t = \frac{\pi}{2 * \theta_z} \quad (4.2)$$

Given that it takes  $t$  seconds to turn a quarter of a circle, how large is the distance based on the  $v_x$  velocity? From the resulting distance, the circle's radius is thus obtained (Equations 4.3 and 4.4).

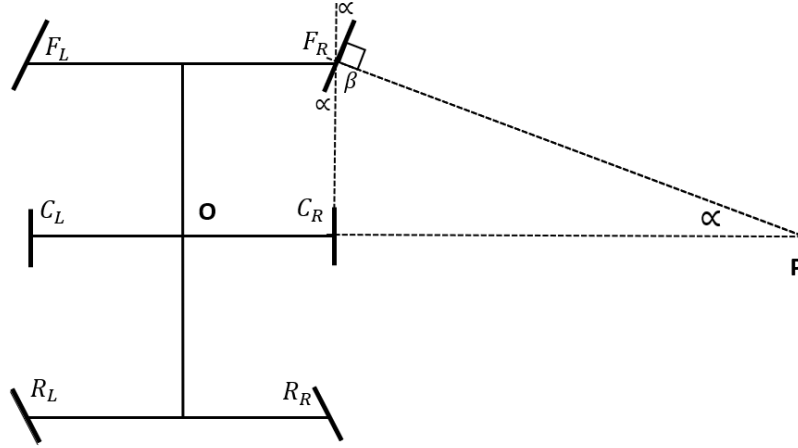
$$d = v_x * t \quad (4.3)$$

$$\frac{\pi}{2} * r = v_x * \frac{\pi}{\theta_z * 2} \quad (4.4)$$

Finally, the turning radius  $r$  (m) is obtained (Equation 4.5).

$$r = \frac{v_x}{\theta_z} \quad (4.5)$$

Once the radius is computed, the steering angles have to be calculated. These can be visualized with a simplified model of the robot: Figure 4.4. In it, the turning radius  $r$  calculated obtained in Equation 4.5 is the distance between the center of the robot  $O$  and the turning point  $P$ . The following algorithm computes the turning angle  $\alpha$  (just above the front right  $F_R$  wheel). The same can be applied for



**Figure 4.4:** Simplified model of the rover's kinematics, focusing only on the front right  $F_R$  wheel's steering angle  $\alpha$ .

the rest.

Observe that the steering angle  $\alpha$  is equal to the  $\widehat{C_R P F_R}$  angle. Thus the angle alpha is calculated in Equations 4.6 and 4.7.

$$\overline{C_R P} = r - \overline{C_R O} \quad (4.6)$$

$$\alpha = \text{atan2}\left(\frac{\overline{C_R F_R}}{\overline{C_R P}}\right) \quad (4.7)$$

The turning radius is with respect to the center of the robot. However, each wheel will have to drive over a smaller or larger circle depending on their offset from the center  $O$ . The radius of the inner and outer circles are given by the distance from the turning point  $P$  and the center of the wheels. In the case of the front right  $F_R$  wheel (Figure 4.4), the radius  $r_{\text{wheel\_turn}}$  is given by Equation 4.8.

$$r_{\text{wheel\_turn}} = \sqrt{\overline{C_R P}^2 + \overline{C_R F_R}^2} \quad (4.8)$$

The last step is to calculate the angular velocity of the wheels. The turn radius for each wheel is computed with Equation 4.8. A quarter of the circle with radius  $r_{\text{wheel\_turn}}$  has to be travelled in  $t$  seconds (Equation 4.2). Thus the wheel angular velocity  $\theta_{\text{wheel}}$  is given by Equation 4.9.

$$\theta_{\text{wheel}} = \frac{\pi * r_{\text{wheel\_turn}}}{2 * t * \text{wheel\_diameter}} \quad (4.9)$$

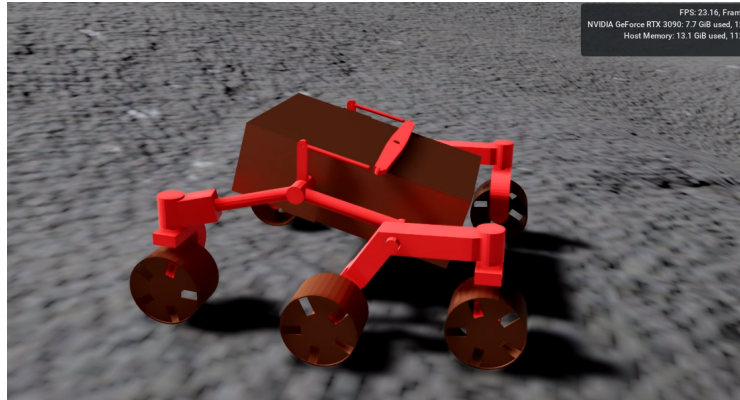


Figure 4.5: The simulated rover rendered in Isaac Sim.

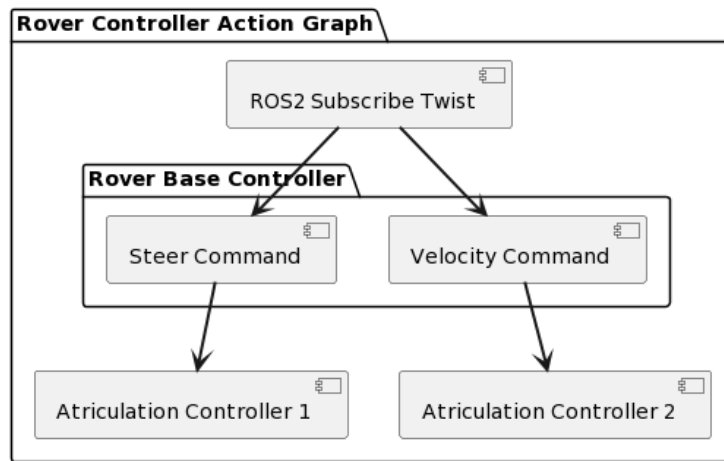


Figure 4.6: Simplified Action Graph showing the conversion from `/cmd_vel` to steering angles and wheel angular velocity.

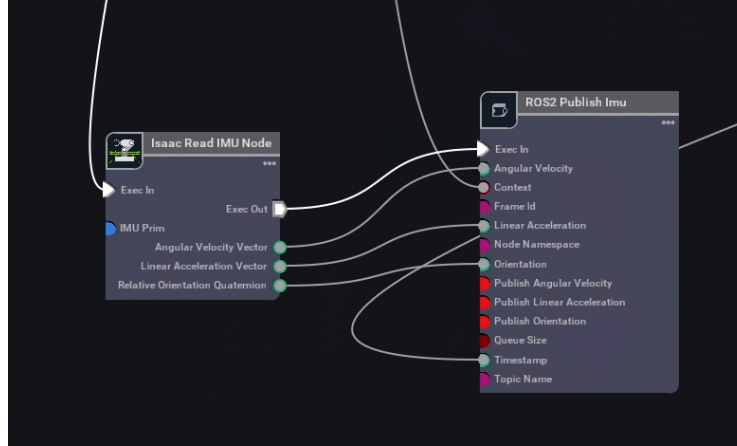
#### 4.1.4 Simulation Model

The initial prototyping and development for the navigation stack was facilitated by the use of the Isaac Sim robotics simulation environment. In it, a URDF model of the rover has been converted to a USD and then added to the environment (Figure 4.5). To this model, two action graphs have subsequently been added. The first is for subscribing to the linear and angular velocity topic in ROS 2 and then converting it to steering angles and wheel velocities. The second is used to publish the stereo cameras RGB and depth images as well as the IMU data, which are then used by the navigation stack.

The Action Graph used for executing the movement commands (Figure 4.6) works by creating a ROS2\_Bridge graph node that subscribes to the `/cmd_vel` topic and



**Figure 4.7:** Simplified Action Graph showing the path Isaac Sim takes to publishing the robot camera feed.



**Figure 4.8:** The Isaac Sim Action Graph node chain required to publish the IMU data.

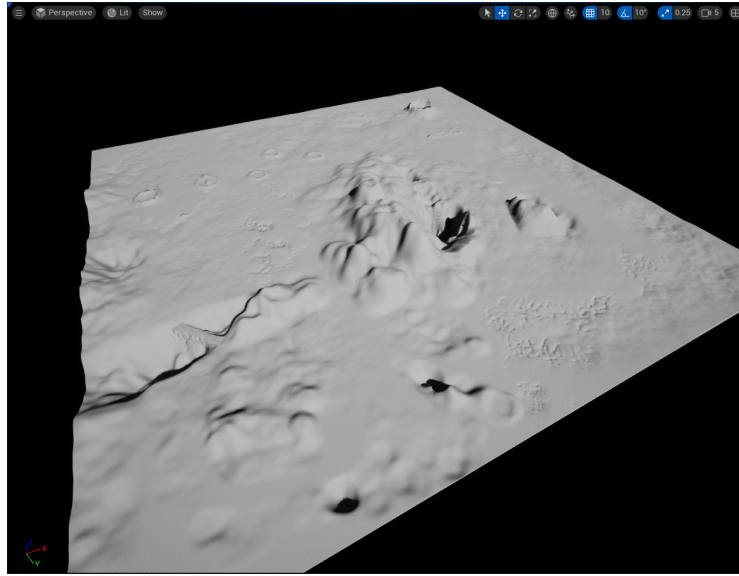
then feeds this information to a *Rover Controller* node. The controller is a custom node that converts the linear and angular velocities to steering angles and wheel velocities using the equations described in Section 4.1.3 and its code can be found in [39]. Lastly the resulting angular commands are sent to two *Articulation Controllers*, one for the 4 steering angles and 6 for the angular velocity of each wheel. A detailed visualisation of this can be found in the Annex .1.1 Figure 5.

The camera and IMU feed is handled by the second Action Graph which subscribes to the appropriate Viewport, encapsulate the images in a ROS 2 message and publishes it (Figures 4.7 and 4.8). A screenshot of the action graph alongside an explanation of its components can be found in Figures 1 and 2 in the Annex .1.1.

## 4.2 Simulation Environment

The simulation environment used for this project is Isaac Sim [29]. The environment provides a realistic physics engine, a wide array of sensor data, a convenient interface to modify and configure the physical properties of the robot and lastly, a ROS2 interface. Combined with the fact that the simulated world surpasses classic robotic simulation environments such as Gazebo makes this piece of software the





**Figure 4.9:** A map which has been created in UE5 order to resemble the terrain during ERC

ideal choice for deploying the navigation stack.

### 4.2.1 Maps

The maps proposed for the simulation represents the spatial environment of Mars. Those were created in Unreal Engine 5 such to be as realistic as possible. As previously stated in Chapter 3.5 the format of the maps is an USD format, created by Pixar in 2016, and now are used in different industries such as: visual effects, architecture, robotics, design and CAD. 16 maps have been provided by the University of Luxembourg and an extra one has been created, inspired based on the map shown on YouTube during ERC 2022. That map created can be seen in Figure 4.9. Since all the maps were created in UE5, the format of the maps are USD. Hence, those maps are compatible with Isaac Omniverse and can be applied various physical properties such as gravity and friction.

## 4.3 Software and Hardware Setup

While versatile, Isaac Sim and the overarching Omniverse ecosystem requires powerful hardware in order for it to run, making it less accessible. As per its requirements page [40] a Good hardware requirement is a CPU (minimum of Intel Core i7 9th Generation or AMD Ryzen 7) with at least 8 cores, 64 GB of RAM, 500 GB SSD and a Ray Tracing capable GPU such as GeForce RTX 3080 with at least 10GB of VRAM. The machine used for this project has the specifications presented in

Table 4.2 and meets the minimum requirements imposed.

Regarding the software infrastructure, Isaac Sim is running on Ubuntu 22.04 inside a Docker container provided by NVIDIA itself. The reason for using the Docker image is that NVIDIA ensures that all its dependencies installed automatically. Moreover, used docker since our machine has Ubuntu 22.04 installed but Isaac Omniverse requires Ubuntu 20.04 for the ROS 2 bridge.

Specs of the PC - RobotLab	
<b>RAM Memory:</b>	128GB
<b>Processor:</b>	AMD Ryzen threadripper 3970x 32-core processor x64
<b>Disk Capacity:</b>	1TB
<b>Operating System Name:</b>	Ubuntu 20.04.6 LTS
<b>Operating System Type:</b>	64-bit
<b>Attached GPUs:</b>	NVIDIA GeForce RTX 3090

Table 4.2: Hardware specifications of the RobotLab PC

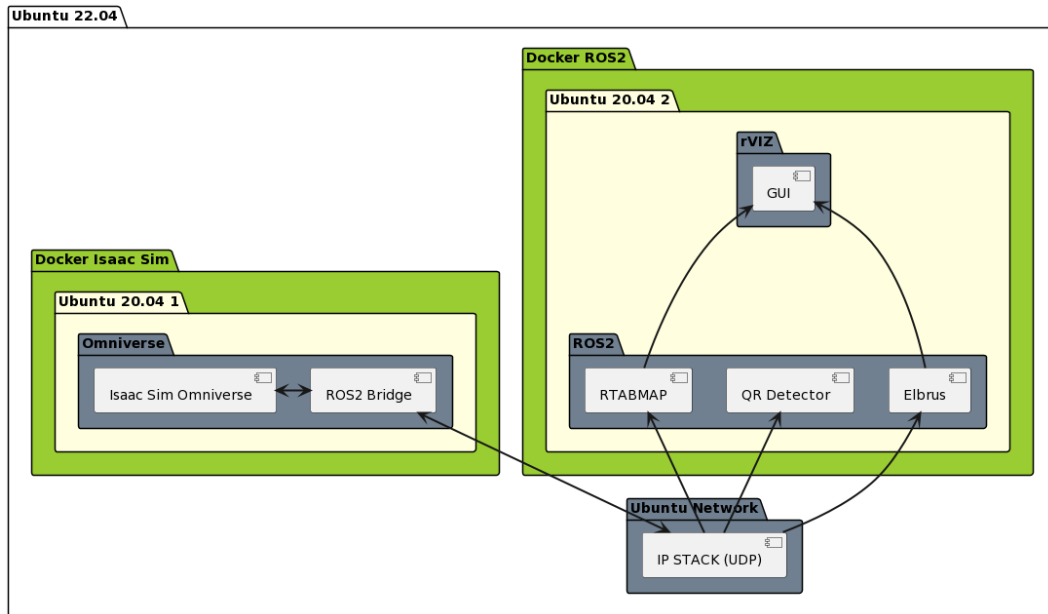
#### 4.3.1 Prim Setup and Robot Control

As mentioned in Section 3.5, the simulated robot is formed of Prims which are connected via links and joints (articulations). The former are used for static connections that do not deform over time, just like a welding. The latter are used to connect two Prims and allow rotational movement akin to a hinge joint. To control the rotation of an articulation joint, Isaac Sim uses the concept of Action Graph. With this, the user can create an *Articulation Controller* node, target a joint, and then set either an angular velocity or an absolute position.

To convert the linear and angular velocity to a wheel angular velocity the user can either employ a *Differential Controller* node or make a custom node if the mobile robot uses a different steering model such as Ackerman. If the rover would have all its wheels fixed in a forward position, then a differential model could be used. However, since the front and rear wheels can be individually steered, an Ackerman model is better suited. More details of the model and its implementation can be found in Section 4.1.4.

#### 4.3.2 ROS 2 Bridge

Omniverse Isaac Sim provides both a ROS 1 and ROS 2 bridge for ROS system integration. For this project, the *ROS2 Bridge: Humble* is used. The motivation is that the Humble distribution is the first long-term-distribution version of ROS



**Figure 4.10:** A presentation of the setup used for the solution proposed

2, having an end-of-life (EOL) set for May 2027. At startup, Isaac Sim defaults to enabling ROS 1 bridge which the user must first deactivate it and then enable the ROS 2 Humble bridge. This must be done before any environment is loaded, otherwise the system will crash when the simulation is started. As the name suggests, the bridge allows a for ROS messages to be sent between Isaac Sim and a ROS instance(s) running in a different terminal even different Docker instances.

The setup for this project consists of two Docker instances running on the same machine, as can be seen in Figure 4.10. The first instance is used for running Isaac Sim while the second is an instance containing the ROS 2 code, the NVIDIA Visual Odometry (Chapter 3.2.3) and RTAB (Chapter 3.2.4). The topics published are the RGBD streams for the stereo camera, IMU data, odometry, simulation time and transformation frames. The subscribed topic is the velocity command. The publishing of Isaac Sim sensor data via the ROS 2 interface is not straightforward and requires a few steps before a message is correctly published. Moreover, there are pitfalls that, without due care, can cause the setup to stop working in an unpredictable manner. The setup for these publishers and subscribers are detailed in the following paragraphs.

To publish the RGB and depth images, a *Viewport* must be created and then set to stream the appropriate feed. In Action Graph, the following node chain must be created in order to publish a video feed: *Isaac Create Viewport* → *Isaac Get Viewport*

*Render Product* → *ROS2 Camera Helper*. The Action Graph can be seen in Appendix .1.1 in Figure 1 and Figure 2. The first two nodes create a Viewport, which usually pops out and then encapsulates the video feed from it. The *ROS2 Camera Helper* node uses this feed and converts it to a ROS *Image* type then publishes it. The topic name is supplied by the user. The major issue observed during the project was that sometimes, when streaming the video feed to Rviz, either the left or the right camera feed would become black. We discovered that the issue was the fact that in order for the Viewports to stream images, it requires for them to be active. In Isaac Sim, in to keep the environment tidy, we decided to put the Viewports as tabs and only highlight the ones we were interested in. Possibly to save computing power (we don't know, we couldn't find this information on any forum), if a Viewport is not active, there will be no rendering and thus no image feed.

Publishing the IMU data is done in a facile manner. The user must first add an IMU sensor to the robot and link it to its main body. Then, in action graph the following two nodes are required: *Isaac Read IMU Node* → *ROS2 Publish Imu* (Figure 4.8). It is important to note that when Isaac Sim is loaded, there are two IMU drivers that are loaded by default. In order to the data to be correctly streamed, the user must deactivate the *DEPRECATED: ISAAC SIM ISAAC SENSOR* (.1.1, Figure 3) plugin before any environment is loaded.

ROS requires a timestamp in order to synchronise the sensor data with the transformation frames and odometry. These two topic are published by Isaac Sim as well. The Action Graph chains required are:...

There is one subscriber in Isaac Sim for the velocity command. This is used to parse the */cmd\_vel* parameters and feed this information to the differential controller. The action Graph used is the following: *ROS 2 Subscribe Twist* → *Differential Controller* → *Articulation Controller* (Appendix .1.1, Figure 4). The robot used to test the navigation will be an *NVIDIA Carter* as the setup for the Mars Rover described in Section 4.1.4 has issues with publishing the transformation frames. However, the Action Graph used for driving M.R.F.A.M.E. is slightly different and will be explained in detail in Section 4.1.4.

Regarding the environment, as stated in Chapter 4.2 this project will use a 3D Moon model obtained from the University of Luxembourg. An example of this can be observed in Figure 4.11. It is in this environment that the navigation stack will be mainly deployed. To test basic functionality the *NVIDIA Warehouse* is also used. It is expected that a degradation in in Visual odometry accuracy on the Moon environment due to significantly less distinct features in the environment.

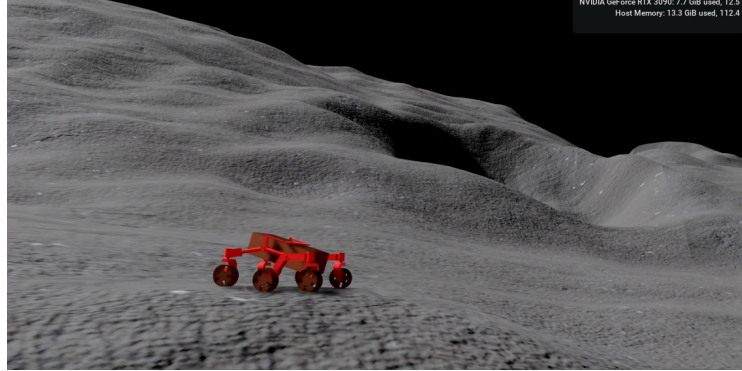


Figure 4.11: Rover deployed in a Moon environment.

## 4.4 Detection Stack

The detection stack is in charge of giving a better understanding of the rover's surroundings using cameras. In Isaac Sim a camera is simulated such to recreate the properties of RealSense D435i camera, hence this being the camera used by the physical robot. This stack is an important tool for providing a better localisation and navigation with via detecting landmarks and and extracting 3D world coordinates with depth.

### 4.4.1 QR Code Detection

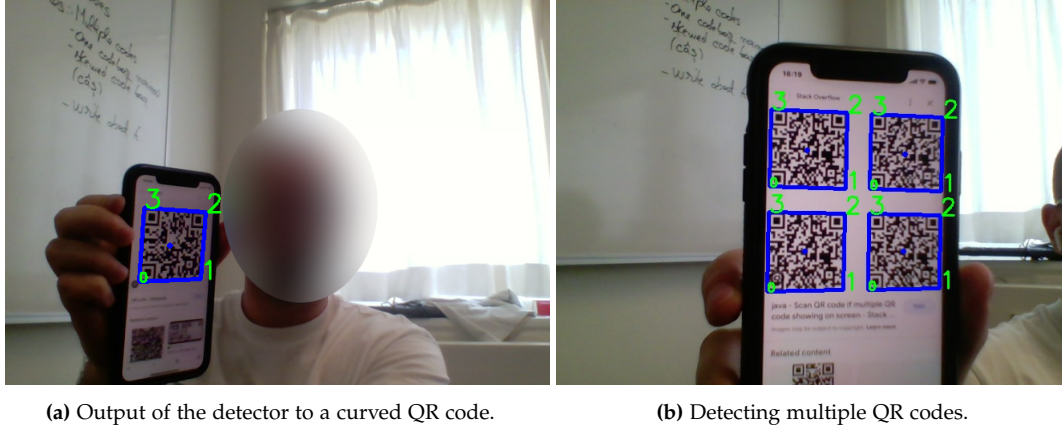
As seen in the sub-chapter 3.3 QR codes can improve the navigation stack. Since more information regarding the approach offered by [27] and also, using mainly OpenCV for detecting QR codes has shown to be buggy in previous approaches, it has been decided that a combination of OpenCV and ZBar would be use.

The method used in this project it will decode the information of the QR code, create a bounding box of the detected QR code and extract the center of that bounding box. This is going to be used for detecting the QR codes on the test tubes on Mars and for the navigation challenge in the ERC, having a role of a landmark.

The location of the corners which can be seen 4.4.1 were extracted via ZBar library. Once the corners' position was determined, a bounding box was created in order to emphasis the detection of the QR code.

To find the middle of the ROI, the following formula was used:

$$C_x = \frac{(P_{0x} + P_{1x})}{2} \quad (4.10)$$



**Figure 4.12:** Output of the detected QR code. Provides the position of the four corners of the bounding box (0-3), the center of the bounding box, and the decode message.

$$C_y = \frac{(P_{1y} + P_{2y})}{2} \quad (4.11)$$

where:

- $C_x$  and  $C_y$  are the homogeneous coordinates of  $x$ , respectively  $y$  of the center of the boundy box
- $P_{0x}$  and  $P_{1x}$  are the homogeneous coordinates in the  $x$  axis of the corner 0 and 1 as shown in Fig 4.4.1
- $P_{1y}$  and  $P_{2y}$  are the homogeneous coordinates in the  $y$  axis of the corner 1 and 2 as shown in Fig

#### 4.4.2 Real-World Coordinate Transformation

The main scope of this project is to navigate, map and localize. This being an extension of the ExoMyRover, which is going to be used for ERC navigation challenge and to detect the probes from Mars labeled with QR Codes. Hence, having a better understanding of the real-world environment is necessary.

In order to extract the real world coordinates, a camera Prim has been created in Isaac Sim and has been attached to the rover. This camera simulates a stereo camera and provides via ROS 2 topics information about: depth, imaging and camera matrices.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} F_X & S & P_X & 0 \\ 0 & F_Y & P_Y & 0 \\ 0 & 0 & 1.0 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ Z_s \\ 1 \end{bmatrix} \quad (4.12)$$

The mapping of data appearing in the world-frame scene to homogeneous image coordinates is described by Equation 4.12. The equation makes use of the extrinsic camera matrix, which relates the frame of the coordinates to the camera frame, and the intrinsic camera matrix projecting and applying an affine transformation to produce the homogeneous coordinates. Standard right-hand rule axis are assumed, with the Z axis being orthogonal to the image plane (i.e. being the depth). From the information offered by the ROS 2 CameraInfo the values of the intrinsic matrix are:  $F_X = 1466$ ,  $F_Y = 1467$  which represents the focal length measured in pixels,  $S = 0$  being the skew parameter,  $P_X = 640$ ,  $P_Y = 320$  representing the center coordinates of the camera windows. The matrix in the middle represent the extrinsic matrix, denoting the transformations suffered by the camera output.  $R$  and  $T$  are a  $3 \times 3$  and  $3 \times 1$  matrices describing the rotation and translation of the camera-frame relative to the real world. Since the frames of simulated camera coincide in the same axis. the extrinsic matrix becomes an  $I^4$ .  $[u, v, w]$  are the homogeneous image coordinate and in order to calculate the digital pixel coordinates simply compute  $u' = u/w$  and  $v' = v/w$  rounded to integers.  $[X_s, Y_s, Z_s, 1]^t$  is the scene point world coordinates measured in mm. Finding 3D coordinates it is possible only if  $Z_s$ , the depth, is known. As mentioned above, the depth is provided by the topic in ROS 2 /depth. Hence, the equations used to find the real world coordinates for  $X_s$  and  $Y_s$  can be seen in Equations 4.13 and 4.14.

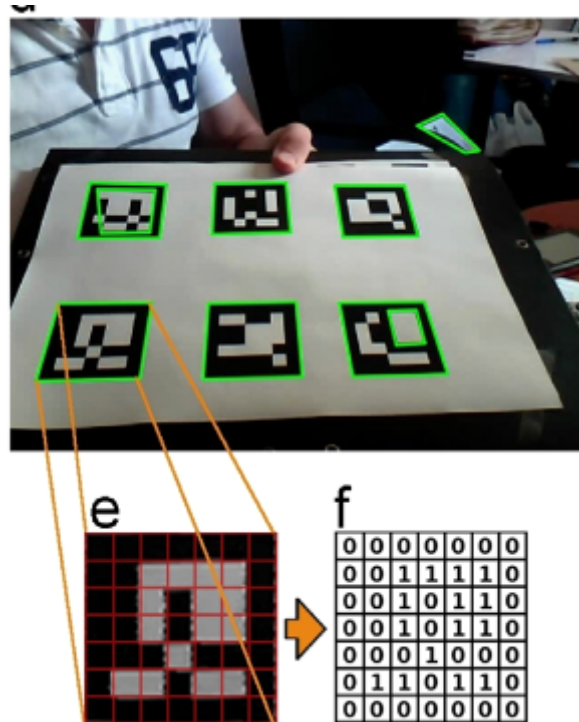
$$X_s = \frac{(u - P_X Z_s)}{F} = \frac{(u' Z_s - P_X Z_s)}{F} \quad (4.13)$$

$$Y_s = \frac{(v - P_Y Z_s)}{F} = \frac{(v' Z_s - P_Y Z_s)}{F} \quad (4.14)$$

#### 4.4.3 AprilTags Detection

Due to lack of QR codes materials offered by Isaac Sim Omniverse, an alternative for AprilTags has been used, Aruco ROS 2 package [41]. This package works as following and can be seen in Figure 4.14:

- Subscribe to the camera's output
- Segmentation of the image: grayscale the image to reduce noise and computation and using a Canny edge detector.
- Contour extraction and filtering by removing the irrelevant contours by setting a threshold



**Figure 4.13:** Extraction and analyzing of the message encrypted by bits [41]

- Analyze the marker code by computing the bit for each pixel cell of the matrix. This process can be seen in Figure 4.13

## 4.5 SLAM Stack

The main idea of the SLAM Stack is to map undiscovered areas and to provide a better understanding of the location of the rover. There are two SLAM algorithms deployed for this project, RTAB-Map (section 3.2.4) and Elbrus (section 3.2.3). Kimera is another contender due to its powerful toolbox, but due to its Docker Image not being updated since 2020 and having libraries out of date which can not be synchronized between them, it has not been chosen.

The SLAM algorithms have been deployed separately but tested in the same scenarios. The main sensor which is used for this implementation is mainly the RealSense D435i camera, which also has been simulated in Isaac Sim. The following subsections describe their setup and observed issues.



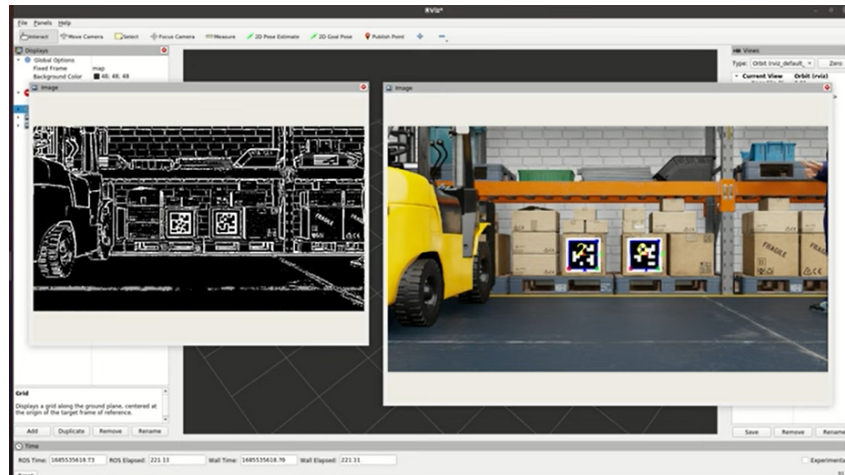


Figure 4.14: AprilTag detection done in Isaac Sim with ROS2 and Aruco package.

### 4.5.1 RTAB-Map Setup

RTAB-Map has been chosen for being part of the stated solution due to its easy manageable memory and its 3D mapping. These are satisfactory requirements for the scope proposed in Chapter 2.4 given that RTAB-Map provides visual odometry and IMU, plus a good understanding of the environment, even though the test results shown in Chapter 3.2.1 shown that it's not the best algorithm for outdoors SLAM.

In order to connect the robot to the RTAB-Map a connection between those two it is needed. This has been done via launching two Docker containers, one for Isaac Sim and one for ROS 2 as can be seen in Figure 4.10. The connection is done by enabling the *ROS 2 Bridge Humble* for Isaac Sim and setting for both containers the *ROS\_DOMAIN\_ID* to 5. RTAB-Map can be seen working in Figure 4.15.

The parameters provided for the RTAB-Map to work are:

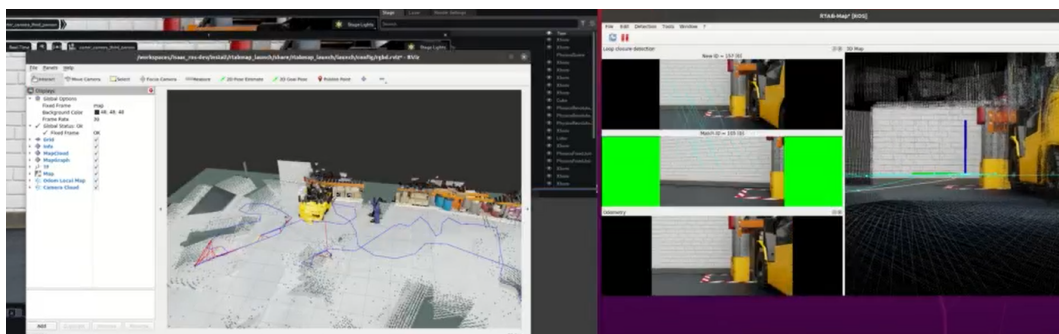
- `visual_odometry:=True` (enabling the VO node)
- `frame_id:=base_link` (the fundamental link of the rover and can be considered as its center)
- `odom_topic:=/odom` (provides information of rover's position via VO in xyz and orientation)
- `use_sim_time:=true` (simulation time is needed to synchronized the messages with the time inside Isaac Sim)
- `rgb_topic:=/rgb_right` (used to perform imaging)

- `depth_topic:=/depth_right` (provides depth information, needed to preform VO)
- `camera_info_topic:=/camera_info_right` (provides information based on the camera matrix, needed for the cameras to preform action such as extracting depth or to preform VO)

### 4.5.2 Elbrus Setup

Elbrus' is another SLAM algorithm chosen for implementation due to being tailored by NVIDIA and it's being compatible with NVIDIA's software such as Isaac Sim and hardware such as Orin. This comes as a package for ROS 2 made for stereo cameras and IMU to estimate VIO. This VSLAM approach has the best results of the KITTI VO and SLAM dataset in translation (0.94%) and rotation (0.0019 deg/m).

To run it, the Docker Container must be launched. The connection is done by enabling the *ROS 2 Bridge Humble* for Isaac Sim and setting for both containers the *ROS\_DOMAIN\_ID* to 5. After, the package has to be launched and it is given sparse point clouds and VO data (position xyz and orientation).



**Figure 4.15:** Implementing RTAB-Map in a warehouse environment for testing purposes. It can be seen that a 3D map has been created with meshes and the blue lines represent the VO. In the right panel it can be seen that a loop closure was found due to green color between the images that match

# 5 - Testing

In this section the different aspects of the SLAM and the navigation stack will be individually tested in both virtual and real-life environment in order to assess their suitability for a rover mission in uneven terrain. The requirements will first be discussed alongside their respective acceptance tests, then the test setup and results will be analysed.

## 5.1 Requirements

The scope of this project is to showcase how various technologies such as SLAM, navigation and landmark detection can be deployed on a rover in order for it to map and traverse an unknown and uneven terrain. An important aspect is to highlight how these systems perform in low-feature environments (deserts) versus indoor environments where there are far more visual features. The intent is for this system to be deployed on a celestial body such as the Moon or Mars. However, due to cost, the system presented will be deployed in simulation and, in a limited manner, in a laboratory environment.

There are two main components presented in this project. The first is the simulation environment Isaac Sim. The second is the overall perception and control system. As such, the requirements are divided into two sections to reflect the aforementioned observation.

### 5.1.1 Simulation Environment Requirements

Isaac Sim must provide a realistic environment, offer a ROS 2 plugin and provide a controllable robot in order for it to be suitable for testing. To this end it should:

- **Allow the import of 3rd party maps.** It should allow the user to import any map formed of meshes into its environment and use it as the terrain onto which the assets such as robots can be deployed.
- **Allow the use of a URDF defined robot into its environment.** URDF is a common standard in defining the build of a robot. The simulation environment should be able to accept this format and either use it directly or provide the tools necessary to convert it to its local format.

- **Provide a ROS 2 interface to access sensor data and control the robot.** ROS 2 is used to develop the perception and control stack and as such, the simulation environment should be able to interface with it.
- **Allow the deployment of the simulated rover.** The software version of the rover should be able to be deployed in the simulated environment and be controlled via the ROS 2 interface.

### 5.1.2 Perception and Control Requirements

For the system to be considered suitable for the desired task, it should:

- **Create a 3D map of its surroundings.** A 3D map of its environment should be created.
- **Detect landmarks.** Here, the landmarks are defined as either QR codes or an AprilTag. The vision system should be able to identify them and provide the information needed to calculate its location relative to the robot.
- **Detect wheel slippage.** The system should be able to detect when slippage occurs during its navigation.

### 5.1.3 Acceptance Tests

**Table 5.1:** Table specifying the requirements for the solution and the acceptance test for said requirements.

Requirement		Acceptance Test
1.1	<i>Allow the import of 3-rd party maps.</i>	Import a 3-rd party terrain mesh and deploy an asset over it to test the collision. Visually inspect that the asset does not fall through the terrain.
1.2	<i>Allow the use of a URDF defined robot into its environment.</i>	Import a URDF robot and deploy it in an environment. Visually inspect that all its components are available to be accessed and modified in its native environment.
1.2	<i>Allow the use of a URDF defined robot into its environment.</i>	Import a URDF robot and deploy it in an environment. Visually inspect that all its components are available to be accessed and modified in its native environment.
Continued on next page		

Table 5.1– continued from previous page

Requirement		Acceptance Test
1.3	<i>Provide a ROS 2 interface to access sensor data and control the robot.</i>	Deploy a robot in simulation and list in a separate terminal the ROS 2 topics generated by the simulation environment.
1.4	<i>Allow the deployment of the simulated rover.</i>	Deploy the rover in simulation and issue a velocity command. Visually inspect that the rover executes the correct command. Display in Rviz the left and right camera video feed to demonstrate simulated camera functionality.
2.1	<i>Detect landmarks.</i>	Run the landmark detection algorithm and add a bounding box around it in the video feed. Display the distance from the camera to it.
2.2	<i>Create a 3D map of it's surroundings.</i>	Run the Elbrus and RTAB-Mapping and show that a map can be generating with either algorithms. Visually inspect the map generation.
2.3	<i>Detect wheel slippage.</i>	Show that the trajectory output of either Elbrus or RTAB-Map reflects that the rover is not moving even though the command is to move forward. Visually inspect this behaviour.

## 5.2 Delimitation

Due to issues with Isaac Sim and time constraints, requirements 2.2 and 2.3 can only be tested in simulation. Furthermore, Isaac Sim does not provide a simple method of extracting the exact position of the rover (ground truth) in order to compare that with the visual odometry. As such, for this project, the accuracy of the visual odometry is not quantified. Mitigating actions are discussed in Chapter 7.1.1.

There are two robots that have been used in simulation, one is the 6-wheel rover discussed in Chapter 4.1 and the other is Carter, differential drive robot model provided by NVIDIA. Since the navigation stack is agnostic of the robot type, it can be deployed on either of them. Due to transformation frame issues with the 6-wheeled rover, all the perception and control requirements have been tested with Carter.

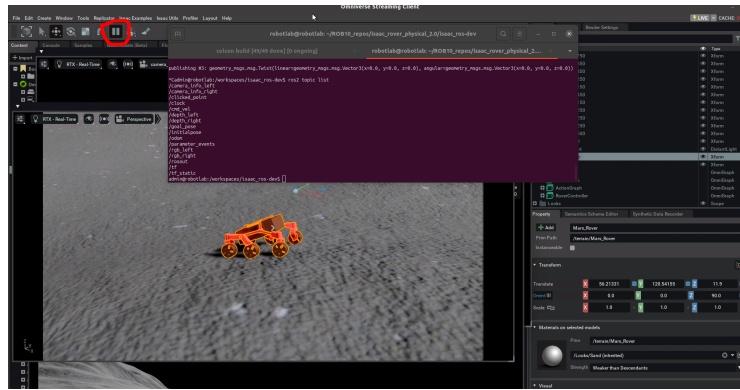


Figure 5.1: The simulation is started and the topics are being listed.

Lastly, the real-world landmark detection test has been performed with the RGB-D being handheld and not mounted to the robot. Requirement 2.1 does not necessarily need for the camera to be attached to the robot.

## 5.3 Tests

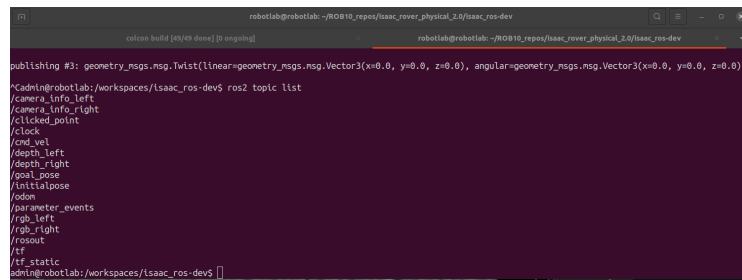
### 5.3.1 Test 1

This test consists of deploying the simulated rover on the surface of the moon and issue a move command using ROS 2 from an external terminal. The observed behaviour is that the rover begins moving and correctly outputs the left and right camera feeds in the Viewports in Isaac sim. Rviz is deployed and correctly streams the left and right RGB and depth image feed.

Figure 5.1 shows that the rover was loaded, the simulation is running and that the appropriate topics are listed. Figure 5.2 provides a closer view of the topics which are depth, RGB and info for the left and right cameras, odometry and transformation frames.

In Figure 6 located in Appendix .2.1 the Mars Rover asset can be found in the stage of the simulation. Note that it contains all the components such as Differential, FL\_Driver, etc. which have been converted from URDF to USD format. This satisfies Requirement 1.2.

In Figure 7 located in Appendix .2.1, proves that the custom made driver has been used in this simulation.

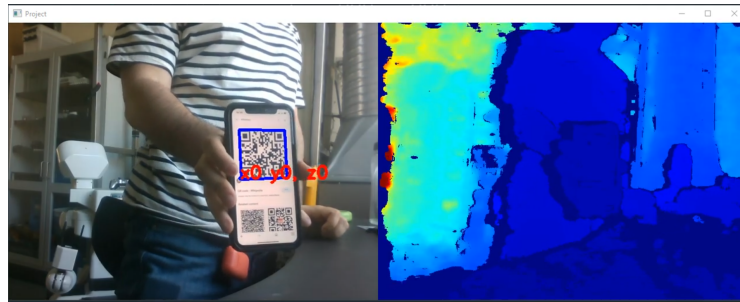


```

robotlab@robotlab:~/ROB10_repos/isaac_rover_physical_2.0/isaac_ros-dev
colcon build [play done] [0 ongoing]
robotlab@robotlab:~/ROB10_repos/isaac_rover_physical_2.0/isaac_ros-dev
publishing #3: geometry_msgs/Twist(lin=linear-geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), ang=angular-geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
Caching robotlab:/workspaces/isaac_ros-dev5 ros2 topic list
/camera_info_left
/camera_info_right
/clicked_point
/clock
/cmd_vel
/depth_left
/depth_right
/goal_pose
/initialpose
/rosout
/parameter_events
/rgb_left
/rgb_right
/rosout
/tf
/tf_static
admin@robotlab:/workspaces/isaac_ros-dev5 []

```

**Figure 5.2:** The ROS 2 topics listed once the simulation was started.



**Figure 5.3:** QR Code being detected using a RealSense Camera.

Lastly, the video [42] shows how the rover is controlled using a ROS 2 command and the images being streamed in Rviz. The environment is a Moon scenario using a map provided by University of Luxembourg. This video combined with the observations previously made in this test satisfy topics 1.1 to 1.4.

### 5.3.2 Test 2

This test demonstrates the functionality of the landmark detection algorithm. This project uses two algorithms. A custom made one made for this project which uses OpenCV and Pyzbar library. This is showcased in video [43] and Figure 5.3. In it, the RGBD Realsense camera is used to obtain the video and depth feed. In the left image the QR code is highlighted in a blue square which moves with its position. On the left the depth image is displayed. The X,Y and Z coordinates of the center of the QR Code are also displayed. With this information, the position of the landmark with respect to the robot can be calculated.

Moreover, a second test was made using the Aruco [41] ROS module. This can be seen in video [44]. This test was run using the NVIDIA Warehouse environment which already has AprilTags setup in its environment. The tags are highlighted with blue squares and, though difficult to observe, a XYZ axis is projected from the center of the tag. Aruco library provided a transformation frame from the camera

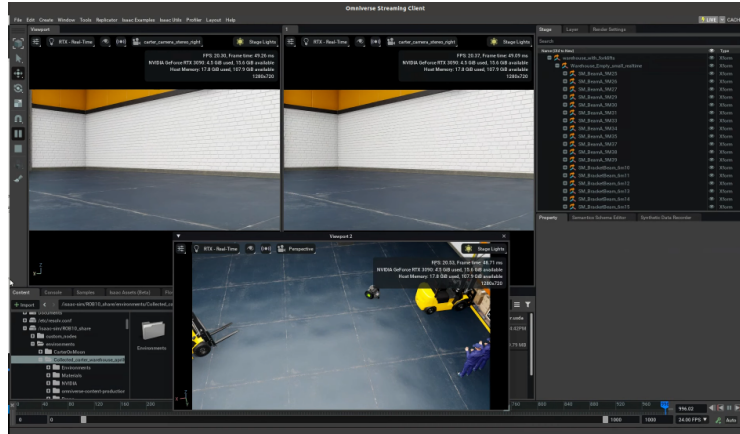


Figure 5.4: Carter deployed in the warehouse environment.

to the tag. The issue is that while it is displayed in the output image, the topic which should contain this transformation is empty. Time constraints prevented the debugging of this issue.

Currently there are two solutions that are able to detect and track both QR and AprilTags and offer sufficient information to compute their location with respect to the robot. This satisfies Requirement 2.1

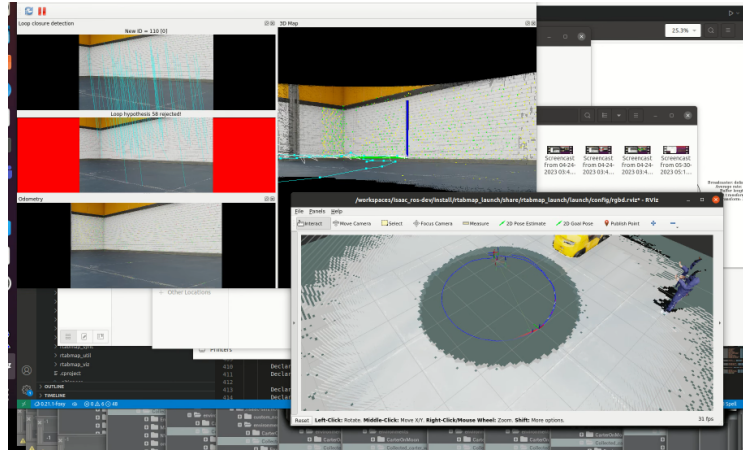
### 5.3.3 Test 3

This test will evaluate the performance of RTAB-Map in both indoors and outdoors. The robot used for this test is NVIDIA Carter due to its already made sensor setup and functioning TF tree. In both environments, the robot is given a constant linear and angular velocity, making it turn in a circle.

The outcome of the deployment in the warehouse (indoors) environment can be seen in video [45]. The right-hand side of the video displays the Isaac Sim environment Figure 5.4 with left and right camera feed on the top and a perspective view of the robot on the bottom. The left-hand side displays the output of RTAB-Map Figure 5.5.

From time 0:00 until 1:19 there is no loop closure. At 1:20, when the robot reaches the starting point a loop-closure is detected, as seen by the green border of the window on the far left. Note that the match ID is 1 meaning that it matched the current frame with the first keyframe. This is to be expected based on the description provided in Chapter 3.2.4.





**Figure 5.5:** The output of RTAB-Map. The 3 images on the left are the loop closure debug output. The middle image is the 3D map containing the keypoints. The bottom right image is the resulting 3D map.

At 1:29 there is another match between the current frame and frame 93. This time however, there are fewer matching keypoints indicated by the yellow border. Around this time when the matching starts to fail, the estimated path in the bottom right image shows that the robot's odometry is corrected more aggressively.

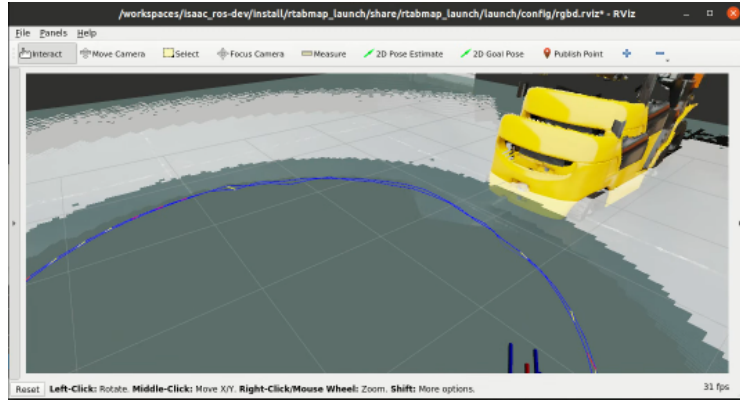
At 1:39 the matching is rejected altogether, signalled by the red border.

Between 1:45 and 2:05, there are no loop-closure. This is caused by the fact that the keypoints are far away on the walls. The walls are uniform thus providing little in terms of variation. At 2:05 however a loop-closure is achieved and the path is recursively updated. Notice how the blue circle is smoothed when the loop-closure is detected.

At time 2:50 (Figure 5.6), looking at the 3D map (bottom right image), it can be observed that the bottom part of the forklift is closer to the robot than the upper half. This is because the voxels closer to the robot have been updated on the first and second loop closer, while the rest were too far away to be considered for update.

This behaviour proves that it can perform a 3D map generation and also perform local updates whenever the same region is visited again.

RTAB-Map was also used in a Moon terrain as seen in video [46]. Here its performance is significantly worse. Firstly, the similarity of the terrain leads to false positives when it comes to loop closure detection. This is illustrated at time 0:39.



**Figure 5.6:** The local 3D map updated. The bottom part of the forklift is closer to the robot than the upper part.

At 1:22 the robot is roughly in the same place as it started, yet the path resembles a semi-circle. This further emphasises that the loop-closures were made in error. Moreover, between 0:32 and 0:40 the robot encounters a rough path which results in a very shaky and jerky video feed. At 0:40 when a loop-closure is made two large spikes can be observed in the corrected path.

Nonetheless, between 0:43 and 0:50, slippage can be observed in Isaac Sim. Looking at the 3D map output of RTAB-Map the position of the robot does not change while the slippage occurs, demonstrating that it can be used to detect slippage. It is to be noted that, while incorrect, a 3D map is still constructed.

The fact that a 3D map is constructed and slippage is detected, Requirements 2.2 and 2.3 respectively.

#### 5.3.4 Test 4

This test assess the performance of Elbrus SLAM. It is done in the same manner of **Test 3** (Chapter 5.3.3), where the Carter robot is commanded to go in a circle in both an indoor and outdoor environment. Both videos have the same format where the left-hand side displays Rviz showcasing the trajectory and point cloud, and the right-hand side displays the Isaac Sim environment.

The first test, shown in video [47], is made in the warehouse environment. Between time 0:00 and 1:12, there is no loop-closure and the trajectory is shown in blue. It can also be observed that while the rover is turning a point cloud is generated. This consists only of keypoints. It may be difficult to observe on a small screen, but they are in 3D.

At 1:13, after completing the circle, the first loop-closure is detected. The path from the first frame until the loop closure is marked in green and updated. The difference can be seen between the previous, estimated path marked in blue and the newly updated path.

While in theory capable of generating a 3D map, issues were encountered while trying to deploy it. In the end, time constraints prevented the generation of a 3D map.

The second test with Elbrus was made using the Moon terrain and can be seen in video [48]. In here the initial position and orientation of the robot was up a hill. Between 0:07 and 0:31, the robot is slipping and immediately the estimated trajectory matches this behaviour. From 0:32 until 0:45 the rover is not slipping and turning slightly to the left as expected.

At time 0:46 the angle of the 3D environment in Rviz is adjusted to showcase that indeed the robot is going uphill and that the path matches (at least visually) with the path observed in Isaac Sim. At around 1:01 the robot starts slipping and this is accurately depicted in Rviz.

The lack of 3D map generation implies that Requirement **2.2** was not satisfied. However, the wheel slippage was not only detected but done so in an accurate manner, meaning that Elbrus meets Requirement **2.3**.

# 6 - Results

## 6.1 Simulation Environment Results

Isaac Sim meets all the stated requirements. It generates accurate and reliable data which is then wrapped in ROS 2 topics. Moreover, it also allows the user to interface with the robots deployed in its environment in a robust manner via ROS 2 messages as well.

While it delivers a wide array of features, the overall system suffers from crashes in the best cases and silent, unpredictable malfunctions in the worst. The latter are particularly disruptive as it may lead the user to look in the wrong place while debugging. An example would be the fact that if a simulation is stopped or paused, and then started again, the TF frames are no longer valid and neither RTAB-Map nor Elbrus will function. The solution is to exit Isaac Sim and restart the program.

Even with the aforementioned issues, the system is still suitable for deploying complex, visual-based algorithms and use it to either test robots in complex scenarios or perform machine learning. The fact that provides a ground truth for complex assets such as simulated humans offers it a significant advantage over similar robotic simulation environments.

## 6.2 Perception and Control Results

### 6.2.1 Landmark Detection Results

The landmark detection algorithms used, both were able to detect and track the QR and April Tags (landmarks). Moreover, both demonstrated the capability of providing sufficient data to compute a transformation frame between the robot and the landmarks.

Nonetheless, both approaches had issues. Aruco was not outputting the transformation frames even though it showed in the result video that it was able to generate them. This most likely stemmed from the fact that it was not given the correct transformation frame and base link name. Various combinations that have been demonstrated to work with Elbrus and RTAB-Map have been tried, but none

were successful. Time constraints prevented further debugging.

The custom code suffered from sporadic errors in depth perception. This behaviour has been observed in other projects and is caused by the depth calculation and RGB alignment not being run with with CUDA cores. Our tests was run on a laptop which did not have CUDA capabilities. The symptom of this issue is that, sporadically, there will be a zoom caused by the reduction in image resolution.

### 6.2.2 Visual SLAM Results

**RTAB-Map** performed well in the indoors environment. It was able to detect loop-closures and smoothed the estimated path every time this occurred. It proved that it can generate a 3D map and update the voxels whenever it passed through an area previously visited.

Nonetheless, while it was facing a distant wall were few close features were detected, its performance was impacted. This was even more apparent when it was deployed in the Moon environment. In that scenario its odometry estimation was unusable.

Yet, while the odometry was poor, it was still able to detect when the robot was getting stuck or slipping. This is to be expected as in this scenario the odometry is calculated from frame to frame (F2F) and the features are tracked with optical flow.

**Elbrus SLAM** was unfortunately unable to be set up in order to generate a 3D map. However, observing its estimated paths and loop-closures, it appears that its performance is superior to that of RTAB-Map. The estimated path it generated in the Moon environment appears to be accurate, yet without ground truth it is very difficult to tell from the video alone. A explanation for its performance lies in the fact that, it incorporates IMU data and fuses it with the visual odometry.

Based on the results, Elbrus SLAM would be a more suitable candidate for visual odometry if the robot is to be deployed in a rugged, featureless terrain such as the Moon or Mars.

# 7 - Discussion and Conclusion

## 7.1 Discussion

For a robot to navigate autonomously navigate an environment that is rugged and has sparse visual keypoints, it requires a complex and modular system. This system has to perform localisation, mapping and detect landmarks to which the robot has to navigate.

This project discussed the state of the art of these technologies and provided a simulation environment to test their performance. The chosen simulation environment was Isaac Sim as it met the necessary requirements to provide reliable and high quality sensor data and generate realistic environments.

Furthermore, two approaches for landmark detection have been assessed both in simulation and in the real-world. These have proven to be highly reliable and simple to deploy.

Lastly, two Visual SLAM algorithms have been tested: Elbrus and RTAB-Mapping. From the results of these tests, Elbrus SLAM performed better than RTAB-Map achieving a far more accurate estimation of the robot's trajectory in the Moon terrain. This assessment is purely based on visual observations. The ground truth from Isaac Sim was difficult to obtain. Attempts at it have yielded no results and incurred a high time cost.

Some observations can be made regarding the performance of the two algorithms. The first is that both are able to detect when the robot is stuck or experiences slipping. Optical flow is a useful approach when there are no immediate distinct features nearby. The drawback is that the camera has to be pointing at the ground and at the same time has to be very close to it. The further it is from the tracked features, the less accurate the optical flow estimate.

The second observation is that combining IMU with visual odometry greatly improves the reliability of the measurements. This is expected as the IMU can offer more reliable information when the robot is traversing an area with sparse visual features.

A solution for the SLAM problem will most likely have to include optical flow and IMU in its sensor fusion.

### 7.1.1 Future Work

#### Extracting Ground Truth Position of the Rover

Extracting the ground truth position from Isaac Sim, console has proven to be a difficult task. Once the extraction of this ground truth will be developed, Test 3 and 4 will be redone and their odometry will be compared with the ground truth, thus providing quantifiable assessment of their performance.

Isaac Sim provides Action Graphs to interrogate the attributes of a Prim such as it X,Y,Z and orientation, however, it does not provide a method of exporting this outside of Isaac Sim. The solution envisaged is the creation of a custom node that intercepts this data and either streams it to a file or generates a custom ROS 2 node that publishes this information.

#### Navigation Stack

Inspired by Zhang et al. [27] an autonomous navigation helped by QR messages will be implemented. This method implies detecting the landmarks, in this case QR codes, navigate to them and, at certain distance, execute the commands encoded in the QR code. An example of this type of landmark was offered by ERC and can be seen in Figure 8. The motivation behind this approach comes from the good results presented in the aforementioned paper and due to the improvements in speed and efficiency during autonomous navigation. This behaviour will be integrated using the Nav2 ROS2 module.

#### Physical Rover Testing

As stated previously, Isaac Sim setup was a time consuming process which occupied a significant portion of the time allotted for this project. Unfortunately, this meant that the perception stack was only deployed in simulation and not on the real rover. Since this rover will be used in future competitions, it is important that a working solution be given to it.

On the one hand, the deployment should not incur many problems as the interface between the hardware and the perception and navigation stack is done in ROS 2. The topics, publishers and subscribers will be the same as those used with Isaac Sim. Furthermore, the perception stack is run within a Docker container. This

should further help with deployment.

It is expected that problems will occur when the sensors will be used. From previous experience, RealSense cameras have had issues with Docker containers whereby the USB was not detected or it was missing libraries that were incompatible with the hardware.

### **Kidnapped Robot Problem**

In order to solve the kidnapped robot problem, the generated 3D map has to be saved along with 3D features needed for loop-closure detection. This feature is required for the situation where the rover is moved by sandstorms on Mars or its sensors malfunction, requiring a restart. This has not been a priority due to the perception stack being deployed in controlled or simulated environments.

This feature will be needed if the rover is to be used in competitions such as the ERC. The implementation will require that a OctoMap server is setup and then connected to the SLAM stack.

### **Automatic Environment Generation**

A challenge faced while setting up the environment is that manually importing an asset such as the rover was a tedious and complicated task. The user had to ensure that the paths to the assets and all of their sub-components were consistent and not corrupted during a copy-paste operation. Isaac Sim provides a python script interface that can be used to procedurally generate an environment in which the adding of assets was an automated task. This will save time required for deployment and reduce the errors incurred from copying assets between different environments.

## **7.2 Conclusion**

This project has presented the challenges of deploying a robot in extreme environments such as the Moon or Mars. It then began presenting solutions to those challenges specifically, in the domain of localisation, mapping and navigation with an exposition of the state of the art. An analysis then followed on the advantages of using visual odometry to perform the localisation and mapping.

Two Visual SLAM approaches have been selected: Elbrus SLAM and RTAB-Map. To test these, a simulation environment was used. The software used was Isaac



Sim, chosen for its wide array of features, quality of sensor data and visually realistic environments.

Testing showed that by RTAB-Map was not a suitable candidate for the required task, this being better suited for Elbrus. However, a proper solution would have to use aspects from both algorithms that have proven to be very useful such as optical flow or the use of IMU.

Future work is needed in order to obtain a solution that is reliable and robust. This solution will also have to be deployed on a real rover, not just in a simulation.

# Bibliography

- [1] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. “A review of mobile robots: Concepts, methods, theoretical framework, and applications”. In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596. doi: 10.1177/1729881419839596. eprint: <https://doi.org/10.1177/1729881419839596>. URL: <https://doi.org/10.1177/1729881419839596>.
- [2] *What’s Mars Solar Conjunction, and Why Does It Matter?* <https://www.jpl.nasa.gov/news/whats-mars-solar-conjunction-and-why-does-it-matter>. Accessed: 2023-05-21.
- [3] Cyrill Stachniss and Wolfram Burgard. “Exploring unknown environments with mobile robots using coverage maps”. In: *IJCAI*. Vol. 2003. 2003, pp. 1127–1134.
- [4] *NASA Robots Compete in DARPA’s Subterranean Challenge Final*. <https://www.nasa.gov/feature/nasa-robots-compete-in-darpa-s-subterranean-challenge-final>. Accessed: 2023-06-1.
- [5] *Mars in our Night Sky*. <https://mars.nasa.gov/all-about-mars/night-sky/solar-conjunction/>. Accessed: 2023-05-21.
- [6] *NASA Will Go Silent During the Mars Solar Conjunction*. <https://science.howstuffworks.com/mars-solar-conjunction-news.htm>. Accessed: 2023-06-1.
- [7] Kenneth Farley, Kenneth Williford, Kathryn Stack, Allen Chen, Manuel Torre, Kevin Hand, Y. Goreva, Christopher Herd, Ricardo Hueso, Yang Liu, Justin Maki, German M. Martinez, Robert Moeller, Adam Nelessen, Claire Newman, Daniel Nunes, Adrian Ponce, Nicole Spanovich, and Roger Wiens. “Mars 2020 Mission Overview”. In: *Space Science Reviews* 216 (Dec. 2020). doi: 10.1007/s11214-020-00762-y.
- [8] *Mars Report: Dust Storms on Mars*. <https://mars.nasa.gov/resources/26555/mars-report-dust-storms-on-mars/>. Accessed: 2023-05-21.
- [9] *The Fact and Fiction of Martian Dust Storms*. <https://www.nasa.gov/feature/goddard/the-fact-and-fiction-of-martian-dust-storms>. Accessed: 2023-06-1.

- [10] Guy Pyrzak, Michael McCurdy, Kenneth J. Rabe, Jeffrey S. Norris, Joseph C. Joswig, Jason M. Fox, Thomas Crockett, and Mark W. Powell. "Targeting and Localization for Mars Rover Operations". In: *2006 IEEE International Conference on Information Reuse and Integration*. 2006, pp. 23–27. doi: 10.1109/IRI.2006.252382.
- [11] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), 1309–1332.
- [12] Alexey Merzlyakov and Steve Macenski. "A Comparison of Modern General-Purpose Visual SLAM Approaches". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 9190–9197. doi: 10.1109/IROS51168.2021.9636615.
- [13] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping". In: 2020, pp. 1689–1696. doi: 10.1109/ICRA40945.2020.9196885.
- [14] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. doi: 10.1109/CVPR.1994.323794.
- [15] J.-Y. Bouguet. "Pyramidal implementation of the lucas kanade feature tracker". In: 1999.
- [16] M. Fischler and R. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395. url: /brokenurl#http://publication.wilsonwong.me/load.php?id=233282275.
- [17] Frank Dellaert. "Factor Graphs and GTSAM: A Hands-on Introduction". In: 2012.
- [18] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: Sept. 2017, pp. 1366–1373. doi: 10.1109/IROS.2017.8202315.
- [19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. doi: 10.1109/ICCV.2017.322.

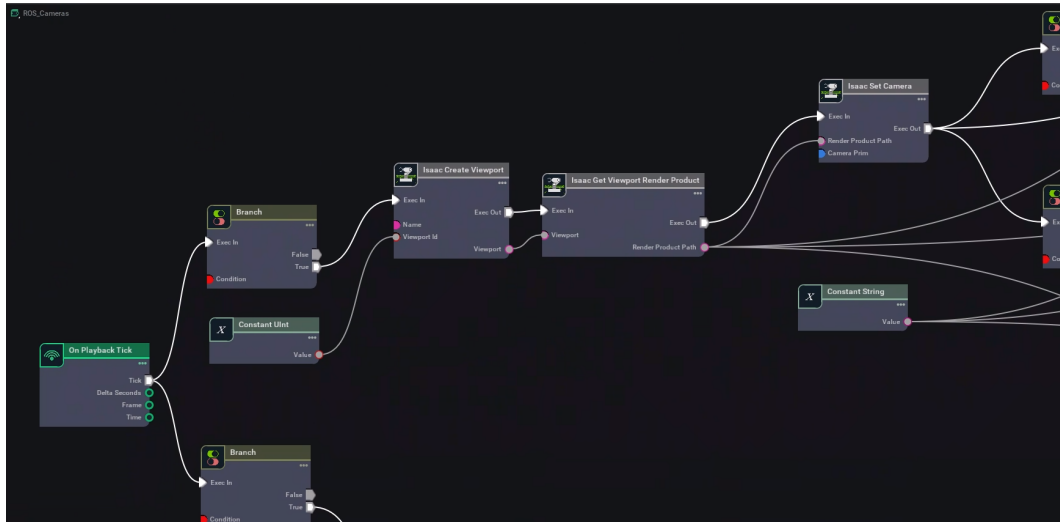
- [20] NVIDIA Corporation. *Elbrus Stereo Visual SLAM based Localization*. [https://docs.nvidia.com/isaac/packages/visual\\_slam/doc/elbrus\\_visual\\_slam.html#inertial-measurement-unit-imu-integration](https://docs.nvidia.com/isaac/packages/visual_slam/doc/elbrus_visual_slam.html#inertial-measurement-unit-imu-integration). Accessed on 20.05.2023. 2023.
- [21] Mathieu Labbé and François Michaud. "Online global loop closure detection for large-scale multi-session graph-based SLAM". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2661–2666. doi: 10.1109/IROS.2014.6942926.
- [22] Davide Scaramuzza and Friedrich Fraundorfer. "Visual Odometry [Tutorial]". In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 80–92. doi: 10.1109/MRA.2011.943233.
- [23] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. doi: 10.1109/CVPR.1994.323794.
- [24] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. "BRIEF: Binary Robust Independent Elementary Features". In: vol. 6314. Sept. 2010, pp. 778–792. ISBN: 978-3-642-15560-4. doi: 10.1007/978-3-642-15561-1\_56.
- [25] David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. doi: 10.1023/B:VISI.0000029664.99615.94.
- [26] Pranesh Kumar and Arti Khaparde. "QR Code Detector and Follower with Kalman Filter". In: *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)*. 2022, pp. 1423–1426. doi: 10.1109/IIHC55949.2022.10060570.
- [27] Huijuan Zhang, Chengning Zhang, Wei Yang, and Chin-Yin Chen. "Localization and navigation using QR code for mobile robot in indoor environment". In: *2015 IEEE International Conference on Robotics and Biomimetics (RO-BIO)*. 2015, pp. 2501–2506. doi: 10.1109/ROBIO.2015.7419715.
- [28] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. doi: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [29] NVIDIA Corporation. *NVIDIA Isaac Sim*. [https://docs.omniverse.nvidia.com/app\\_isaacsim/app\\_isaacsim/overview.html](https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/overview.html). Accessed on 20.05.2023. 2023.
- [30] Pixar Animation Studios. *Introduction to USD*. <https://openusd.org/release/intro.html>. Accessed on 20.05.2023. 2022.

- [31] Open Robotics. *Gazebo*. <https://gazebo-sim.org/docs>. Accessed on 20.05.2023. 2022.
- [32] *Nav2*. <https://navigation.ros.org/index.html>. Accessed: 2023-05-25.
- [33] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2718–2725. doi: 10.1109/IROS45743.2020.9341207.
- [34] *Navigation Concepts*. <https://navigation.ros.org/concepts/index.html>. Accessed: 2023-05-30.
- [35] Laia Vives Benedicto Lionel Emil Burg Mads Rossen Madsen Anton Bjørndahl Mortensen Emil Tribler Pedersen. "Narrowing the Reality Gap: Two-stage Learning Process for Mapless Navigation on a Mars Rover". In: Nov. 2022.
- [36] Miro Voellmy and Maximilian Ehrhardt. "ExoMy: A Low Cost 3D Printed Rover". In: Oct. 2020.
- [37] NVIDIA corporation. *NVIDIA Orin Data Sheet*. <https://developer.nvidia.com/embedded/downloads#?search=Data%20Sheet>. Accessed on 21.05.2023. 2022.
- [38] Intel® *realsense™ depth camera D435 (starter kit)*. 337020-005. Intel Corporation. 2019. URL: <https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf>.
- [39] BS. Miron AM. Smau. *Custom Isaac Sim Rover Controller*. [https://github.com/abmoRobotics/isaac\\_rover\\_2.0/tree/ROB10-boglex/isaac\\_sim/custom\\_nodes/aau.rover.base\\_controller2](https://github.com/abmoRobotics/isaac_rover_2.0/tree/ROB10-boglex/isaac_sim/custom_nodes/aau.rover.base_controller2). Accessed on 20.05.2023. 2023.
- [40] *Isaac Sim Requirments*. [https://docs.omniverse.nvidia.com/app\\_isaacsim/app\\_isaacsim/requirements.html](https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/requirements.html). Accessed: 2023-05-21.
- [41] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2014.01.005>. URL: <https://www.sciencedirect.com/science/article/pii/S0031320314000235>.
- [42] *Test 1. Rover deployed on the Moon*. <https://youtu.be/B531-1IPbjw>. Accessed: 2023-06-1.

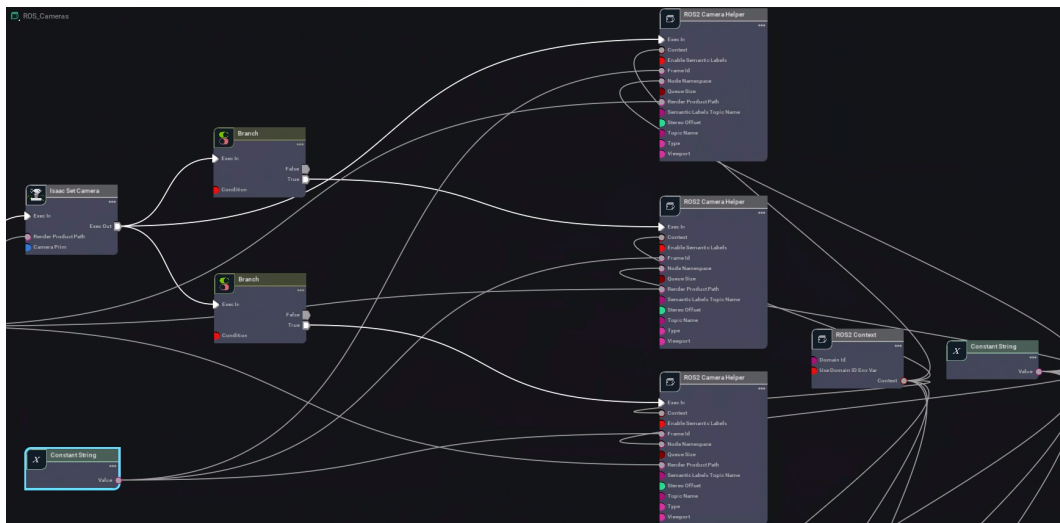
- [43] *Test 2. QR Code detected using a RealSense RGBD camera.* <https://youtu.be/txVcP33NCkc>. Accessed: 2023-06-1.
- [44] *Test 2. April Tag detected using using the Aruco library in Isaac Sim.* [https://youtu.be/ArD\\_X5dhYH8](https://youtu.be/ArD_X5dhYH8). Accessed: 2023-06-1.
- [45] *Test 3. RTAB-Map deployed in warehouse.* <https://youtu.be/KoGJC112KjY>. Accessed: 2023-06-1.
- [46] *Test 3. RTAB-Map deployed on the Moon terrain.* <https://youtu.be/O1JYQz-tgfM>. Accessed: 2023-06-1.
- [47] *Test 5. Elbrus deployed in warehouse.* <https://youtu.be/AzWcLAcJx6M>. Accessed: 2023-06-1.
- [48] *Test 5. Elbrus deployed in warehouse.* [https://youtu.be/Dr\\_hCFDlds4](https://youtu.be/Dr_hCFDlds4). Accessed: 2023-06-1.

## .1 Isaac Sim Setup

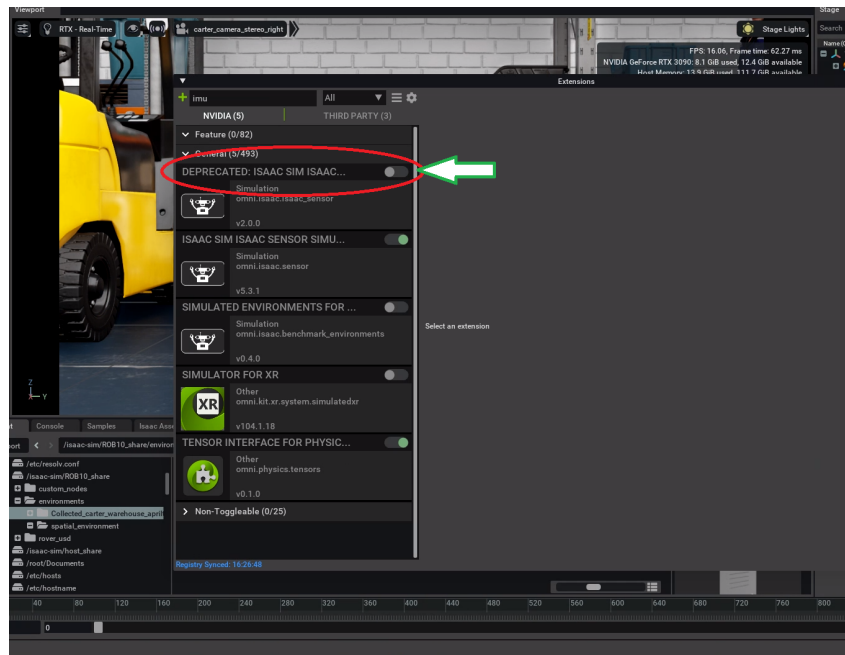
### .1.1 Omnigraph Setup



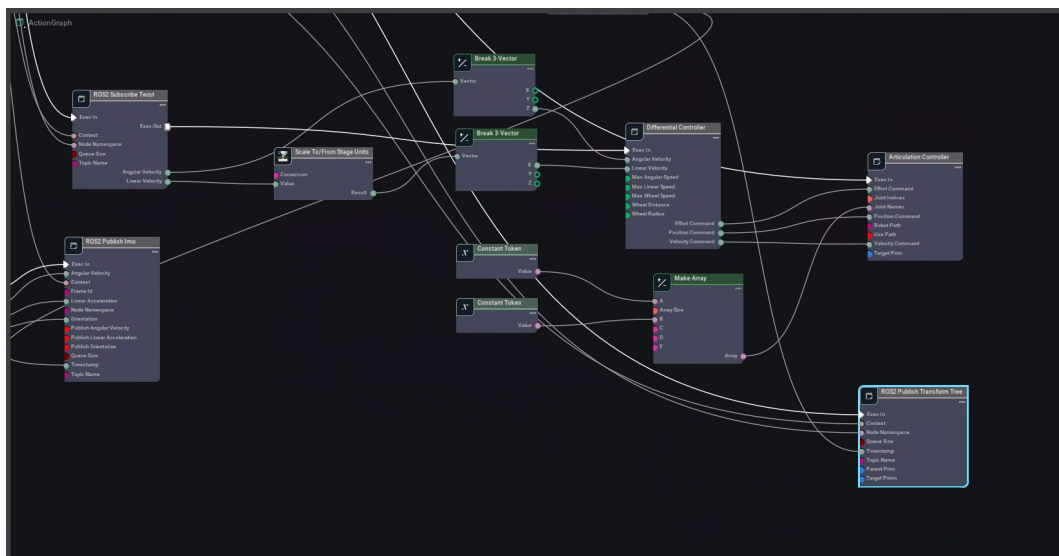
**Figure 1:** The first part of the Action Graph chain required to publish an image topic. Here, the Viewport is created, then the rendered image is obtained. The *Isaac Set Camera* is used to supply the camera Prim path. The *Branch* node is the equivalent of an "if" statement that enables or disables this Action Graph branch.



**Figure 2:** The second part of the Action Graph chain required to publish an image topic. Here the rendered image mentioned in Figure 1 is sent to the *ROS2 Camera Helper* which publishes the topic. There are 3 *ROS2 Camera Helper* nodes for the 3 topics: RGB, depth, and camera\_info.

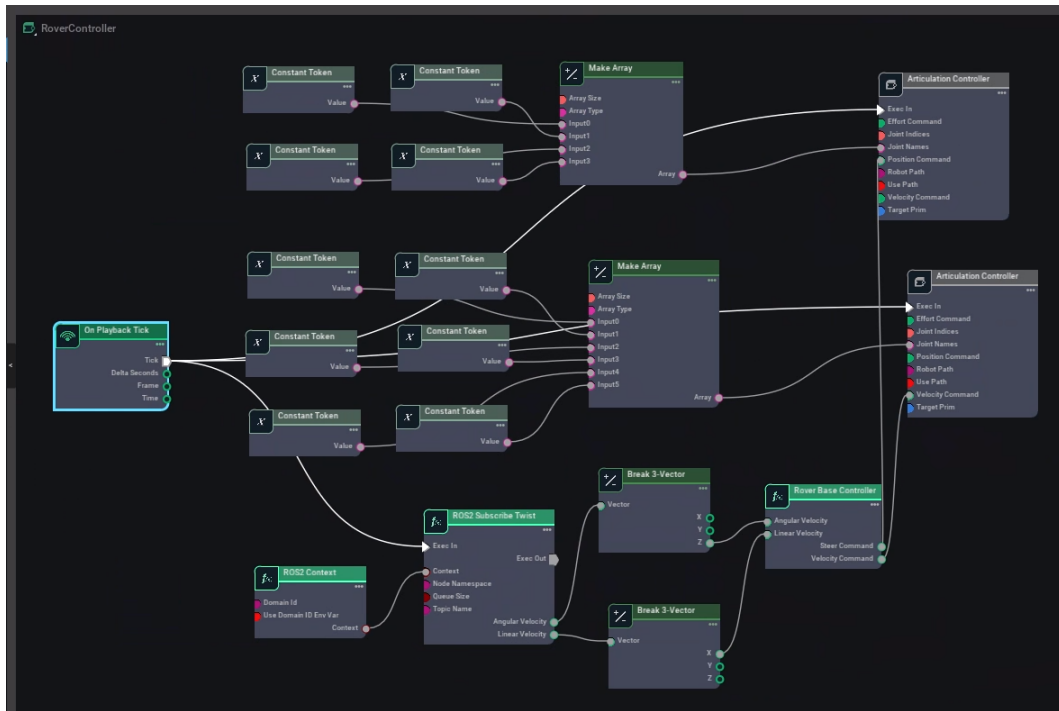


**Figure 3:** The deprecated plugin that has to be disabled in order for the IMU sensor to work.



**Figure 4:** The node chain required to convert a `/cmd_vel` message into an angular velocity for the wheels of a differential drive robot.





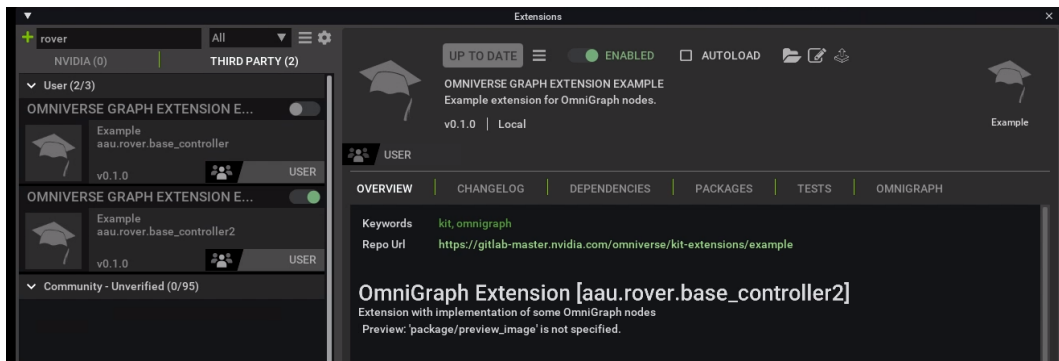
**Figure 5:** The node chain required to convert a `/cmd_vel` message into a angular velocity and steering angles for the wheels of the 6 wheeled rover.

## .2 Testing

### .2.1 Test 1



**Figure 6:** The Mars Rover Asset is present in the Stage of the environment. Note that all its components (Differential, FL\_Boogie, etc.) are listed and can be modified.



**Figure 7:** Proof that the custom made rover driver has been correctly imported as a plugin in Isaac Sim.



**Figure 8:** A presentation of a landmark shown by ERC which it will be placed on a special Martian track. This landmark comes to help rovers during the competition to traverse autonomously the trail

view\_frames Result  
Recorded at time: 1685458084.606155

