Design of an Utility-Frontier Based Exploration Strategy Coupled with 3D Object Localization for Modular Open-Source Autonomous Rovers

A State Machine Framework Approach Implemented and Tested on the Lunar Rover Mini

Design of an Utility-Frontier Based Exploration Strategy Coupled with 3D Object Localization for Modular Open-Source Autonomous Rovers

A State Machine Framework Approach Implemented and Tested on the Lunar Rover Mini

Master Thesis

for the purpose of obtaining the degree of Master of Science
at Delft University of Technology
to be defended publicly on
Wednesday 17, December 2025 at 10:30 o'clock

by

Tomás PLÁCIDO DE CASTRO

Student number: 6047459

Project duration: April, 2025 - October, 2025

Composition of the thesis committee:

Dr. S. Speretta,	Delft University of Technology, *Chair*
Dr. ir. C. de Wagter,	Delft University of Technology, *Independent examiner*
Dr. J. Guo,	Delft University of Technology, *Supervisor*
Dr. -Ing. A. Wedler,	German Aerospace Center, *Supervisor*

Faculty of Aerospace Engineering

Thank you to everyone who played a part in my academic journey that culminated in the writing of this thesis.

Tomás Plácido de Castro

Abstract

An increasing number of open-source, affordable, and compact robotic platforms built from commercial off-the-shelf components are being developed to approach the capabilities of complex space rovers. This trend is motivated by the fact that traditional space rovers take many years to develop, are extremely expensive, and are typically mission-specific, making them inaccessible as learning platforms for students and researchers. This thesis presents an open-source, modular state machine framework that integrates a utility-frontier-based exploration strategy with real time 3D object localization for low-cost autonomous rovers, and validates the approach on DLR's Lunar Rover Mini (LRM).

Exploration is decomposed into reusable low-level state machines within a hierarchical architecture that handles frontier detection, clustering, and filtering, as well as position and orientation monitoring, implemented in RAFCON, DLR's open-source software tool to manage autonomous tasks. A utility function balances information gain, computed by performing 3D ray casting, and travel cost, determined by the estimated travel time to a frontier centroid, to decide the next frontier centroid.

Moreover, a frontier coverage algorithm is employed to determine the most efficient set of orientations that maximize information gain, based on the normalized cumulative entropy within the camera's iFOV across the full azimuth range around each frontier centroid.

A parallel perception pipeline runs, in real time, a quantized, custom-trained YOLOv7 model on the LRM's Intel NUC to detect objects of interest, and compute their 3D coordinates in the global map frame using stereo depth data and the camera's intrinsic and extrinsic parameters.

The design was tested in DLR's Planetary Exploration Laboratory across a set of benchmarks and mission scenarios. Results show the proposed *Utility With Edges* strategy performs better than classical *Closest*-frontier and *Entropy*-only methods, with the *Utility With Edges* strategy improving exploration efficiency by 27% over the *Closest*-frontier baseline, while achieving accurate real time CPU-only object detection and localization.

Key limitations of this implementation include the computational cost of ray casting, the use of a full OctoMap that stores the full range of occupancy probabilities instead of a binary map, and drift in the visual odometry estimates.

Future work recommendation entail studying how more complex utility functions affect selecting the next frontier centroid, building a fully autonomous mission pipeline with autonomous object grasping, and testing the implementation of the open-source ready-to-use exploration strategy on other robotic platforms with user-defined parameters.

Contents

Nomenclature

ASA	Acrylonitrile Styrene Acrylate
BFS	Breadth-First Search
BoW	Bag-of-Words
BRIEF	Binary Robust Independent Elementary Features
CCW	Counterclockwise
CFE	Cogent Frontier Exploration
CPU	Central Processing Unit
DC	Direct Current
DFS	Depth-First Search
DLR	German Aerospace Center
DOF	Degrees of Freedom
FOV	Field of View
FSM	Finite State Machine
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HCSM	Hierarchical Concurrent State Machine
iFOV	Instantaneous Field of View
IMU	Inertial Measurement Unit
LRM	Lunar Rover Mini
LN	Links and Nodes
MAV	Micro Aerial Vehicle
NBV	Next Best View
NUC	Next Unit of Computing
OBC	On-Board Computer
OoI	Object of Interest
OS	Operating System
PEL	Planetary Exploration Laboratory
POV	Point of View
PWM	Pulse Width Modulation
RC	Radio Control
RH	Receding Horizon
RM	Robotics and Mechatronics
RMSE	Root Mean Square Error
ROS	Robot Operating System
RRT	Rapid-exploring Random Tree
RTAB-Map	Real-Time Appearance-Based Mapping
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
VNC	Virtual Network Computing
VO	Visual Odometry
WFD	Wavefront Frontier Detector
YOLO	You Only Look Once

1 Introduction

The development of mobile robotic platforms has transformed space exploration by enabling sample retrieval on the surface of planetary bodies. While the Apollo Program conducted 11 crewed missions, including six successful lunar landings between 1969 and 1972, advancing scientific exploration of the Moon [1], such missions are costly, limited to nearby celestial objects, and carry considerable risks to human life. Moreover, although significant progress has been made in rover teleoperation since the successful lunar landings of Lunokhod 1 and Lunokhod 2 in the 1970s [2], increasing communication delays make real-time control from Earth impractical for more distant targets. For example, Mars exploration missions, beginning with the Mars Pathfinder mission that deployed the Sojourner rover in 1997 [3], operate either on a delayed "send-command-and-execute" model or rely heavily on onboard autonomous systems. Recent examples include NASA's Perseverance rover, which landed on Mars in 2021 [4], and ESA's ExoMars rover, planned for launch in 2028 [5], both incorporating advanced autonomous capabilities. Additionally, the MMX rover, developed jointly by DLR and CNES and scheduled for a 2026 launch, will land on Phobos as part of JAXA's MMX mission [6, 7]. This highlights the importance of space rovers capable of independently navigating terrain, performing experiments, and collecting samples, as they form a cornerstone of future deep-space missions.

These important capabilities have driven extensive research within the scientific community into optimal strategies for exploring unknown environments, with frontier-based exploration, where frontiers are regions that separate explored free space from unknown space, introduced in 1997, emerging as one of the most widely used approaches for autonomous exploration [8]. Since this breakthrough, numerous strategies have been proposed in the literature, for both single- and multi-robot exploration, each offering significant advantages and aiming to improve the efficiency of autonomous exploration [9, 10, 11]. However, these approaches are often system-specific, making them difficult to apply to other robotic platforms, and some are evaluated only in simulated environments.

Ultimately, they seek to identify the set of parameters, such as travel cost and information gain, that maximize a chosen objective function, with substantial

literature dedicated to comparing different methods [12, 13, 14]. As previously mentioned, another important capability is autonomous sample retrieval, where object detection plays a crucial role. This field of knowledge expands beyond the space robotics domain and has been gaining significant traction in recent years, largely as a result of progress in AI and machine learning. Several state-of-the-art real-time object detection algorithms have been proposed in this context [15, 16]. All things considered, the integration of a frontier-based exploration strategy coupled with real-time object detection, allows a robotic platform to efficiently map the environment while detecting objects of interest. These capabilities, when combined with autonomous object grasping, enable fully autonomous exploration, which are then tested in analogous mission environments on Earth that simulate space conditions.

However, these advanced systems pose significant challenges for STEM researchers and students, including high hardware costs, closed-source and non-modular software, and the complexity involved in manufacturing and testing, which makes them impractical as experimental platforms. In response, there has been increasing interest in developing more accessible, open-source alternatives built from off-the-shelf components, while still incorporating many of the capabilities of high-end systems. Such example that addresses this gap is the Lunar Rover Mini (LRM), Fig. 1.1. Inspired by the locomotion system of the ExoMars rover and utilizing some of the same software packages as DLR's Lightweight Rover Unit (LRU), designed for autonomous search and exploration [17], the LRM offers a cost-effective, modular, and easy-to-assemble solution, sharing software components with other DLR robotic platforms, and benefits from an open-source middleware framework, such as ROS, RAFCON [18], and Simulink.

Other similar modular, built with off-the-shelf components, robotic platforms tailored to research and educational purposes have increasingly being adopted due to their ease of prototyping, assembly, affordability, and compatibility with widely used software.[1]

Moreover, ROS presents itself as the most used flexible framework for writing robot software. Hence, it comes as no surprise that almost all autonomous exploration implementations in literature are done through ROS. Yet, most implementations remain closed-source, and the few that are available typically offer only basic functionalities when it comes to autonomous exploration, such as moving to the closest frontier [20] or simply considering the 2D occupancy grid instead of the more informative 3D point cloud [21].

In addition, a gap in the literature exists regarding few documented open-source finite state machine-based ROS implementations to coordinates the execution of autonomous tasks [22], even though they are widely recognized as powerful tools for introduction to robotics, with their importance scaling while task complexity grows [23].

This report focuses on the design of an autonomous exploration strategy built on an open-source framework, that combines ROS with a hierarchical state machine

[1] DLR took the first step in this direction by launching ASURO, followed by NASA's JPL Open Source Rover, launched in 2019. Later, in November 2020, ESA developed the ExoMy rover [19].

execution software, implemented and tested on the LRM. The devised strategy can explore a greater space volume in the same amount of time as existing ROS-based open-source approaches in the literature, without compromising performance. Its core relies on a utility function that balances travel cost-estimated by the distance to a frontier-with information gain, which is computed at each frontier centroid through ray casting to estimate entropy, a measure of uncertainty in the occupancy state of the environment.

The results indicate that it offers a promising alternative for adoption within open-source robotics, since the middleware is designed to be extensible, and can be applied to future research and missions involving other small ground-based rovers. Furthermore, it advances toward a fully autonomous mission pipeline, by integrating a real time object detection algorithm running locally on the NUC of the LRM, parallel to autonomous exploration. This enables it to survey and create a 3D map of the environment while detecting objects on interest, and storing their position for future object grasping. All in all, this work aims to mature the LRM project and contribute to the state-of-the-art in autonomous exploration techniques for open-space rovers and the growing pursuit to develop evermore accessible, reliable, and modular robotic systems.

Figure 1.1.: Picture of the Lunar Rover Mini 1, on the right with the robotic arm, and the Lunar Rover Mini 2, on the left.

This thesis is structured as follows. Chapter 2 reviews the state-of-the-art in autonomous exploration techniques, with a particular focus on open-source approaches. It also discusses the capabilities of the LRM and similar rovers, as well as behavior modeling using state machines, and object detection and 3D localization.

In Chapter 3, the research proposal derived from the preceding literature review is presented, along with its corresponding research objective and research question. Chapter 4 focuses on the theoretical background, presenting the key concepts and relevant theories that underpin the development of autonomous exploration strategies and object detection. Besides, it covers the foundational capabilities required *a priori* in a robotic platform to implement an exploration algorithm.

Further, Chapter 5 defines a functional framework based on low-level state machines that are responsible for detecting, segmenting, and filtering candidate frontier centroids. The exploration algorithm, designed in Chapter 6, then selects the best centroid to navigate to based on a defined set of criteria. Furthermore, Chapter 8 validates the proposed approach through a series of experiments, with the resulting data subsequently post processed and evaluated.

At last, Chapter 9 concludes this thesis work by summarizing the key findings, revisiting the research questions, and providing recommendations for future work.

2

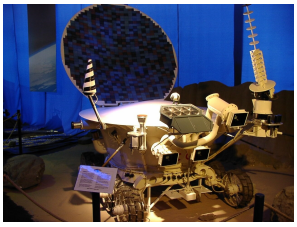
Literature Review

This literature review is structured as follows. Section 2.1 describes briefly the development of space rovers leading up to the LRM, including low-cost open-source rover platforms. Further, Section 2.2 depicts the capabilities that have been implemented on the LRM over the years, highlighting the major contributions to the project. Section 2.3 presents the state-of-the-art in autonomous exploration techniques, covering both frontier and sampling-based methods, as well as hybrid approaches and of those, the ones that have been open-sourced and their characteristics. It also addresses miscellaneous methods and emerging trends in the field. Further, Section 2.4 centers on behavior modeling with state machines in autonomous systems, referencing RAFCON, DLR's software tool to develop autonomous tasks through HCSMs. Furthermore, Section 2.5 speaks to object detection algorithms over the years and subsequent development of pipelines for the localization on objects of interest in 3D space for robotic applications, referencing YOLO as the most recognized and influential framework in achieving real-time, high-accuracy detection and tracking performance. Besides, Section 2.6 provides a summary of the reviewed literature. Finally, after reviewing the literature, Section 2.7 discusses the findings in order to identify the research gap and formulate the research questions that will drive the direction of this study.

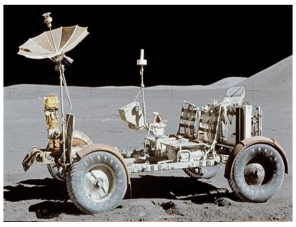
2.1. History of Space Rovers

Space rovers developed throughout history for various mission profiles can operate in multiple modes, including teleoperation, crew or payload transport, and autonomous exploration. Depending on the mission and configuration, a single rover may perform more than one of these functions. The first teleoperated rovers were the Soviet Union's Lunokhods. Although Lunokhod 0 crashed during a launch failure, the Lunokhod 1 and Lunokhod 2, Figure 2.1a, similar in design, successfully landed on the Moon, marking the first deployment of a remotely controlled robotic platform on a celestial body [24]. Crewed exploration vehicles, such as NASA's Lunar Roving Vehicle, Figure 2.1b, used during the Apollo 15, 16, and 17 missions in 1971-72 were designed to transport astronauts on the Moon while carrying

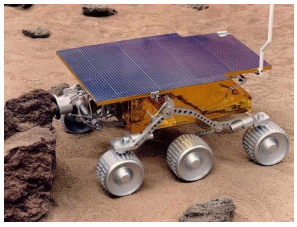
payload. They were manned vehicles, but could be teleoperated in case the crew was impaired [25]. Since then, autonomous navigation capabilities have been combined with space rovers to achieve considerable breakthroughs in autonomous planetary exploration. The Mars Pathfinder lander is often considered the first mission to feature real autonomous navigation capabilities. It landed in Mars in 1997 carrying the rover Sojourner [26]. Sojourner, shown in Figure 2.1c, carried three cameras including a front stereo vision camera and a back color camera, which allowed the rover to steer autonomously using its wheel odometry and IMUs to generate commanded goal locations.



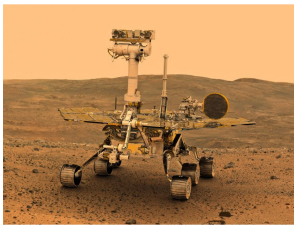
(a) Lunokhod 2 rover.



(b) Lunar Roving Vehicle.

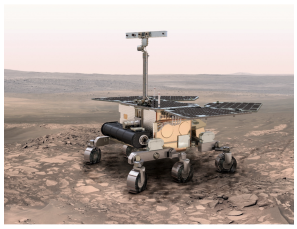


(c) Sojourner rover.



(d) Opportunity rover.

(e) Perseverance rover.



(f) ExoMars rover.

Figure 2.1.: Historical and contemporary planetary rovers. *[Credits: Wikipedia, JPL/NASA, ESA.]*

Following Sojourner, further Mars exploration missions have been equipped with increasingly advanced autonomous exploration space rovers. Opportunity, Figure 2.1d, launched in 2003 with the same six-wheeled rocker-bogie as its predecessor, a design choice that would become NASA's standard locomotion architecture [27]. These were designed to be highly autonomous, including autonomous instrument deployment and sample retrieval. This was followed by the Curiosity rover, in 2012, as part of NASA's Mars Science Laboratory mission. The rover has been exploring Mars with the goal of investigating if it can sustain life [28]. Compared to its predecessors, Curiosity featured not only general improvements in autonomous navigation capabilities but also an increased size and ability to carry more payload. Next came the Perseverance rover, Figure 2.1e, launched in 2020 toward Jezero Crater on Mars. This rover represents a next-generation improvement with more sophisticated scientific instruments, with the aim of understanding Mars' morphological geology. Furthermore, it was equipped with Ingenuity, the first aircraft

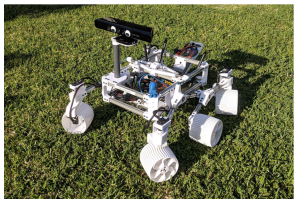
to flight on another planet [29]. Parallel to the development of Perseverance, the ExoMars rover, Figure 2.1f, was being designed in a collaboration between ESA and Roscosmos, targeted to search and find well-preserved organic material and other biomarkers [30]. The launch was originally scheduled for a 2022, but has been postponed to 2028.

2.1.1. Low-Cost Open-Source Rover Platforms

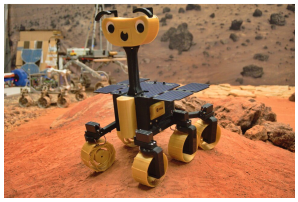
With this groundwork in place, more open-source, affordable, and smaller robotic platforms built with commercial off-the-shelf components are increasingly being developed to approach the capabilities of more complex space rovers. This is motivated by the fact that these systems take many years to develop and are extremely expensive and mission specific, making them inaccessible as a learning platform for students and researchers.

DLR was one of the pioneers in this field, by launching ASURO as early as 2003, selling over 30000 units at a price of 50 €. This mobile robot included light sensors and light barriers for odometry and line tracking, allowing the ASURO to detect and avoid obstacles. More recently, NASA JPL developed the Open Source Rover (OSR), launched in 2019, Figure 2.2a. This remote-controlled rover is equipped with the same six-wheeled rocker-bogie as NASA's complex space rovers, achieving a speed of 1.75 m/s, with a minimum autonomy of 5 hours. The rover is priced at 2160 € and can feature a LED head display. The OBC is based on a Raspberry Pi, running a Linux and Python software stack with ROS as the middleware framework. Nonetheless, the OSR does not include autonomous navigation capabilities, leaving it to the user to mount additional hardware on the rover, such as a stereo vision camera for visual VO or an IMU for SLAM sensor fusion. In line with this trend, the Sawppy rover project was created, Figure 2.2b.

(a) Open Source Rover.



(b) Sawppy rover.



(c) ExoMy rover.

Figure 2.2.: Low-cost, open-source rover platforms designed for educational and research purposes. *[Credits: JPL/NASA, Sawppy, ESA.]*

The Sawppy rover featured the same six-wheeled rocker-bogie locomotion system, but brought the total cost down to 430 €. This was possible because of smaller overall dimensions and using a combination of aluminum and 3D-printed parts, compared to the frame of the OSR. Moreover, this rover uses hobby servo motors, when compared to the individual DC motors of the OSR. Thereafter, in 2020, ESA launched the ExoMy rover, Figure 2.2c. Similarly to the OSR, it runs

on a Raspberry Pi, with ROS as the middleware framework, but it incorporates a Raspberry Pi Camera as well. The suspension system is the same as the aforementioned rovers, achieving a maximum speed of 0.23 m/s. This option was the most affordable at just 250 €, thanks a fully 3D-printed design.

The Lunar Rover Mini project, born in 2015, adds to this field of modular open-source rovers. Influenced by the ExoMars rover locomotion system, where DLR played a crucial role in its development [31], and using the same high-level software of DLR's Lightweight Rover Unit [17], the LRM project was created with the goal of providing a low-cost, accessible testing platform for educational and research purposes [32]. Thanks to its modular components and 3D-printed body parts, as well as its small size 36x26x39 cm, it is easy to carry and assemble. The LRM is equipped with a servo-controlled pan-tilt unit with an Intel RealSense Depth Camera D435i and a 6-DOF robotic arm with an end effector, which allows for autonomous navigation, and arm manipulation, respectively. This has the advantage of offering these capabilities out of the box, rather than requiring additional costly upgrades.

Moreover, it is powered by an Intel NUC, rather than a less powerful Raspberry Pi. Its estimated cost is around 2000 €. In addition, the LRM benefits from an open-source software framework based on ROS, Simulink, and Python. The core capabilities of the aforementioned rovers are summarized in Table A.1.

2.2. Lunar Rover Mini

The LRM was created as a project to serve as an open-source, low-cost, robotic platform for educational purposes. Since the start of the project, students have contributed to it by continuing to increase the LRM capabilities. The project is guided by the aim of making robotic platforms more accessible to researchers and students. The LRM is designed for deploying modular software packages and is built using off-the-shelf hardware components.

Recently, the effort has been focused on improving the autonomous navigation capabilities of the rover. Ricardez Ortiogsa [33] integrated the LRM with an autonomous navigation algorithm and corresponding data pipeline, apart from validating the fully functional system integration. Next, Sam Bekkers [34] implemented a visual-inertial SLAM algorithm that combines data from stereo camera images and onboard IMU measurements to improve the accuracy of tracking the rover's movement over time. Also the work of Marco Conenna [35] improved the accuracy of the 6DOF robotic arm of the rover, through an elasto-kinematic calibration.

2.2.1. System Architecture Overview

The LRM is considered a small ground-based rover of dimensions 36x26x39 cm, with a 3D-printed acrylonitrile styrene acrylate (ASA) body frame, supported by a passive '3-bogie' suspension system for a total of 6 wheels. This configuration, inspired by the ExoMars Rover locomotion system [36], keeps all wheels in contact with the ground and the main body frame stable, when moving through uneven

terrain. The motivation of using ASA is based on its lightweight structure and ability to withstand higher temperature variations and mechanical stress compared to traditional fused deposition modeling material such as polylactic acid, as well as the fact it can be reproduced using 3D printers at home.

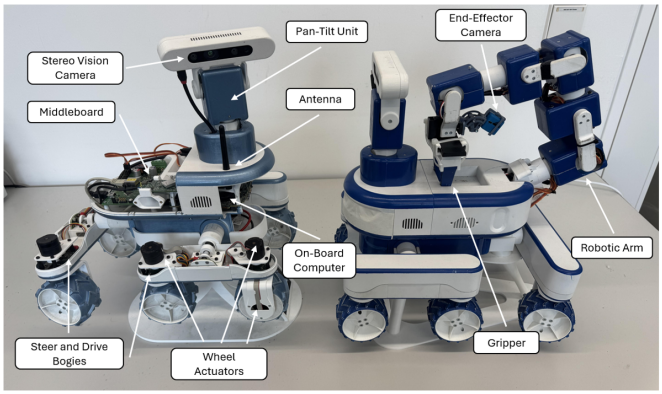


Figure 2.3.: LRM's key components overview, including mechanical and electronic systems.

Its main body frame contains the OBC, middleboard PCB, and wireless antenna. The antenna connects to a ground station that provides VNC access to the Intel NUC. Besides, each wheel is powered by its own servo motor, meaning each bogie has two servo motors, allowing for a 6-wheel differential steering. On top of that, each wheel is driven by a servo motor placed inside it. Thus, in total, there are 12 servo motors for wheel movement and the 2 servo motors for moving the pan-tilt unit through Pulse Width Modulation (PWM) signals. Figure 2.3 presents the rover's key components.

2.2.2. Onboard Electronics and Communication

An overview of LRM's onboard electronics is presented in Figure 2.4. The schematic illustrates the system's communication architecture, showing how the OBC connects with other components. The OBC is an Intel NUC with an i7 processor running an Open-Suse Leap 15.4 OS. On top of that, USB connections connect the Intel NUC to both the pan-tilt unit (via USB-C) and the body PCB (via USB Mini-B). The NUC also uses an WiFi antenna to link with external devices, although there is the possibility of a direct Ethernet connection. Additionally, RS-485, a reliable, differential serial protocol, is used by the middleboard to connect to the six Bogie PCBs. The entire system is powered by a 14.9 V LiPo battery.

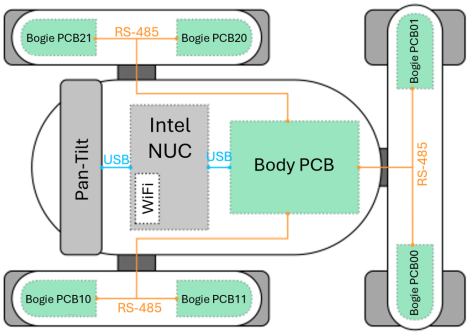


Figure 2.4.: Overview of the onboard electronics and module communication.

The OBC is responsible for running the autonomous navigation algorithm, interfacing directly with the stereo vision camera and determining the steering angles and driving speeds of the wheels. Too, the body PCB also handles communication between the OBC and all actuators.

2.2.3. Software Architecture

As previously mentioned, the LRM shares the same high-level software infrastructure of DLR's LRU. This allows to exchange and reuse software packages from different robotic platforms from the Robotics and Mechatronics (RM) institute of DLR. All software packages are deployed through the Links and Nodes (LN) Manager. The LN Manager is an open-source process manager tool developed by DLR-RM for deploying a hierarchical set of processes on robotic systems [37].

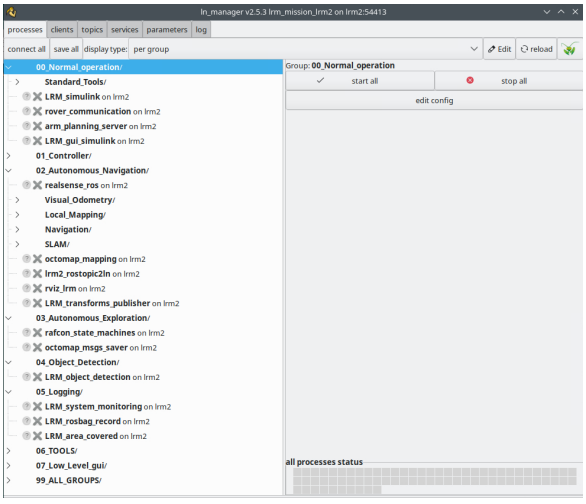


Figure 2.5.: Links and Nodes Manager of the LRM.

As it can be seen from Figure 2.5, the LN Manager allows to monitor the status of each process. Thus, the LN Manager manages different processes, regardless of whether they are developed in ROS, MATLAB, Python or C++. Regarding the LRM, ROS is used for all tasks related to perception, navigation, and mapping. Hence, two middleware framework are identified on the LRM, the LN Manager and ROS.

2.2.4. Rover Operations

The LRM features a low-level and a high-level GUI. The high-level GUI allows to steer and drive the LRM, rotate the pan-tilt unit and specify the target position for the robotic arm's end-effector. Also, it enables switching between different control locomotion modes, including the high-level GUI, a gamepad controller, and the autonomous navigation stack. The different inputs from these control modes are converted to a universal velocity command vector passed through a Simulink model that determines the behavior of the rover's steering and driving servo motors.

There are three different driving modes, defined by a combination of linear velocities along the x and y direction, and angular velocity around the z-axis. They are *Ackermann Mode* (car-like turning with sharper angles on the inner wheels), *Rotation Mode* (rotation in place), and *Crabwalk Mode* (sideways motion with all wheels aligned in the same direction). The conditions for each driving mode are depicted in Figure 2.6.

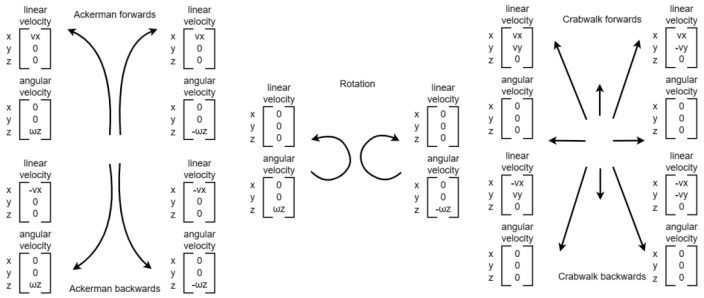


Figure 2.6.: LRM different driving mode conditions [33].

Moreover, the low-level GUI is used for calibrating the wheels zero position, and tuning the PID control values for both steering and driving, including both the inner and outer control loops.

2.2.5. Perception

The LRM uses a feature-based approach to VO, which involves detecting and tracking specific keypoints in the environment. It uses a Good Features to Track detector [38] to extract features from the stereo vision camera images and and describes the local environment around these keypoints using Binary Robust Independent Elementary Features (BRIEF) [39]. Then, the features between

consecutive images are matched to estimate the motion of the rover. Figure 2.7 depicts this process of extracting features from the environment. This is implemented in the LRM within the ROS `rtabmap_odom`[1] node.

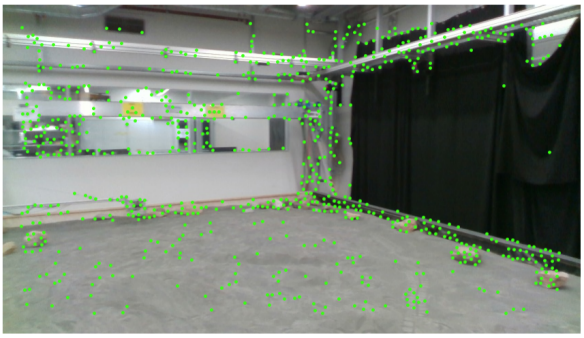


Figure 2.7.: GFTT algorithm with detected visual odometry keypoint captured in one camera frame [34].

The LRM is equipped with autonomous navigation capabilities built upon a visual SLAM pipeline using the Intel RealSense D435i RGB-D camera, which combines stereo vision and an IMU. The open-source RTAB-Map module, used for solving the SLAM problem [40], integrated with the GTSAM optimization backend, updates a factor graph in real time to generate a consistent 3D map [41]. This is implemented in the LRM within the ROS `rtabmap_slam`[2] node.

This map is shared with the ROS-based navigation stack, which includes a local planner with a path-planning algorithm, a motion controller for drive mode and specific velocity commands, and a global planner to correctly estimate the rover's pose in a global map frame. This allows to create a real time 3D point cloud representation of the environment, and an occupancy grid, which identifies occupied, free, or unexplored space, illustrated in Figure 2.8 and Figure 2.9, respectively. More specifically, the local planner's path planning algorithm employs the classic A* search method to generate a short, collision-free path within the local costmap [42]. In Figure 2.9, grey grid cells (■) are considered free, black grid cells (■) are occupied, and green ones are unknown (■).

[1] Information about the `rtabmap_odom` ROS node can be found here.
[2] Information about the `rtabmap_slam` ROS node can be found here.

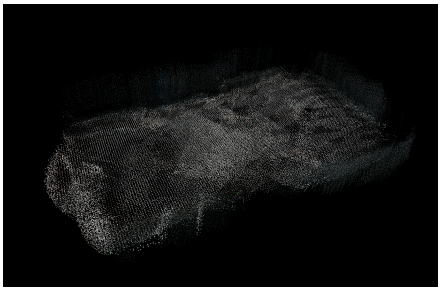


Figure 2.8.: Example of 3D point cloud generated.

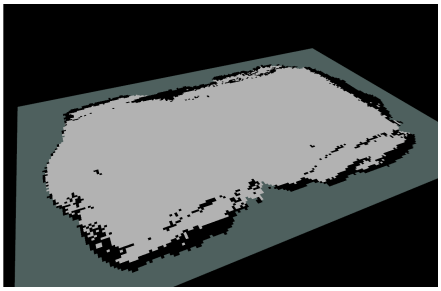


Figure 2.9.: Example of 2D occupancy grid generated.

2.3. Autonomous Exploration Techniques

Additionally, this section deals with different autonomous exploration techniques. More specifically, Subsection 2.3.1 examines frontier-based exploration algorithms, Subsection 2.3.2 looks into sampling-based exploration algorithms, Subsection 2.3.3 covers hybrid frontier-based and sampling-based exploration algorithms, and Subsection 2.3.4 briefly mentions emerging trends and other miscellaneous methods when it comes to autonomous exploration techniques. This is followed by Subsection 2.3.5, which discusses the state-of-the art in open-source ready-to-use exploration strategies.

2.3.1. Frontier-Based Exploration

Frontier-based exploration was first introduced by Yamauchi in 1979 [8]. It involves detecting and using frontiers, defined as the boundaries between known and unknown areas, to move through an unknown environment while building a map that can be used for subsequent navigation. Thus, detecting frontiers represents the central process in frontier-based exploration algorithms, meaning that variations across methods lie on how frontiers are selected [12, 43, 44]. When moving towards a new frontier, either part or all of the previously unknown space will be mapped. If only part of the space is mapped, a new frontier will emerge, separating known and unknown areas.

The first, simplest method, consisted in detecting candidate frontiers from an occupancy grid map [8]. This is possible since occupancy grid maps contain cells that can be represented by one of three states: cells with low occupancy probability, free space, cell with unknown occupancy probability, unknown space, and cells with high occupancy probability, occupied space. Then, candidate frontiers are clustered into frontier clusters based on proximity, and the centroid of each frontier cluster is computed, where frontier clusters below a minimum size threshold or unreachable based on the arrangement of high occupancy probability grid cells, are disregarded. By knowing the cartesian coordinates of each centroid in space, as well as the robot's pose, the euclidean distance from the robot to the centroid is computed.

Finally, the robot can select the closest frontier and navigate towards it.

Topiwala et al. [11] highlighted that this frontier detection algorithm is rooted in Breadth-First Search (BFS) from graph theory. When the robot fails to reach a goal after some time, that location is marked as an inaccessible frontier and the process of moving towards the closest, accessible, unvisited frontier is triggered. However, one of the downsides of this strategy is that it requires to go through the entire occupancy grid map at every run of the algorithm between moving through different frontier centroids.

To overcome this limitation, an improved version of this methods called Wavefront Frontier Detector (WFD) was introduced [11], which limits scanning to the known regions of the occupancy grid, thereby reducing computational load. This work showed that total travel time and distance covered are lower using the WFD algorithm, when compared to a human tele-operating the same robot to generate a 3D map of the surroundings. Hence, it also contributes to the ever-increasing certainty on the importance of autonomous exploration systems.

As well, in contrast to BFS-based frontier exploration, [45] proposed a recursive Depth-First Search (DFS) strategy, also rooted in graph theory, which incrementally builds a tree structure from sensor data without scanning the entire grid. Unlike BFS, where all nodes are explored on the same level before going into the next depth level, DFS follow a node path as deep as possible before backtracking.

The aforementioned strategies can all be considered utility-based if the only parameter of the utility function to be minimized is distance between robot and frontier. Nonetheless, selecting frontiers solely based on distance overlooks other parameters, such as information gain, and environmental complexity, which could result in redundant exploration, poor map scalability, and limited mapping efficiency. This being the case, a utility-based goal selection, where the score of each candidate goal position is determined by its traveling cost and expected information gain, weighted by a scaling factor in an utility function, was introduced by González-Baños and Latombe [46]. Traveling cost can be seen as the time between the initial and final pose of the robot, which can be inferred by the distance between the two points, and maximum linear and angular velocity. Information gain stems from the concept of entropy, which measures uncertainty about a system.

In the context of a robot exploring an environment, entropy quantifies uncertainty about the map or the state of a grid cell given the probability of occupancy of each. The first entropy-based utility function was introduced by Bourgault et al. [47] in 2002, using a convex combination of the map and the robot's pose entropies. Overtime, most state-of-the-art utility functions are based on Shannon's entropy, a special case of Rényi's entropy [48], which quantifies the average uncertainty or information content of a random variable as the sum of the negative probabilities of each outcome multiplied by the logarithm of those probabilities [49].

In fact, most of newly implemented frontier-based strategies in the literature can all be traced back to a more or less complex utility function that balances traveling cost and expected information gain. For instance, Dai et al. [50] implemented a fast frontier-based information-driven algorithm for a Micro Aerial Vehicle (MAV), where the utility function is a ration between expected information gain and travel time.

Because most of these strategies are tested in simulated environments, the literature shows some effort into comparing experimentally the performance of different frontier-based exploration strategies [10, 12] or through theoretical trade-off comparative studies [14].

Nevertheless, purely theoretical trade-off comparative studies are more qualitative than quantitative in showcasing results, and the ones that have been experimentally compared are rather simple strategies, given the effort of implementation. Some extensions of the original two parameters of the utility function are also present in the literature. For instance, Cogent Frontier Exploration (CFE) which chooses new frontier position goals based on four parameters-cluster factor, based on the size of a frontier cell cluster, distance factor, related on the Eucledian distance, clearance factor, based on the accessibility of a frontier cell, and unreachable-point factor, that gives a penalty to frontier cells that cannot be reached-through the information of the 2D occupancy grid map [51].

In addition, Gomez et al. added a semantic utility term, responsible for giving a semantic importance to transit areas, giving higher priority to areas that are gateways to unexplored regions, even if they are small in size [52]. Too, the work of Li et al. extended the parameters of the utility function to account for the overlap between the new information from the partial map at every observation point, and the current global map. This is particularly important on exploration strategies that heavily rely on accurate localization [53].

It is also important to mention that even though only one robot has been considered for a particular strategy so far, a parallel set of literature is also dedicated to expanding these frontier-based exploration strategies for swarms of robots [9, 54, 55]. Nonetheless, the more recent work of Kuckling et al. provides evidence that multi-robot systems experience a critical point in system size beyond which performance plateaus or starts decreasing [56]. Too, that the vast majority of such exploration strategies are implemented using ROS as the middleware. This is because frontier detection relies on a generated 3D point cloud from the environment, and respective 2D occupancy grid, which is well integrated within a ROS framework.

2.3.2. Sampling-Based Exploration

An alternative approach to frontier-based exploration is sampling-based exploration, which relies on generating random samples in the environment. The simplest example of a random selection mechanism is known as a random walk, where the robot selects a random direction and moves towards it until a new one is selected. Examples of random walk variants include correlated random walks [57], Lévy walks [58], Lévy taxis [59], and ballistic motion, which vary in step lenght and turning angle over time. In contrast, pure random action selection may be very inefficient. Thus, heuristic criteria can be employed on the randomly selected set of points to determine the next position goal. With regards to sampling-based methods, the state-of-the-art approach is receding horizon (RH) [60, 61, 62]. A RH approach means the robot repeatedly plans and executes short-horizon paths toward the best

exploration goal, replanning whenever the map updates, so it remains adaptive and efficient in unknown environments. In other words, the robot moves to the first node along the path and once it reaches this node, a new planning tree is generated, using the best branch from the previous tree as the starting point.

Moreover, LaValle pointed out that this method offers simplicity and guarantees completeness, ensuring that any sequence of actions will eventually be executed and that a solution will always be found, if one exists [63]. Additionally, sampling-based exploration enables computing different information gains directly from the map, without using a fixed definition of what is considered as useful information.

The underlying most adopted method in the literature through which randoms sample points are generated for sampling-based exploration is Rapid-exploring Random Tree (RRT), initially introduced in 1998 as a randomized data structure tool for path planning problems [64]. It is a method that incrementally builds a space-filling tree by sampling random points in the configuration space and connecting them to the nearest existing node in the tree. When applied to sampling-based exploration, the RRT algorithm incrementally grows a tree into unknown regions of the environment, effectively guiding the robot toward areas that have not yet been observed. Each new node in the tree represents a potential exploration goal, and the connections between nodes form feasible paths that the robot can follow. Of course, each node that represents a potential exploration goal can also be evaluated through an utility function, as demonstrated by Bircher et al. using a RH-NBV (Receding Horizon Next Best View) planner [60].

Nonetheless, this approach, does not scale well for large-scale environments. Thus, in this field, focus has been given to proposing RRT variants that improve exploration efficiency, since the core RRT algorithm provides a flexible and scalable framework for incrementally covering unknown space, naturally balancing exploration of unvisited regions with feasible path generation [64]. Aside from RRT, another pioneer sampling-based motion planner comes from Probabilistic RoadMaps (PRM). In contrast to RRT, PRM build a global roadmap of the free space by randomly sampling nodes and connecting them into a graph [65]. PRM is well suited for multi-query planning, since once the roadmap is built, many start–goal queries can be answered efficiently. RRT, on the other hand, is more effective for single-query or online exploration tasks, where rapid incremental growth toward unexplored regions is preferred. The work of Karaman and Frazzoli proposed improved versions of both methods, RRT* and PRM*, which extend the original algorithms with asymptotic optimality guarantees.

Unlike classical RRT and PRM, which are only probabilistically complete and may converge to suboptimal solutions, RRT* and PRM* ensure that as the number of samples grows, the cost of the solution converges almost surely to the optimal value, all while maintaining a computational complexity within a constant factor of the original algorithms [66].

In recent literature, a large number of papers have presented the state-of-the-art when it comes to variations of the original RRT method, with respective advantages and disadvantages, that could potentially be used as part of sampling-based exploration [67, 68]. Moreover, some scholars have proposed autonomous

exploration solutions that build on top of these different sampling methods.

One such variation is Sensor-based Random Trees (SRT) which samples randomly within Local Safe Regions (LSR) defined by sensing the environment [69]. The SRT method addresses the limitations of RRT by adapting step lengths to obstacle density, avoiding unnecessary collision checks, and using a depth-first, backtracking exploration strategy better suited for sensor-based environments. In the authors' work, Oriolo et al. propose two different SRT algorithms: SRT-Ball, which uses a circular, fixed-radius LSR, and SRT-Star, which uses an adaptative star-shaped LSR according to sensor data. Thus, while SRT-Ball prioritizes safety, it performs poorly in cluttered environments. In contrast, SRT-Star offers better coverage and more efficient navigation in narrow spaces.

Furthermore, Xu et al. proposed a Dynamic Exploration Planner (DEP) based on PRM, motivated by the fact that most sampling-based methods fail to efficiently reuse sampled information from previous planning iterations, resulting in redundant computation and longer exploration times [70]. Their approach incrementally builds the roadmap by adding nodes that are evenly distributed within the explored region, enabling the selection of informative viewpoints while reducing unnecessary sampling for a UAV. Also, the literature suggests that even though a lot of effort has been dedicated into proposing improved variants of the RRT algorithm, such as Informed RRT* [71], Goal-biased Bidirectional RRT [72], RRT-Connect [73], among others, most implementation of sampling-based methods for autonomous exploration, especially concerning more recent literature, are coupled with some frontier-based method [61, 74, 75, 76], expounded upon below.

2.3.3. Hybrid Frontier-Based and Sampling-Based Exploration

Hybrid exploration methods combine frontier-based strategies with sampling-based approaches such as RRT, leveraging the complementary strengths of both methods. Frontier-based exploration is best at identifying and guiding the robot toward informative boundary regions between known and unknown space. However, it suffers from inefficiency when navigating large or complex environments, as the robot may need to backtrack repeatedly between distant frontiers. On the other hand, sampling-based methods, like RRT and its variants, provide flexible and feasible path planning generation, but may lack the global perspective needed for complete coverage. By integrating the global awareness of frontier methods with the adaptability and scalability of sampling-based planning, hybrid approaches can overcome these individual limitations and enable robots to explore unknown environments more effectively.

Freda and Oriolo [74] extended the SRT method presented before, by introducing a frontier-aware variant, Frontier-Based SRT. While the original SRT builds a roadmap by sampling within sensor-defined LSRs without distinguishing obstacles from unexplored areas, FB-SRT incorporates frontiers to improve exploration efficiency. In this method, the robot's LSR boundary is divided into three types of regions: obstacle regions, free regions, and frontier regions. By identifying frontier regions, FB-SRT biases the robot's movement toward unexplored regions, while still

maintaining the random sampling nature of the SBE process.

Selin et al. [61] proposed a hybrid exploration strategy that combines Frontier Exploration as a global planner with RH-NBVP for local exploration, enhanced by fast information gain estimation and caching to improve efficiency in large environments. Interestingly, hybrid methods are applied more to UAVs or MAVs rather than rovers in the literature. For instance, Dai et al. [50] introduced a hybrid exploration framework for MAVs that leverages octree-based occupancy mapping to implicitly cluster frontier voxels, samples candidate next-views directly from frontiers, and evaluates them using a utility function combining entropy reduction via sparse ray casting with travel cost.

As well, Wang et al. [77] proposed a systematic 3-D UAV exploration framework that incrementally builds a roadmap to capture the environment's topology, enabling efficient next-best-view evaluation through information gain and cost-to-go estimation, while a potential field-based local planner guides the UAV toward information-rich regions, achieving higher exploration efficiency than existing methods. Ning et al. [76] introduced the Hybrid Multi-Strategy Rapidly-Exploring Random Tree (HMS-RRT). This approach combines sampling-based exploration with frontier detection by leveraging a sub-region sampling strategy and a greedy frontier-based exploration strategy together with Voronoi diagrams to divide the environment into sub-regions. Results in the authors work showed that this strategy improve the efficiency and reliability of exploration in both travel time and distance covered.

2.3.4. Emerging Trends and Miscellaneous Methods

Recent literature goes beyond classical frontier-based and sampling-based strategies. Notably, several studies focus on innovative approaches to enhance both exploration efficiency and decision-making under uncertainty. Long et al. [75] proposed the Hierarchical Planning based on Hybrid Frontier Sampling method (HPHS), which combines frontier sampling from LiDAR data with a hierarchical planning strategy. This approach divides the environment into sub-regions, optimizing the order where each sub-region is accessed according to a maximum score.

In the field of knowledge of learning-based techniques, Leong [78] integrated reinforcement learning with frontier-based exploration. By leveraging a reward-based system, the proposed algorithm enables robots to learn optimal exploration path, increasing the accuracy of visual SLAM and addressing challenges associated with multiple frontiers of similar distances. On top of that, Cai et al. [79] introduced an enhanced hierarchical planning framework for multi-robot systems. This approach combines the efficiency of frontier-based methods with reinforcement learning. By employing a multi-graph neural network and policy-based reinforcement learning, the framework effectively improves exploration performance in complex environments.

Further, Sun et al. [80] developed FrontierNet, a learning-based model that uses 2D visual cues from RGB images to detect frontiers and predict their information

gain. This image-only frontier-based exploration system addresses limitations of traditional 3D map-dependent methods, achieving notable improvements in early-stage exploration efficiency. What is more, Zhang et al. [81] put forward Local Constrained Sampling and Frontier Prioritization RRT, a hybrid exploration algorithm that integrates RRT with frontier detection. Instead of relying on image-based methods, it identifies frontiers by recording the top nodes of the RRT. By combining this with a local constrained sampling strategy and prioritizing frontiers based on their utility, the method significantly increases exploration efficiency in large-scale environments.

Methods that deviate from the norm include a multi-objective optimization approach presented by Amigoni and Gallo [9]. In this approach, the system evaluates each candidate observation point based on its individual values for traveling cost, information gain, and localization overlap. It then identifies the set of Pareto-optimal candidates—those for which no other candidate is better in all three aspects. Besides, Dumitru et al. [82] presents a navigation system for autonomous mobile robots operating in unknown environments based on the artificial potential field method. In this approach, obstacles are associated with repulsive fields and target points with attractive fields, both calculated based on distance and relative position. The robot's movement is determined by the resulting vector from the combination of these fields.

These methods are very complex to be applied to modular, built with off-the-shelf components, robotic platforms in a reliable way, but are relevant to mention given how many different ideas have been presented in this field of autonomous exploration techniques. Also, the fact that more comprehensive studies need to be carried out to prove the efficiency of such strategies when compared to more standard ones. However, this is difficult to achieve because proper comparison between methods means that all shall be implemented in a single system, whereas these are system-specific.

2.3.5. Open-Source Exploration Strategies

From the previous subsections, it is evident that most autonomous exploration techniques are system-specific, and even by consulting the appropriate references are difficult to replicate and implement on different robotic platforms. This being the case, some literature is specifically dedicated to providing ROS-based autonomous exploration techniques that can be shared and reused across different systems, thereby promoting reproducibility, easing integration, and enabling the research community to build upon common open-source frameworks. Such strategies are then presented to the scientific community through GitHub repositories, sometimes with complementary papers.

One of the difficulties associated with this is the fact that different systems need to share similarity between middleware frameworks, and a level of maturity where the robot has to perceive its environment, localize itself within in, and navigate effectively. In other words, enabling autonomous exploration depends on its ability to simultaneously solve mapping, localization, and motion planning tasks beforehand,

as depicted in Figure 2.10. On top of that, because an increase in the complexity of exploration strategies demands a system architecture more tailored to that particular strategy, makes it so they are rather simple, as explained below.

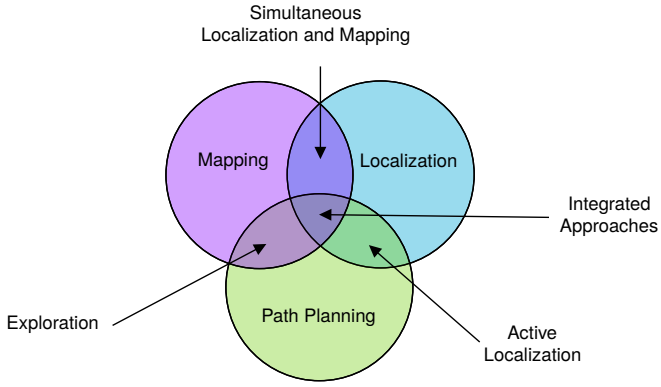


Figure 2.10.: Foundation capabilities for autonomous exploration, taken from Darmanin and Bugeja [83].

For instance, an Autonomous Explorer Node for Frontier Exploration[3] has been proposed as a ROS package that detects frontiers, leverages the ROS 2 Navigation (Nav2) framework—the standard navigation stack in ROS 2, which provides localization, mapping, costmaps, path planning to enable safe, collision-free navigation [84]-and chooses the closest unexplored frontier as the next position goal. A more complex open-source strategy was proposed by Umari and Mukhopadhyay [85]. This exploration strategy works by growing RRT tree until a point that lies in an unknown region of the global map. Then, after detecting possible frontiers, the revenue, R, of each frontier point, x_{f_p}, is computed by balancing navigation cost, N, and information gain I, through a scaling factor, λ-a utility function is employed.

Nonetheless, the navigation cost is just taken to be the euclidean distance between the current pose and the frontier point, overlooking the time it would take to get to the actual point given the presence of obstacles in the map or current orientation. In addition, the information gain is given by the number of unknown cells around a frontier point within a certain radius in the 2D occupancy grid, which fails to take into account the more reliable information coming in from the 3D point cloud. This information can be retrieved by representing the 3D map through an OctoMap [86]. Here, the 3D space is represented using voxels, 3D volumetric cubes that have an attached occupancy probability. The downside side it that OctoMaps require more memory and processing because they store 3D spatial information and need to manage hierarchical tree structures. Thus, updating and querying an OctoMap is computationally more intensive than a 2D grid, especially in large environments. Having an OctoMap also allows to compute entropy in each frontier point through ray casting, something that is not possible by just resorting to the 2D grid.

[3]Available: https://github.com/AniArka/Autonomous-Explorer-and-Mapper-ros2-r

2.4. Behavior Modeling with State Machines

Another important point to make is that these ROS frameworks are great for providing standardized interfaces for sensors, actuators, and navigation modules, but they often lack a user-friendly visual interface for monitoring and controlling the exploration process. To address this, Finite State Machines (FSMs) can be employed to structure and manage the robot's behaviors, allowing for clearer visualization of states, transitions, and decision-making during autonomous exploration. FSMs, pertaining to the broader field of behavior modeling, is a foundational aspect in the development of robotic systems for autonomous mobile robotic platforms. These systems are required to perform complex tasks—including perception, decision-making, and motion planning.

Behavior models provide a structured way of organizing and controlling the flow of robotic tasks into states. In the literature, behavior-based control architectures such as subsumption [87], have laid the foundation for reactive robot behavior, where complex actions arise as the result of the interaction of simpler ones. Further development allowed the introduction of Hierarchical Concurrent State Machines (HCSMs) [88], which combine the modularity of hierarchical state machines with the concurrency of parallel processes, enables the modeling of complex, real-time behaviors by allowing multiple tasks to operate simultaneously and interact in a structured way.

Variation of the original FSM concept are also present in the literature under the form of Extended Finite State Machines [89], Communicating Finite State Machines [90], or Probabilistic Finite State Machines [91].

When it comes to integrating FSMs for robotic tasks, more particularly autonomous exploration, there is low research in this domain. Steinbrink et al. [92] describe a state machine tailored for exploration and inspection tasks with a GUI designed for ROS, which comes already with some basic functionalities, such as boot state, idle state, teleoperation state, and waypoint following state. To enable this, three plugins must be implemented, each providing interfaces to arbitrary ROS packages. They are a calculate goal state, a mapping state, and a navigation state.

Nevertheless, as the authors state in their paper, such state machine has only been successfully demonstrated in 2D scenarios, with missing plugins and development for 3D environments. Another example is SMACH developed by Bohren and Cousins [93]. It is a ROS framework for designing and managing hierarchical finite state machines, allowing developers to structure robot behaviors, define states and transitions, and integrate them with ROS nodes for modular and reusable autonomous control through a GUI.

However, even though additions have been made that improve SMACH's capabilities over the years, it is not user-friendly, with poor visualization, the lack of a visual editor and efficient runtime monitoring. FlexBe [94] tried to fix some of the issues with SMACH, mainly a better visual interface with a visual editor. Some additions to FlexBe included integrating the ROS Navigation stack [22]. Even though this is a notable effort, it simply allows the user to have more control over decisions that are internal to the ROS Navigation plugin model, as fails to describe a broader approach for managing autonomous tasks, specifically autonomous exploration, in

this framework. This is a recurring theme, as in most of the contributions to either SMACH or FlexBe come from integration already existing ROS packages into the framework.

Other standalone project exist when it comes to ROS-based decision-making packages[4], but they lack the capabilities of the aforementioned ones.

Coupled with this, RAFCON (RMC Advanced Flow Control) is another open source software tool, which was developed by DLR, based on hierarchical and concurrent state machines, that facilitates the creation, management and execution of autonomous robotic tasks [18, 96]. RAFCON operates with a better GUI and visual editor than the aforementioned, that simplifies the design and management of such state machines. Its interface supports three basic states. They are execution states, hierarchy states and concurrency states. The software tool was validated during the SpaceBotCamp, where it was used to program complex autonomous tasks with more than 700 states, 1200 transitions and 8 hierarchy levels on DLR's LRU [97].

2.5. Object Detection and 3D Localization

Object detection plays a central role in autonomous robotic systems, as it enables the identification and localization of relevant objects within the robot's environment. This is relevant in many different fields within robotics, from autonomous exploration and manipulation, where robots must recognize and interact with specific objects for sample retrieval, to environmental monitoring and inspection, where detecting changes or anomalies is critical.

This capability is typically achieved on robotic platforms equipped with stereo vision cameras, which provide both RGB and depth information. By combining these data sources with the camera's intrinsic and extrinsic parameters, the system can accurately compute the 3D coordinates of detected objects within the environment if the pose of the rover's body frame and camera frame are known at the time the images were taken, as it will be explained in the theoretical background. Traditional object detection techniques were based on handcrafted features such as Scale-Invariant Feature Transform (SIFT) [98] and Speeded Up Robust Features (SURF) [99], often combined with classical machine learning classifiers like Support Vector Machines (SVMs). While effective under controlled conditions, these methods suffered from limited robustness to environmental variability, occlusion, and complex lighting.

Further developments and interest of deep learning and convolutional neural networks (CNNs) revolutionized object detection by enabling end-to-end feature learning directly from data. Early CNN-based detectors, such as R-CNN [100] and its successors Fast R-CNN [101], and Faster R-CNN [102], introduced region proposal networks and shared convolutional features, significantly improving detection speed and accuracy. Single-shot detectors like YOLO (You Only Look Once) [103], Single Shot MultiBox Detector (SSD) [104], and RetinaNet [105] further enhanced real-time

[4]One such example is the Decision Making ROS package or YASMIN [95].

performance, making them particularly suitable for robotic applications where computational efficiency is critical.

From these, YOLO has been widely regarded as the state-of-the-art solution for real time object detection due to its balance between speed and accuracy. The YOLO family of models treats object detection as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation, thus enabling high frame-rate inference suitable for onboard robotic computation.

The latest version, YOLOv11 [106], is a versatile computer vision model supporting object detection, image segmentation, image classification, pose estimation, and oriented bounding box (OBB) detection.

Numerous studies in the literature describe the implementation of YOLO-based object detection pipelines for robotic applications [107, 108, 109]. Even so, others demonstrate how autonomous exploration can be coupled with object of interest (OoI) search such as the work of Dang et al. [110]. In the domain of aerial robotics, the authors applied a sampling-based semantically-enhanced exploration strategy, where an utility function maximizes a gain that related to exploring unknown space, as well as a gain related to the resolution of detected objects of interest. This is particularly interesting because it goes beyond having the autonomous exploration and object detection running in parallel independently of each other, to a fully integrated approach, where exploration decisions are directly influenced by the presence and importance of OoI.

Another example is the work of H. Kim et al. [111], where a frontier-based exploration strategy is combined with object detection to dynamically segment a 2D map based on the robot's path and detected frontiers. This approach reduces the risk of collisions with 3D obstacles and improves exploration efficiency by prioritizing unexplored areas while ensuring safe navigation. Nevertheless, these strategies could not be easily replicated on other robotic platforms, as they are not open-source. There exists some open-source packages integrated with ROS to detect and localize OoI in a 3D environment [5].

However, these packages are primarily designed for real-time video inference and rely on systems with high computational capabilities, which limits their applicability. For instance, both rely on CUDA, NVIDIA's parallel computing platform and programming model, for accelerated inference, a capability often unavailable on resource-constrained robotic platforms equipped only with CPU-based systems such as Intel NUCs.

2.6. Summary

The reviewed literature provides a brief overview of the history and breakthroughs of space rovers, leading to the development of the LRM. Section 2.1 traces the evolution of space rovers from the early teleoperated systems, highlighting the

[5] Such examples are the Complex YOLO ROS 3D Object Detection and the ROS-Based-3D Detection Tracking

growing need for autonomous capabilities, which ultimately drove the development of fully autonomous space rovers.

The development of low-cost educational rover platforms is also explored, emphasizing how these innovations, combined with advancements in software architectures and off-the-shelf hardware components laid the foundation for the LRM. Designed as a small, low-cost rover for educational and research purposes, as presented in Section 2.2, the LRM incorporates state-of-the-art features such as autonomous navigation, a modular software stack, integrating ROS, Python, and LN Manager, and a functional 6DOF robotic arm.

Furthermore, Section 2.3 thoroughly examined the state-of-the-art in autonomous exploration techniques, with a particular focus on frontier-based, sampling-based, and hybrid strategies. As detailed in Subsection 2.3.1, frontier-based methods classify the limits between known and unknown space as frontiers and use them to guide exploration, often using techniques like utility-based goal selection based on travel cost and information gain. Alternatively, sampling-based methods, Subsection 2.3.2, use random samples to explore the environment, predominately through RRT, with several variants of this framework in the literature. Emerging trends in this domain focus on bridging the gap between the two strategies. One example of this is the HMS-RRT hybrid strategy, as shown in Subsection 2.3.3. Subsection 2.3.5 addresses the exploration strategies that have been open-sourced through ROS, with emphasis on their modularity and ease of integration between different robotic systems, but compromised strategy complexity. Too, that this is only applicable in systems that already have mapping, localization, and path planning capabilities.

Section 2.5 underscores behavior modeling with state machines as a critical component for enabling autonomy in robotic systems. Classical approaches like FSMs remain widely used for structuring reactive and goal-oriented behavior, although variants such as HFSMs and HCSMs extend FSMs to handle complexity, concurrency, and multi-agent coordination. Validated tools such as RAFCON, which leverages hierarchical and concurrent state machines, offer a scalable and visual environment for managing autonomous tasks.

Finally, Section 2.6 expounds on how object detection pipelines have evolved in robotics, from simply detecting an object in a RGB image, to using that information with a stereo vision camera and respective camera intrinsic and extrinsic parameters to infer the coordinates in 3D space of OoI. This effort is bolstered by the increasing breakthroughs in deep learning and CNNs with YOLO being the most widely used real-time object detection algorithm used in computer vision. It is pointed out that open source pipelines for 3D localization exist but are designed for systems with high computational capabilities relying on GPUs for accelerated inference.

2.7. Discussion

This discussion of the reviewed literature serves as the first step to move towards the research gap in the next chapter.

First, the in-depth look into the current capabilities of the LRM provides a clear foundation for developing an autonomous exploration algorithm. With its core

features, VO, SLAM, and a navigation stack, the LRM is well-equipped to support advanced exploration strategies in real-world environments, with its autonomous navigation capabilities on moving from point A to point B detecting and avoiding obstacles already demonstrated in previous work [33]. This is also motivated by the fact that the LRM is one of many mentioned examples of low-cost open-source rover for space robotics. When compared to its peers, the LRM is more complex by integrating a more powerful Intel NUC, an Intel RealSense Depth Camera D435i and a 6-DOF robotic arm, enabling autonomous navigation and arm manipulation, which none of the other have.

Among the reviewed exploration strategies, frontier-based exploration emerges as a promising baseline, particularly because it can take advantage of the already generated 2D occupancy grid to detect frontiers without introducing redundant exploration inherent to sampling-based exploration such as RRT methods. Also, the navigation stack developed in-house by DLR incorporates a path planning algorithm with obstacle avoidance capabilities, wherein the local planner employs the classic A* search method to compute a short, collision-free trajectory within a local costmap derived from elevation maps generated by a local mapping node, rather than utilizing an RRT-based approach. However, existing open-source exploration strategies lack complexity, relying on simply moving to the closest frontier or using just the information on the 2D occupancy grid, rather than the 3D point cloud, to compute information gain.

From the reviewed literature, it can be concluded that there is an ongoing challenge in balancing the complexity of these implementations with the goal of making them plug-and-play.

Several improvements have been made to classical frontier-based exploration, most notably the introduction of utility functions that balance travel cost and expected information gain. Travel cost is typically measured as the time or distance to a frontier, while information gain is quantified as the entropy around a frontier point using ray casting. Implementing these functions on low-cost rovers like the LRM remains challenging, and few studies assess their feasibility on comparable platforms. Utility-based goal selection, successfully applied on high-performance systems such as DLR's LRU, may be constrained by the LRM's onboard Intel NUC processor.

Furthermore, most ROS-based exploration strategies lack a structured approach for autonomous task execution. Hierarchical and concurrent state machines can address this by managing robot behavior in real time. Existing models, however, are often limited frameworks with minimal prebuilt functionality. Evaluating dedicated tools like RAFCON, with a complex library of state machines, could improve usability, facilitate rapid development, and support scalable, open-source autonomous operations. A well-defined behavioral architecture would ensure that exploration tasks are efficiently controlled and easily extended to new capabilities.

Another important component that complements autonomous exploration is the robot's ability to perceive and interpret its environment through object detection and 3D localization. This capability enables estimating the position of OoI within the 3D environment, which is crucial for tasks such as autonomous sample retrieval,

manipulation, and environment understanding. Nonetheless, existing ROS-based 3D localization pipelines for object detection are for high-performance computing platforms, with no existing pipeline for optimized for deployment on low-cost, CPU-based robotic systems such as the LRM.

Moreover, coupling object detection with the behavior management framework discussed earlier could facilitate dynamic task allocation, allowing the rover to adapt its exploration strategy based on detected objects or regions of interest. This integration represents a key step toward achieving higher autonomy levels in low-cost, open-source robotic platforms like the LRM. This is also important because it builds the stage for autonomous grasping of OoI.

Taking everything into account, utility-based exploration emerges as a particularly promising direction for the LRM. While its effectiveness has been demonstrated in high-performance systems like DLR's LRU, further investigation is needed to assess its feasibility within the LRM's computational limitations and capabilities.

Also, integrating utility-based exploration within a behavior modeling framework such as RAFCON could provide a strong foundation for developing an open-source, scalable platform. Once the state machines for autonomous exploration are implemented, they can be extended to incorporate additional capabilities—most notably, object detection and 3D localization, which would enable detecting OoI during exploration. Coupled with the robotic arm, this would allow for autonomous manipulation and sample retrieval, further enhancing the rover's operational autonomy.

Ultimately, such integration would establish a modular architecture adaptable to other low-cost, open-source rover platforms previously discussed, which could similarly benefit from the implementation of this framework. This synthesis also reveals a research gap: the need for integrated open-source frameworks that combine the reliability of FSM-based behavior modeling, the adaptability of utility-based exploration methods, and the perception capabilities provided by object detection and 3D localization, all within the constraints of low-cost robotic platforms and a scalable framework such as RAFCON.

Bridging this gap could lead to more generalizable, scalable, and autonomous robotic systems capable of extended operations in unknown environments.

3 Research Proposal

Following the discussion about the literature in the previous section, this chapter outlines the research proposal by presenting a structured plan based on the research objective and questions derived from the identified research gap. Firstly, the research gap is identified in Section 3.1, highlighting the limitations in current knowledge and the motivation for this study. Secondly, Section 3.2 defines the research objective and the key questions the thesis aims to answer. Following this, Section 3.3 describes the approach and methods that will be employed to conduct the research. The expected outcomes are discussed in Section 3.4. At last, Section 3.5 presents the research plan.

3.1. Research Gap

The conducted literature review and subsequent discussion provided several key insights that helped define the underlying research gap addressed in this work. It is clear that while autonomous exploration remains a central research topic in mobile space robotics, and foundational capabilities such as VO, SLAM, and navigation stacks have seen significant advances, integrating these components into a state machine framework for fully autonomous exploration remains an undocumented research problem, particularly for systems built on low-cost, built with off-the-shelf hardware components when compared to high-end, custom-designed robotic platforms used in advanced research or space applications. Additionally, existing open-source autonomous exploration strategies lack complexity compared to closed-source ones, and frameworks for designing and managing HFSMs are often not user-friendly and provide limited functionality for autonomous exploration and object perception. Furthermore, while object detection and 3D localization have been widely studied in high-performance robotic systems, their integration with low-cost, open-source exploration platforms remains largely unexplored, limiting the rover's ability to perceive and interact with objects of interest autonomously.

Moreover, the LRM, as described in Section 2.2, has seen steady development to serve as a modular robotic platform for educational purposes. It supports foundational capabilities that provide a solid base for implementing autonomous

exploration in unknown environments. Although the LRM can navigate effectively from point A to point B, it still lacks an integrated exploration framework that autonomously selects goals and manages the flow of actions in a structured way. This limitation reflects a broader challenge across other low-cost robotic platforms, as similar concept rovers such as the OSR and the ExoMy rover also lack frameworks to guide exploration. Several strategies have been proposed in the literature. Classical frontier-based exploration, for instance, prioritizes navigation to the nearest unexplored boundary in the occupancy grid. While computationally efficient, this approach often results in suboptimal behavior, such as low-information trajectories. In response, researchers have introduced utility-based methods that balance information gain with travel cost. Utility-based strategies appear particularly well-suited for the LRM, but a concrete evaluation is needed to confirm their feasibility. A similar consideration applies to object detection, as it is necessary to assess whether real-time perception is possible on the LRM's Intel NUC CPU. This contrasts with much of the literature, which primarily focuses on advancing object detection on high-performance, GPU-accelerated systems rather than low-cost, CPU-based platforms like the LRM.

To be precise, there is currently no current integrated modular framework that simultaneously:

1. Is open-source and modular offering a reusable and extensible architecture that can be deployed and further developed across different robotic platforms with minimal effort.

2. Employs an autonomous exploration strategy on low-cost rovers built with off-the-shelf-components that is comparable, in terms of performance and exploration efficiency, to closed-source complex strategies implemented on high-end rovers.

3. Coordinates behavior through HFSMs with a comprehensive library of state machines for autonomous exploration.

4. Is able to detect and localize in 3D space OoI while conducting exploration.

Because such framework can be implemented and tested on LRM, this presents both a scientific and engineering gap.

3.2. Research Objectives and Questions

After identifying the research gap, namely, the need for an integrated, computationally efficient, and open-source modular framework for autonomous exploration with object detection and 3D localization on low-cost mobile robotic platforms, to be implemented and evaluated on the LRM, the research objective and subsequent research questions can be formulated. The overarching research objectives of this thesis are two. First,

To investigate how autonomous exploration strategies, coupled with 3D object localization, can be implemented within a state machine-based framework on a low-cost mobile robotic platform, ensuring an open-source, modular, and scalable design with performance comparable to high-end robotic systems.

Secondly,

To advance the autonomous capabilities of the Lunar Rover Mini by generating accurate maps of unknown environments, detecting objects of interest, and integrating these functionalities within a state machine framework that supports scalability and future expansion, with the overarching goal of enabling a fully autonomous mission pipeline, including object grasping.

The research questions, and respective sub-questions, that follow from the now defined research objectives, are:

1. How can autonomous exploration strategies be implemented within an open-source, scalable and modular state machine-based framework for mobile robotic platforms?
1.1 What are the computational and architectural limitations that limit the implementation and ready-to-use deployment of exploration strategies on different mobile robotic platforms?
1.2 How can frontier- and utility-based exploration algorithms be adapted for implementation within a state machine framework?

2. How can the performance of the designed exploration techniques and real time object detection be guaranteed and evaluated across varying operating conditions?
2.1 What criteria and benchmarks best evaluate the performance and robustness of autonomous exploration techniques?
2.2 How does the implemented exploration algorithm compare to state-of-the-art alternatives in the same domain?
2.3 What are the limitations and challenges of achieving real time object detection on a CPU-based platform like the Intel NUC?

3. To what extent can integrating autonomous exploration techniques, coupled with 3D object localization, increase the maturity of the autonomous capabilities of the Lunar Rover Mini?
3.1 How can hierarchical finite state machine architecture be designed in a modular way that supports future development?
3.2 What experimental validation methods can be used to assess the performance of the Lunar Rover Mini when generating unknown maps of the environment and detecting objects of interest?

3.3. Methodology

The methodology to achieve the research objectives is divided into several stages. To begin with, the theoretical background is established, covering the core principles of robotic autonomy and the foundational concepts of autonomous exploration. Then, this chapter addresses the three underlying capabilities necessary for autonomous exploration: visual odometry, SLAM, and a navigation stack with local mapping. This is necessary because a solid understanding of these concepts provides the basis for designing and implementing an integrated autonomous system with previous work dedicated to validating the autonomous navigation capabilities of the LRM [33]. In addition, the mathematical model that governs utility functions referenced in the literature for autonomous exploration will be presented. This includes not only the utility function, but also the principles behind travel cost and information gain computations. At the end, in the same chapter, the theory that governs object detection algorithms, specifically YOLO, is presented.

Further, a selection of the open-source state machine software will be made based on a trade-off study, along with an autonomous exploration framework that defines how sensor data from the LRM, particularly the 3D point cloud from the stereo vision camera, can be used to detect and filter potential frontier candidates. Hence, its implementation within the chosen software will provide the system with the necessary capabilities to integrate and test different exploration algorithms. This is done through hierarchical state machines, making it essential to ensure that the state machine architecture is well designed and structured.

Furthermore, before the actual implementation of the exploration algorithm, it is necessary to understand how occupancy grids can be managed to determine frontiers, as well as what are the criteria that determine where a frontier is located and how to generate 3D maps of the environment. Once the general architecture for exploration is out of the way, answering research sub-question 1.1 and research sub-question 3.1, it is possible to move towards the design of the exploration algorithm that advances the field of open-source autonomous exploration strategies. To this end, a trade-off study is performed to opt for the most suitable exploration strategy, with the remaining pages of this chapter dedicated to explaining the proper implementation of this exploration algorithm, answering research sub-question 1.2 and the overall research question 1.

Moreover, the focus will shift toward developing a state machine loop responsible for both object detection and 3D localization within the global map frame. A key challenge arises to address research sub-question 2.3, from the fact that inference must run on an Intel NUC rather than on high-end GPUs, which imposes computational constraints on real time performance, which is necessary for this loop to run in parallel with autonomous exploration.

Regarding the LRM, as mentioned in Section 2.2, it is closely integrated with the LN Manager, which manages all processes running on the rover. Therefore, deploying the chosen state machine software within the LN Manager is critical, meaning that upon starting the LN Manager of the rover, there should be a new node that starts LRM's autonomous exploration framework based on hierarchical state machines, with different open-source state machine libraries. Ultimately, the

working architecture will be generalized, using, for example, user-defined global variable for different nodes, to have a generalized and scalable architecture to be used on other low-performance rovers.

The validation experiments will involve the LRM autonomously surveying and generating an accurate 3D map of a previously unknown environment, while localizing objects of interest in the 3D global map frame, covering research sub-question 3.2, answering research question 3. The implemented strategy will be compared to open-source existing ones in the literature to tackle research sub-question 2.2. For this, a trade-off analysis will be conducted between the criteria used to assess the effectiveness of the exploration strategy, dealing with research sub-question 2.1 and, finally answering research question 2. Examples of criteria could be computational load, total distance traveled, map coverage, and total survey time.

3.4. Expected Outcomes

The expected outcomes of this work follow directly from the methodology and aim to address the research questions outlined earlier.

Research Sub-Question 1.1

It is expected that the analysis will identify key architectural and computational bottlenecks when implementing exploration strategies on low-cost robotic platforms such as the LRM, mainly concerning the number of user-defined parameters that need to be defined and the relative position of the stereo vision camera with respect to the based frame in different robotic systems. It is anticipated that the limitations may be related to CPU usage.

Research Sub-Question 1.2

It is expected that frontier- and utility-based exploration algorithms will be successfully adapted as modular state machines, compatible with the chosen state machine software. The expected result includes a state-based implementation where exploration decisions are governed hierarchically, ensuring reusability.

Research Question 1

The expected outcome is a fully functional, open-source, and modular exploration framework for low-cost rovers, implemented within a hierarchical finite state machine. This framework shall be scalable and modular across different robotic systems and provide a template for future autonomous exploration projects.

Research Sub-Question 2.1

It is foreseen that a set of evaluation metrics will be defined and validated for low-cost autonomous exploration systems. These will likely include map coverage,

exploration time, and computational load. The outcome will be a standardized evaluation framework applicable to resource-constrained platforms.

Research Sub-Question 2.2

It is predicted that experimental comparisons will demonstrate that the proposed utility-based exploration strategy achieves better performance according to the benchmarks relative to existing open-source exploration algorithms. Specifically, it is expected that the proposed method will result in a better exploration efficiency. In other words, for the same exploration time, more of the environment has been explored. These results will support the hypothesis that well-optimized, low-cost systems can achieve performance comparable to high-end rovers.

Research Sub-Question 2.3

It is expected that the study will identify the computational limitations that restrict real time inference on CPU-based systems and provide a solution to deploy object detection algorithms for this effect. On top of that, it is likely that the solution path is through model quantization. Nevertheless, the actual challenges will only arise when the implementation is being carried out.

Research Question 2

Taking everything into consideration, a comprehensive performance evaluation will be completed, based on the given benchmarks for the exploration strategy and the accuracy of the proposed object detection pipeline using real-world experiments. The expected outcome is a set of empirical results demonstrating the real time performance of the chosen design.

Research Sub-Question 3.1

It is assumed that that a generalized HFSM architecture will be developed and implemented in the LRM, enabling future modular integration of new functionalities. The outcome will include a library of reusable state machine modules, a clearly documented architecture.

Research Sub-Question 3.2

The outcome will include a set of procedures that validate that the LRM can autonomously generate accurate 3D maps of an unknown environment and detect and localize objects of interest.

Research Question 3

At last, the expected outcome is an increase in the autonomy level of the LRM, moving from autonomous navigation toward a fully autonomous exploration system. The integration of autonomous exploration within a unified HFSM will demonstrate that low-cost, open-source platforms can achieve the same level of performance

as high-end systems. This will also set a baseline for future extensions, including autonomous grasping and sample retrieval.

3.5. Research Plan

The following Gantt Chart presents the thesis planning, with the key phases and milestones.

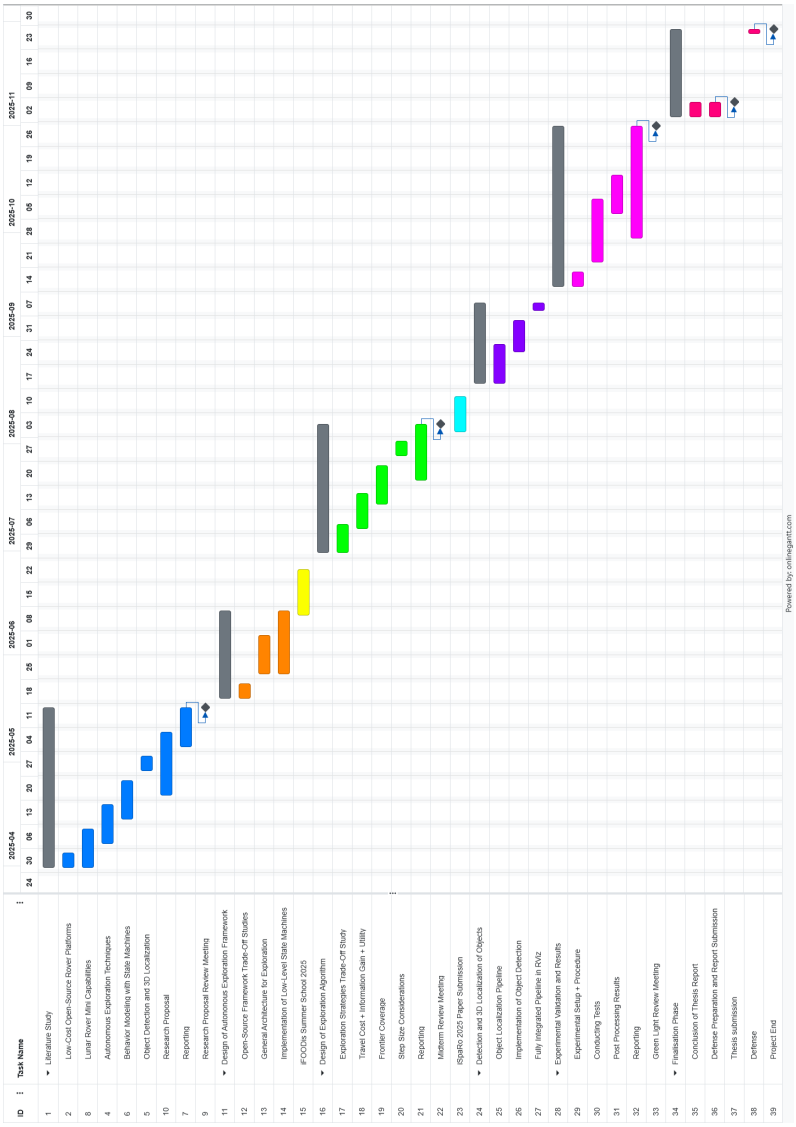


Figure 3.1.: Thesis planning Gantt chart.

4

Theoretical Background

This chapter outlines the underlying concepts and theories that guide the thesis. They provide the foundation needed for the reader to fully understand the research study. The chapter is structured as follows. To begin, Section 4.1 briefly introduces the fundamental concepts and general functional architecture that form the basis of autonomous mobile robotic systems.

three previously mentioned foundational capabilities for autonomous exploration. Even though they have already been implemented on the rover, they are a core part of any autonomous exploration implementation and will need to be properly improved to provide enough robustness for autonomous exploration. Secondly, Section **??** describes frontier-based exploration from a mathematical point-of-view, as well as a thorough definition of the different autonomous exploration strategies that will be implemented.

4.1. Core Principles of Robotic Autonomy

This section presents the core concepts and functional decomposition that underpin autonomous mobile robotic systems. The purpose is not to provide an exhaustive review of each subfield, but to establish the concepts and the key relationships between subsystems that will be used throughout this thesis.

At a high level, a robot, independent of its autonomy level, performs a task by iteratively sensing its environment, planning an appropriate course of action, and executing that action through its actuators. This sense–plan–act architecture [112] forms the backbone of robotic autonomy, ensuring that the system continuously perceives the environment, updates its internal state, generates feasible strategies, and interacts with the world in a goal-directed manner. This process is illustrated in Figure 4.1. The figure also depicts the closed perception–action feedback loop, represented by the connection from actuators to sensors through the environment. When the actuators produce motion, they modify the robot's state and the surrounding environment. These changes are then captured by the sensors, which provide updated information to the perception layer. In this way, the environment

acts as the intermediary between action and perception, completing the feedback cycle that underpins autonomous operation.

Complementary to this, the figure also illustrates a subsumption architecture—a layered control framework in which complex behaviors emerge from the interaction of simpler, independent modules rather than from a centralized planner. It is composed of three hierarchical layers: avoid obstacles, wander freely when safe, and navigate toward a goal. In this structure, higher layers can suppress or subsume the actions of lower ones when necessary, producing context-appropriate behavior through simple local interactions. For example, during exploration, a low-level obstacle-avoidance behavior may temporarily override the goal-directed navigation layer to ensure safety. This mechanism enables robust, adaptive, and real-time autonomy without relying on detailed world models.

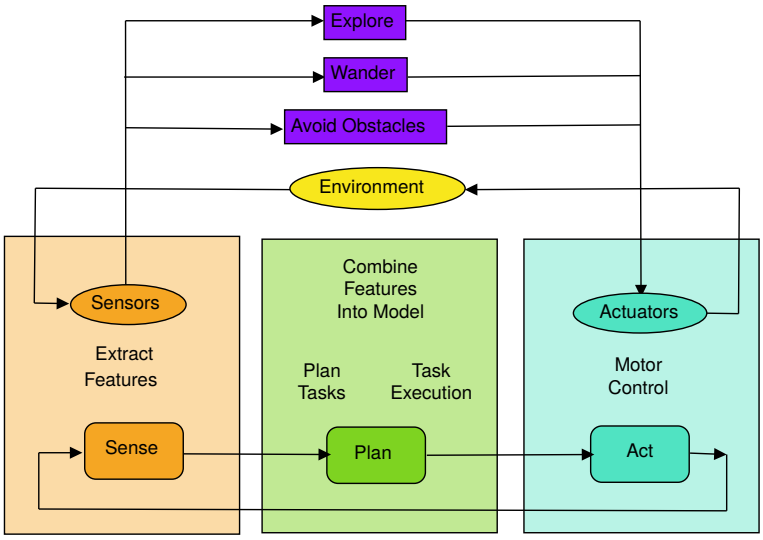


Figure 4.1.: Horizontal decomposition of the sense–plan–act pipeline [113] with subsumption architecture [112].

At a lower level, the autonomy stack shown in Figure 4.2 illustrates the hierarchical organization of an autonomous robotic system and the flow of information across its functional layers. At the hardware level, **Sensors** such as cameras, LiDAR, radar, IMUs, GPS, and wheel encoders capture raw data that describe the robot's interaction with its **Environment**. This information is processed by the **Perception** layer, which performs data fusion, object detection, and SLAM to **Map** and generate a coherent representation of the surroundings and the robot's state. Within the software domain, **Planning** modules use this representation to compute feasible motion trajectories and behavioral strategies that satisfy mission objectives while avoiding obstacles. The **Control** layer then translates these trajectories into low-level actuator commands through algorithms that ensure stability, accuracy, and responsiveness. Finally, **Actuators**—including steering and steering mechanisms

as well as pan-tilt unit rotation—execute these commands to produce physical motion, influencing the environment and closing the perception–action loop.

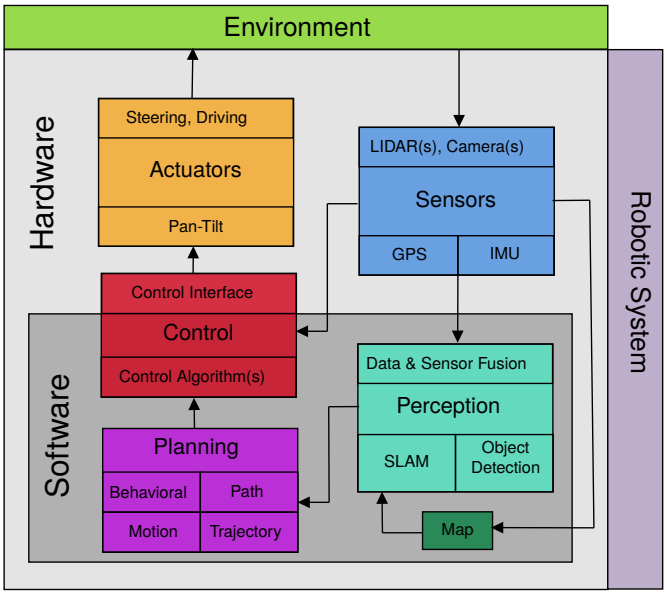


Figure 4.2.: Functional architecture of an autonomous robotic system [114].

4.2. Autonomous Exploration Foundational Capabilities

As previously mentioned, three complementary capabilities, VO, SLAM, and the navigation stack, are widely recognized as essential for the implementation of an autonomous exploration system, as outlined by Uslu et al. [10]. Thus, their mathematical principles and algorithms are presented below.

4.2.1. Visual Odometry

VO is defined as the process of estimating the position and orientation of a moving system by analyzing a sequence of images captured over time by a camera [115]. The goal of VO is to determine the camera motion, *i.e.*, its rotation and translation, between consecutive frames by tracking visual features in the environment. Compared to conventional odometry methods such as wheel encoders or inertial navigation systems, VO provides greater accuracy and robustness, particularly in environments where wheel slippage or drift affects measurements.

Let I_t and I_{t+1} denote two consecutive image frames captured at times t and $t+1$. The goal of VO is to estimate the camera motion $T_{t,t+1} \in SE(3)$, represented

as [115]:

$$T_{t,t+1} = \begin{bmatrix} R_{t,t+1} & t_{t,t+1} \\ O & 1 \end{bmatrix} \tag{4.1}$$

where $R_{t,t+1} \in SO(3)$ is the rotation matrix and $t_{t,t+1} \in \mathbb{R}^3$ is the translation vector between the two camera poses.

Given a 3D point $P_t = [X_t, Y_t, Z_t]^T$ observed in the first frame, its projection on the image plane is given by the pinhole camera model [116]:

$$s \begin{bmatrix} u_t \\ v_t \\ 1 \end{bmatrix} = K[R_t|t_t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4.2}$$

where K is the intrinsic camera calibration matrix and s is the scale factor. When the camera moves, the same 3D point is projected at a new pixel location (u_{t+1}, v_{t+1}), such that:

$$P_{t+1} = R_{t,t+1}P_t + t_{t,t+1} \tag{4.3}$$

The core VO problem is thus formulated as minimizing the reprojection error between matched feature points [117]:

$$E(R_{t,t+1}, t_{t,t+1}) = \sum_{i=1}^{N} ||\pi(K(R_{t,t+1}P_i + t_{t,t+1})) - p_i'||^2 \tag{4.4}$$

where $\pi(\cdot)$ projects 3D points to 2D pixel coordinates, p_i' is the observed pixel in the second frame, and N is the number of matched features. This nonlinear least-squares problem is typically solved using bundle adjustment or Gauss–Newton optimization, allowing refinement of the estimated motion parameters.

In stereo VO, depth information allows each pixel correspondence to be associated with a metric 3D point. The motion between frames can then be estimated directly in Euclidean space using Perspective-n-Point (PnP) or Iterative Closest Point (ICP) formulations. The PnP problem estimates $R_{t,t+1}$ and $t_{t,t+1}$ from 3D-2D correspondences:

$$p_i' \sim K[R_{t,t+1}|t_{t,t+1}]P_i \tag{4.5}$$

The LRM employs a feature-based RGB-D VO approach using an Intel RealSense D435i camera, integrated through the `rgbd_odometry` node in ROS. Keypoints are extracted using the Good Features to Track (GFTT) detector [38], and descriptors are computed using BRIEF [39] and motion estimation follows a Frame-to-Frame scheme, which improving robustness compared to a traditional Frame-to-Map approach.

4.2.2. Simultaneous Localization and Mapping

The SLAM problem can be expressed probabilistically as estimating both the robot trajectory $X_{1:t} = x_1, ..x_t$ and the map m, given a sequence of controls $U_{1:t}$ and observations $Z_{1:t}$:

$$p(X_{1:t}, m|Z_{1:t}, U_{1:t}) \tag{4.6}$$

This can be factored recursively as [118]:

$$p(X_{1:t}, m|Z_{1:t}, U_{1:t}) \propto p(Z_t|X_t, m)p(X_t|X_{t-1}, U_t)p(X_{1:t-1}, m|Z_{1:t-1}, U_{1:t-1}) \tag{4.7}$$

where $p(X_t|X_{t-1}, U_t)$ models motion, *i.e.*, odometry, and $p(Z_t|X_t, m)$ models perception, *i.e.*, sensor measurements. More importantly, this Bayesian factorization underpins both filter-based (EKF-SLAM, FastSLAM) and graph-based (RTAB-Map, ORB-SLAM) approaches.

This being the case, in graph-based SLAM, each node represents a robot pose x_i, and edges represent spatial constraints between poses, Figure 4.3. The optimization problem is to find the set of poses $X = x_1, \ldots, x_n$ that minimize the total error over all constraints [119]:

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{(i,j)\in C} ||e_{ij}(X)||^2_{\Omega_{ij}} \tag{4.8}$$

where $e_{ij}(X) = z_{ij} - \hat{z}_{ij}(x_i, x_j)$ is the error between the measured and predicted relative transformation, Ω_{ij} is the information matrix, z_{ij} is the observed relative pose between nodes i and j, and $\hat{z}_{ij}(x_i, x_j)$ is the expected relative pose given the current estimates.

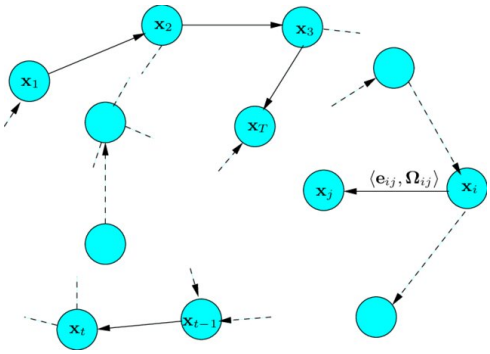


Figure 4.3.: Example of a pose graph in graph-based SLAM. Each node represents a robot pose, and edges denote spatial constraints derived from odometry and loop closures [119].

Similarly to VO, this nonlinear least-squares problem is typically solved via Gauss–Newton or Levenberg–Marquardt optimization.

The LRM implements graph-based SLAM using RTAB-Map within the ROS framework. RTAB-Map stands for Real-Time Appearance-Based Mapping and relies

on visual similarity detection to close loops [40]. Each new key frame is compared against a database of past images using bag-of-words (BoW) appearance features. A loop closure is detected when two key frames I_i and I_j have high visual similarity and spatial consistency, leading to a new constraint z_{ij} added to the pose graph.

Mathematically, the BoW similarity is computed as the cosine similarity between histogram representations h_i and h_j:

$$s(I_i, I_j) = \frac{h_i \cdot h_j}{||h_i||\,||h_j||} \tag{4.9}$$

If $s(I_i, I_j) > \tau$, a loop-closure constraint is proposed and verified geometrically via for example PnP pose estimation.

0nce constraints are added to the graph, RTAB-Map peforms incremental optimization using libraries such as g2o[1] to refine pose estimates in real time [120]. The optimization problem can be expressed as:

$$\min_X \sum_{k=1}^{M} ||z_k - f(x_{i_k}, x_{j_k})||^2_{\sum_k} \tag{4.10}$$

where $f(\cdot)$ models the relative motion function between poses. The interface of both the `rgbd_odometry` and `rtabmap` node with ROS topics is presented in Figure 4.4.

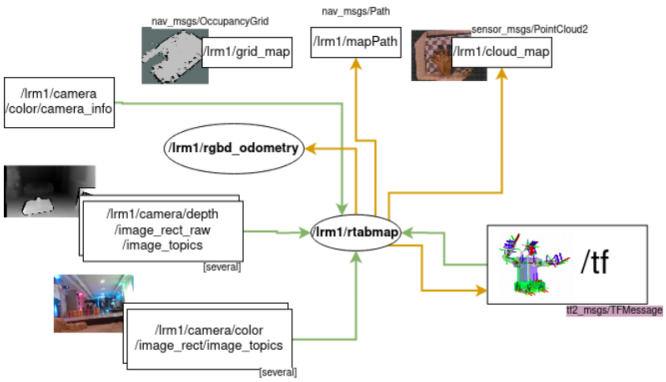


Figure 4.4.: ROS topic interface between the `rgbd_odometry` and `rtabmap` nodes [33].

4.2.3. Local Path Planning and Mapping

Local navigation in the LRM is managed by the `rmc_gbr_navigation` ROS package, a software module developed at DLR for ground-based robotic platforms.

[1] More information about the g2o graph-based nonlinear error functions optimizer can be found here.

This package provides the local path planning, including obstacle avoidance, trajectory generation for path following, and motion control to drive the robot toward a specified local goal. The navigation diagram is presented in Figure 4.5.

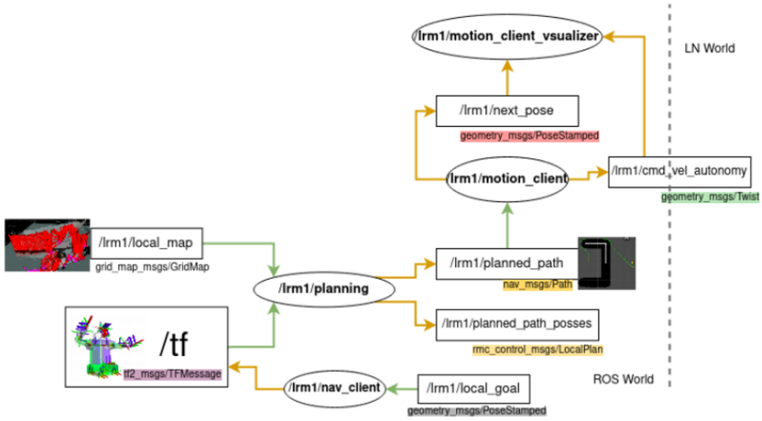


Figure 4.5.: Navigation diagram interfacing with ROS topics and LN [33].

At its core, the planner employs the A* search algorithm to compute an optimal path between the rover's current position and a target waypoint. The A* algorithm explores multiple potential paths through a discretized map and incrementally refines its solution based on a cost function that balances traversability and distance to the goal [42].

Formally, A* operates on a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge $(n, n') \in E$ has an associated traversal cost $c(n, n') < 0$. The algorithm seeks the path:

$$\pi^* = \underset{\pi \in \Pi(s,g)}{\operatorname{argmin}} \sum_{(n,n') \in \pi} c(n, n') \tag{4.11}$$

that minimizes the total accumulated cost from the start node s to the goal node g [42]. Here, $\Pi(s, g)$ denotes the set of all possible paths connecting s and g.

To efficiently approximate this minimization, A* evaluates each node n using the cost function

$$f(n) = g(n) + h(n) \tag{4.12}$$

where $g(n)$ represents the known cost from the start node to n, and $h(n)$ provides a heuristic estimate of the remaining cost to reach the goal. At each iteration, the algorithm expands the node with the smallest $f(n)$, effectively trading off path optimality and computational efficiency.

The heuristic $h(n)$ is typically defined using a distance metric such as:

$$h(n) = \begin{cases} \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}, & \text{Euclidean distance,} \\ |x_n - x_g| + |y_n - y_g|, & \text{Manhattan distance.} \end{cases} \tag{4.13}$$

The search continues until the goal node is reached, after which the optimal path is reconstructed by backtracking through the visited nodes. This approach enables efficient and deterministic motion planning, ensuring that the rover can safely navigate around obstacles while progressing toward its goal. In addition to path planning, DLR's `rmc_gbr_mapping` ROS package, which diagram is depicted in Figure 4.6, integrates a local mapping module based on a 2D grid map representation.

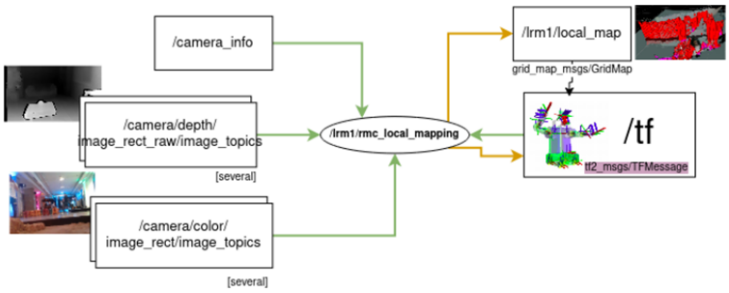


Figure 4.6.: Local mapping diagram interfacing with ROS topics [33].

4.3. Occupancy Mapping and Voxel Representation

Occupancy mapping is a fundamental component of robotic perception, providing a representation of the environment that distinguishes free, occupied, and unknown regions. This representation forms the foundation for higher-level tasks such as path planning, obstacle avoidance, and autonomous exploration. In the context of the LRM, a voxel map is maintained using the `octomap_server` package in ROS, which implements the OctoMap framework [86].

The core idea behind occupancy mapping is to estimate, for each 2D grid cell or 3D voxel m_i, the probability that it is occupied given all past sensor measurements $z_{1:t}$ and robot poses $x_{1:t}$. Formally, this can be expressed as [121]:

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, x_t)\, p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})}, \tag{4.14}$$

where $p(z_t|m_i, x_t)$ is the inverse sensor model describing how likely a measurement z_t is given the occupancy state of voxel m_i.

To simplify computation, the occupancy probability is often updated in the *log-odds* form:

$$L(m_i|z_{1:t}, x_{1:t}) = L(m_i|z_{1:t-1}, x_{1:t-1}) + \log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} - L_0, \tag{4.15}$$

where $L(m_i)$ denotes the log-odds of occupancy and L_0 is the prior log-odds value. This recursive update allows efficient real-time computation while maintaining probabilistic consistency.

The occupancy probability can be recovered from its log-odds form using:

$$p(m_i) = \frac{1}{1 + \exp(-L(m_i))}. \tag{4.16}$$

A 3D occupancy grid requires storing probabilities for every cube voxel, which leads to high memory usage. OctoMap addresses this issue by using an octree data structure, which recursively subdivides space into cubic cells of varying resolution. Each node in the octree corresponds to a cubic volume that can be either occupied, free, or further subdivided into eight child nodes.

Mathematically, the recursive occupancy update for an octree node n is defined as:

$$p(n|z_{1:t}, x_{1:t}) = \begin{cases} p_{\text{occ}}, & \text{if hit by a sensor ray,} \\ p_{\text{free}}, & \text{if passed through by a ray.} \end{cases} \tag{4.17}$$

Each time a new point cloud is received, sensor rays are traced from the sensor origin to each measured 3D point. Voxels traversed by the ray are updated toward the free space probability p_{free}, while the voxel corresponding to the endpoint is updated toward the occupied probability p_{occ}.

4.4. YOLO-Based Object Detection

Object detection refers to the task of identifying instances of objects within an image and assigning them a class label along with a spatial location, usually in the form of a bounding box. YOLO is a state-of-the-art, real time object detection algorithm that treats detection as a single regression problem, directly predicting class probabilities and bounding box coordinates from full images in one evaluation [122].

YOLO divides an input image into an $S \times S$ grid. Each grid cell is responsible for predicting B bounding boxes and associated confidence scores. Formally, for an input image I:

$$y = f_\theta(I) = \{(x_i, y_i, w_i, h_i, c_i, p_i)\}_{i=1}^{S^2 \cdot B} \tag{4.18}$$

where (x_i, y_i) is the center of the bounding box relative to the grid cell, w_i, h_i are the width and height of the box (normalized by the image dimensions), c_i is the confidence score that the bounding box contains an object, $p_i = \{p_i^1, p_i^2, \ldots, p_i^C\}$ are the conditional class probabilities for C object classes, and θ represents the learned network parameters.

The predicted bounding box confidence score is defined as:

$$\text{Confidence} = P(\text{Object}) \cdot \text{IOU}_{\text{pred,truth}} \tag{4.19}$$

where $P(\text{Object})$ is the probability that an object is present in the box and $\text{IOU}_{\text{pred,truth}}$ is the intersection-over-union between the predicted and ground-truth bounding boxes.

YOLO uses a single convolutional neural network to predict all bounding boxes and class probabilities simultaneously. The network is trained by minimizing a

multi-part loss function that balances localization, confidence, and classification errors [103]:

$$\begin{aligned}\mathcal{L} = \lambda_{\text{coord}} &\sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}}\left[(x_i-\hat{x}_i)^2 + (y_i-\hat{y}_i)^2\right] \\ &+ \lambda_{\text{coord}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}}\left[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2 + (\sqrt{h_i}-\sqrt{\hat{h}_i})^2\right] \\ &+ \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}}(c_i-\hat{c}_i)^2 \\ &+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}}(c_i-\hat{c}_i)^2 \\ &+ \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c\in\text{classes}} (p_i(c)-\hat{p}_i(c))^2 \end{aligned} \tag{4.20}$$

where $\mathbf{1}_{ij}^{\text{obj}}$ and $\mathbf{1}_{ij}^{\text{noobj}}$ are indicator functions for the presence or absence of an object in the bounding box, $\hat{x}_i$, $\hat{y}_i$, $\hat{w}_i$, $\hat{h}_i$, $\hat{c}_i$, $\hat{p}_i(c)$ denote the ground-truth values, and λ_{coord}, λ_{noobj} are parameters to weight localization and no-object penalties.

4.5. Frontier-Based Autonomous Exploration Theory

Autonomous exploration is the process by which a robot incrementally constructs a map of an unknown environment while deciding where to move next to maximize knowledge about the surroundings. The theory behind exploration can be formalized using concepts such as frontiers, information gain, and utility functions. Nevertheless, only a general formulation is presented here, as a more detailed one strongly depends on the specific exploration strategy employed, which is provided only in the design phase of the work.

Frontier-based exploration is a widely used strategy for autonomous mapping, where a frontier is defined as the boundary between known free space and unknown space in the occupancy map [8]. Formally, given an occupancy map m, the frontier set F can be defined as:

$$F = \{c \in \text{unknown cells} \mid \exists n \in \text{free cells}, \; n \text{ adjacent to } c\}. \tag{4.21}$$

Exploration based on information gain evaluates candidate frontiers based on the expected reduction in map uncertainty. Let p_i denote the occupancy probability of voxel or cell i, and $H(p_i) = -p_i \log p_i - (1-p_i)\log(1-p_i)$ its Shannon entropy [49]. The expected information gain $I(x)$ for moving to a candidate location x is given by [123]:

$$I(x) = \sum_{i\in S(x)} \left[H(p_i) - H(p_i|z)\right], \tag{4.22}$$

where $S(x)$ is the set of cells observable from the camera's field of view at location x, and z represents the expected sensor measurements at x.

To balance exploration efficiency and travel cost, a utility function $U(x)$ is often defined. The utility function typically combines information gain and travel cost weighted by one or several scaling factors. An example of such utility function is the following:

$$U(x) = \alpha I(x) - \beta d(x), \tag{4.23}$$

where α and β are the weighting parameters. The exploration algorithm can then select the next goal x^* by maximizing the utility:

$$x^* = \arg\max_{x \in F} U(x). \tag{4.24}$$

5

Design of Autonomous Exploration Framework

It is essential to define a functional framework that serves as the foundation before the implementation of the actual exploration strategy, both in terms of the chosen open-source state machine software and general architecture for autonomous exploration. Once the framework is established, its implementation is carried out through low-level state machines.

In light of this, Section 5.1 of the following chapter focuses performing a trade-off study to determine the best open-source state machine software to use and justifies using ROS as the middleware, while Section 5.2 outlines the system-level architecture for autonomous explorationm and Section 5.3 establishes the low-level state machines and corresponding functionalities necessary to support this architecture.

5.1. Open-Source Framework Trade-Off Studies

The foundation of this thesis is the development of an open-source framework for autonomous exploration, built around a state machine-based architecture that can be easily adapted to different robotic platforms. To ensure the effectiveness and portability of this framework, it is essential to conduct trade-off studies that evaluate and select the most suitable software and middleware components for its implementation.

In particular, two key aspects must be carefully considered in this selection process: the middleware framework, which provides communication and integration between software modules, and the state machine software, which governs high-level task control and behavior coordination. As previously discussed in Section 2.2, the LRM relies on ROS for all autonomy-related functionalities. Given this existing integration, it is both practical and advantageous to continue using ROS as the underlying middleware for this work. ROS provides a mature, modular, and community-driven ecosystem that supports a wide range of hardware interfaces, and

sensor drivers, significantly reducing development time and integration complexity.

Furthermore, it has become the standard in research and industry for robotic middleware. For these reasons, a trade-off analysis among middleware options would invariably conclude that ROS is the most suitable choice. Alternative frameworks such as DLR's Links and Nodes Manager [37], YARP [124], or MOOS [125], might offer specific advantages in niche domains but lack ROS's ecosystem size, documentation quality, and integration flexibility. Therefore, ROS remains the most appropriate and future-proof middleware for the development of the autonomous exploration framework proposed in this thesis.

Another important consideration is that the LRM currently operates on ROS 1, whereas the more recent ROS 2 is now widely available and increasingly adopted. Migrating the entire software stack to ROS 2 before implementation would require significant development effort and reconfiguration of existing components, which is beyond the scope of this project. Nevertheless, several tools and bridging packages[1] facilitate interoperability and gradual migration between the two versions. Therefore, although the proposed framework is initially designed for ROS 1, its architecture and modular design allow for straightforward replication and future adaptation to ROS 2.

Nevertheless, this distinction is less nuanced when it comes to selecting alternative task management and execution frameworks for state machines, meaning that a proper trade-off study is necessary. Based on the reviewed literature, several state machine frameworks have been identified for evaluation in the following trade-off study: SMACH, FlexBE, RAFCON, the Decision Making ROS Package, and YASMIN. A set of criteria must be defined to evaluate these frameworks.

First is the **Ease of Integration with ROS**, *i.e.*, how seamlessly the framework interfaces with ROS nodes, topics, and services, enabling communication between the state machine and other software modules.

The second criterion is the **Graphical Interface and Usability**, which evaluates whether the framework provides an intuitive GUI for designing, and visualizing state machines, thereby simplifying development and maintenance.

Third, the **Modularity and Hierarchical Structure** criterion assesses the ability to create reusable and nested state machines, allowing complex behaviors to be composed from simpler components.

The fourth criterion, **Execution Monitoring and Debugging Tools**, refers to the mechanisms available to observe runtime behavior, inspect variable values, and manage state transitions during execution, which are critical for testing and fault diagnosis.

Finally, **Community Support and Documentation**, measures the extent of available resources, including user guides and tutorials, and the frequency of updates or maintenance by the developer community.

A proper method to apply the criteria for the different frameworks must be employed. As such, each framework will be ranked on a scale from 1 to 5 in each of the five criteria, with 1 considered the lowest and 5 considered the highest. For example, RAFCON being given a score of 5 in the graphical interface and

[1] One example of such bridging package is the ROS 1 Bridge, while a lot of documentation exists in the literature regarding Migrating from ROS 1 to ROS 2.

usability means it has a very intuitive GUI. Moreover, it is necessary to establish the importance of each criterion in the work's context. As such, a relative weight of importance is attributed as follows:

- Ease of integration with ROS, 1.0. Since ROS serves as the middleware for all perception, planning, and control tasks, seamless integration is critical. Therefore, this criterion receives the highest importance.
- Graphical interface and usability is given a relative weight of 0.6 because the ability to visually design, monitor, and debug state machines significantly reduces development time but is something that can be learned when using the framework consistently.
- Modularity and hierarchical structure is given 0.9. Modular and hierarchical state design is essential for scalable autonomy architectures, allowing complex missions to be composed of simpler behaviors.
- Execution monitoring and debugging tools, 0.8. Robust runtime monitoring and debugging support are important for iterative testing and ensuring reliable system behavior during autonomous operation.
- Community support and documentation, 0.6. While still relevant, this criterion has slightly lower importance, as it primarily affects long-term maintainability rather than immediate functional performance.

With the criteria being defined and the corresponding relative weight, it is now possible to perform the trade-off study, which results are presented in Table 5.1, where the Decision Making ROS Package was shortened to DMP ROS. The attributed values from 1 to 5 in the different criterion are based on available information for each framework. References for this are provided in the literature review chapter.

Criterion	Weight	SMACH	FlexBE	RAFCON	DMP ROS	YASMIN
Ease of integration with ROS	1.0	5	5	4	3	3
Graphical interface and usability	0.6	1	3	5	1	2
Modularity and hierarchical structure	0.9	3	4	5	3	2
Execution monitoring and debugging tools	0.8	2	3	5	2	2
Community support and documentation	0.6	4	3	3	2	1
Weighted sum		**12.3**	**14.6**	**17.3**	**9.1**	**8.2**

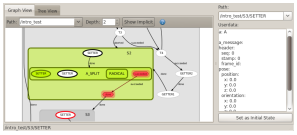
Table 5.1.: Application of the trade-off method to state machine frameworks.

From the trade-off analysis presented in Table 5.1, RAFCON, Figure 5.1c, clearly outperforms the other open-source frameworks, achieving the highest weighted score of 17.3. Its strong performance is primarily due to its comprehensive graphical user interface, hierarchical state management, and advanced debugging and

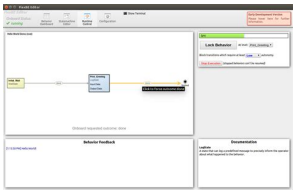
execution monitoring tools. FlexBE, Figure 5.1b, follows with a score of 14.6, showing very good integration with ROS and a reasonably intuitive graphical interface. However, it provides less flexibility and depth than RAFCON when managing large, hierarchical state structures.

SMACH, Figure 5.1a ranks third with a total score of 12.3. Although it integrates seamlessly with ROS just like FlexBE and benefits from mature documentation and community support, it lacks a good native graphical interface, which makes debugging and visualization more difficult during development. The reason SMACH and FlexBE rank better than RAFCON with respect to ease of integration with ROS is because these are ROS packages, while RAFCON state machine blocks interface with ROS by programming in Python. The Decision Making ROS Package and YASMIN, Figure 5.1d and Figure 5.1e, respectively, received significantly lower scores of 9.1 and 8.2, respectively. These frameworks are less actively maintained and limited in both usability and visualization capabilities. Also, note that, as mentioned in Chapter 2, FlexBE is a sucessor of SMACH.

However, it appears to have less documentation when compared to its predecessor. Also, RAFCON scores lower than those in this criterion because it has been been as widely adopted. However, it benefits from a very thorough documentation[2]. All in all, the trade-off results demonstrate that RAFCON provides the best balance between usability, modularity, debugging support, and ROS integration, making it the most suitable choice for implementing the autonomous exploration framework proposed in this work.

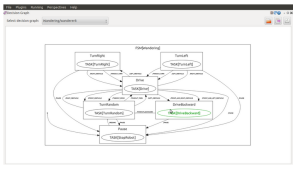


(a) SMACH viewer standalone GUI.



(b) FlexBE interface GUI.

(c) RAFCON editor GUI.



(d) Decision Making ROS Package plugin.

(e) YASMIN web viewer.

Figure 5.1.: Comparison of graphical interfaces from different state machine frameworks. *[Credits: SMACH Viewer, FlexBE, RAFCON, `rqt_decision_graph`, YASMIN [95].]*

[2]Available: https://rafcon.readthedocs.io/en/latest/

5.2. General Architecture for Exploration and Core Capabilities

Following the selection of the state machine framework, and prior to defining the specific exploration strategy to be implemented, it is essential to establish the general architecture for autonomous exploration. This architecture defines the hierarchical and functional organization of the state machines, ensuring modularity, scalability, and clear separation of responsibilities across different layers of autonomy. In this context, each state machine or state within the framework serves a distinct functional role and can be categorized as follows:

- **Hierarchical State**: A state that encapsulates multiple sub-states. This allows complex states to be stores as libraries ready-to-use. One example of such hierarchical state could be a state that contains the total autonomous mission pipeline.
- **Information Retrieval State**: Responsible for acquiring relevant data from ROS topics. For instance, obtaining the current robot pose or 2D occupancy grid.
- **Computation or Algorithm State**: Executes processing tasks or algorithms required for decision-making, such as frontier selection.
- **Action Execution State**: Triggers specific robot actions through control commands, e.g., sending velocity commands to move the robot or rotate the pan-tilt unit.
- **Software Interface State**: Manages communication and data exchange with external software or middleware modules, including mission control systems.
- **Monitoring and Feedback State**: Supervises ongoing actions, monitors success or failure conditions, and ensures that proper transitions occur in response to feedback. This could be monitoring the current position and orientation of the rover within a certain threshold.

Together, these functional state types form the foundation of the general architecture for exploration. In addition to defining the internal logic of the state machines, it is also necessary to specify the user-provided inputs required by the mission state machine. These inputs represent parameters or configurations that may vary between different robotic platforms. Since the goal of this framework is to remain as open-source and platform-independent as possible, the number of required user inputs should be kept to a minimum. For instance, if the user does not define the ROS topic responsible for publishing the position goals used by the autonomous navigation stack, the rover will be unable to move.

Conceptually, the framework can be viewed as a modular “black box” that receives input data, determines the optimal exploration strategy based on these inputs, and commands the robot to navigate toward computed position goals. The low-level motion control and path following are then handled by the autonomous navigation

stack, while the higher-level architecture continues to monitor system performance and environment changes in a closed-loop manner.

At the highest level, the mission state machine does not produce direct outputs in the traditional sense but instead publishes results via ROS topics, allowing them to be visualized or interpreted by the user. For example, after computing the next exploration target, the framework can publish both the selected goal and the set of available frontiers which can then be visualized in RViz. Ideally, the user just needs to press play on the state machine and wait until exploration is complete. The complete user input variables is presented in Table 5.2.

User Input	Description	Unit
Goal position ROS topic	ROS topic where position goals are published to be followed up by the autonomous navigation stack.	-
Camera frame name	Name of the camera frame, used to known the transformation between this frame and the robot reference frame.	-
Area to cover	Defines the total area within which the exploration mission will take place.	m^2
Position threshold	Minimum distance required for the robot to be considered as having reached a target goal position.	m
Orientation threshold	Minimum angle tolerance required for the robot to be considered as having reached a target orientation.	°
Cluster minimum size	Minimum number of frontier points to form a valid exploration cluster. Filters out small, insignificant regions.	-
Cluster maximum size	Maximum number of frontier points allowed per cluster to prevent excessively large frontier regions.	-
Robot reference frame name	Name of the reference frame, e.g., `base_link`, used by the robot for localization and transformation between coordinate systems.	-
OctoMap ROS topic	Name of the ROS topic publishing the 3D occupancy map (`/octomap_full`), used for computing entropy by ray casting voxels.	-
Occupancy grid ROS topic	Name of the ROS topic publishing the 2D occupancy grid map (`/grid_map`), used for frontier detection and navigation.	-

Table 5.2.: Full description of user inputs required for autonomous exploration mission state machine.

Aside from this, a more detailed diagram can be presented to illustrate how the proposed mission state machine integrates with the foundational capabilities required for autonomous exploration, Figure 6.2, namely, visual odometry, local mapping, and the navigation stack. Further, note the OctoMap ROS topic requirement, which originates from the ROS `octomap`[3] package. This constitutes the only component among the aforementioned modules that may not be natively implemented in all systems with autonomous navigation capabilities. However, justification for this is given below, as it enables the generation of a 3D volumetric representation of the environment, voxels, allowing the exploration algorithm to do ray casting across them, as explained in Chapter 4. This will allow for more complex exploration strategies similar to the ones used in more complex robotic platforms

[3]Available: https://wiki.ros.org/octomap

who simply use the 2D occupancy grid to decide on where to go next, as explained in the literature review.

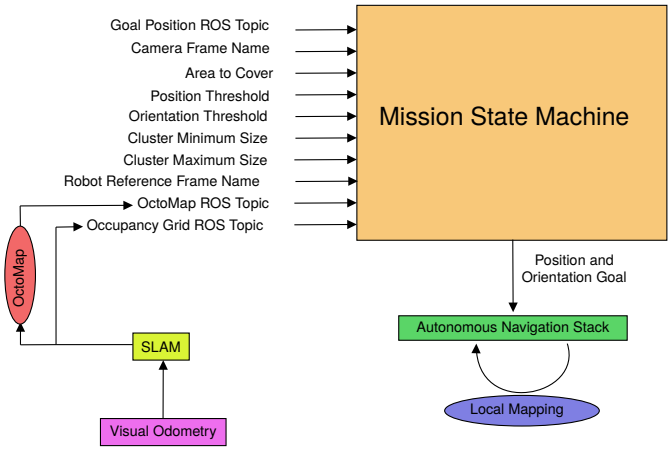


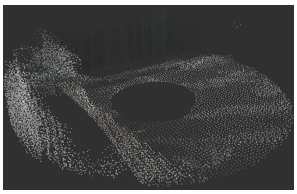
Figure 5.2.: Mission state machine integration with foundational capabilities required for autonomous exploration.

5.2.1. Pipeline Integration of ROS `octomap` Package

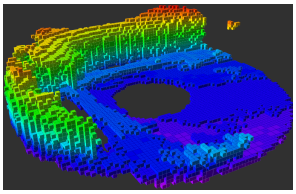
The integration of the `octomap` ROS package into the exploration framework is motivated by the need to perform ray casting operations and manage a volumetric representation of the environment. The need for this is explained more in-depth when the actual exploration strategy is discussed.

The `octomap_server` node within the package is responsible for building and distributing 3D occupancy maps as OctoMap binary streams, which can be published in multiple ROS-compatible formats for use in tasks such as obstacle avoidance, motion planning, or visualization. It uses an octree data structure, where each node represents a 3D volume, and children subdivide it into eight smaller cubes. These maps are constructed incrementally from incoming range sensor data (published as `sensor_msgs/PointCloud2`).

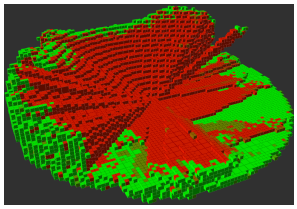
Since ROS messages play a crucial role in this work, Appendix B was created to direct the reader to the respective message definitions. This facilitates understanding of how the data contained in each message can be processed. Every time a ROS message appears, with defition is presented in the appendix. When no prior map is provided, the node starts with an empty representation that is continuously updated as new sensor data becomes available. In other words, this package converts the incoming 3D point cloud data, Figure 5.3a, into 3D probabilistic voxels, where each voxel has an occupancy probability. If the voxel is fully occupied the occupancy probability is 1, whereas if the voxel is fully free the occupancy probability is 0. Thus, a 3D probabilistic voxel map is created, Figure 5.3c gives 3D information of the state of occupancy of the surrounding environment.



(a) 3D point cloud.



(b) OctoMap of occupied voxels.



(c) OctoMap of all voxels.

Figure 5.3.: Illustration of the OctoMap generation process visualized in RViz during an exploration scenario. In (b), voxels are colored according to their height, while in (c), voxels are colored based on occupancy probability using a red-to-green gradient. Red indicates occupied space and green indicates free space. A clear correlation can be observed between (b) and (c).

Furthermore, note the difference between Figure 5.3b and Figure 5.3c. While the first one only shows occupied voxels, inferred directly from the 3D point cloud data, the latter depicts all voxels, free and occupied. This is because for every 3D point, p, in the point cloud, a ray is drawn from the origin to p. Voxels along the ray path are marked as free. The voxel at p is marked as occupied. Each voxel stores a log-odds probability, l_t of being occupied:

$$l_t = l_{t-1} + \log \frac{P(occupied|measurement)}{1 - P(occupied|measurement)} \tag{5.1}$$

Equation 5.1 also allows us to conclude that the more measurements a voxel receives, the more confident the system becomes about its occupancy state. Each update incrementally adjusts the voxel's log-odds probability, integrating information over time in a Bayesian manner. Consequently, repeated observations of free space along a ray decrease the occupancy probability, while repeated hits on a voxel increase it, refining the map's accuracy.

Now that the importance of generating an OctoMap has been established, it is necessary to explain how it integrates with the RAFCON-based state machine architecture. The OctoMap data is published in the form of `octomap_msgs/Octomap` messages, which encapsulate the probabilistic 3D occupancy grid generated by the `octomap_server` node. These messages contain all relevant metadata—including map resolution, reference frame, and the serialized octree structure. Note that the probabilities can be expressed in a continuous range from 0 to 1 or in binary form. The serialized octree structure means that, in order to read this data, it first needs to be deserialized, usually through the `conversions.h` header of the OctoMap package.

Since the state machines in RAFCON are built for Python, a python package that does this would be nice. There are two available python packages that serve as Python binding of the OctoMap library that could work. However, upon closer

inspection the bindings bindings primarily allow to create and manipulate OctoMaps in Python, *e.g.*, inserting point clouds, querying occupancy, or saving maps, but not directly deserialize the binary OctoMap messages published by.

Since the state machines in RAFCON are implemented in Python, having a Python package capable of handling OctoMap data would be advantageous. Two available packages provide Python bindings for the OctoMap library and could potentially serve this purpose[4] However, upon closer inspection, these bindings primarily allow the creation and manipulation of OctoMaps within Python—*e.g.*, inserting point clouds, querying voxel occupancy, or saving map files—but do not provide functionality to directly deserialize the binary OctoMap messages published by ROS. To address this limitation, a custom C++ ROS node was developed based on the `conversions.h` header from the OctoMap package. This node, named `octomap_saver_node.cpp`, remains continuously active and subscribes to a ROS topic named `/trigger_saver`, which uses a `std_msgs/Bool` message type. Every time a ROS topic is mentioned, its description is provided in Appendix B.

A corresponding state machine in RAFCON was designed to publish a `True` value to this topic whenever the algorithm requires access to the deserialized octree data structure for further computation. From the deserialized data, a `.csv` file is generated containing the coordinates of each voxel along with its corresponding occupancy probability. Figure 5.4 outlines how the node is integrated and called in a state machine to read data from the octree structure.

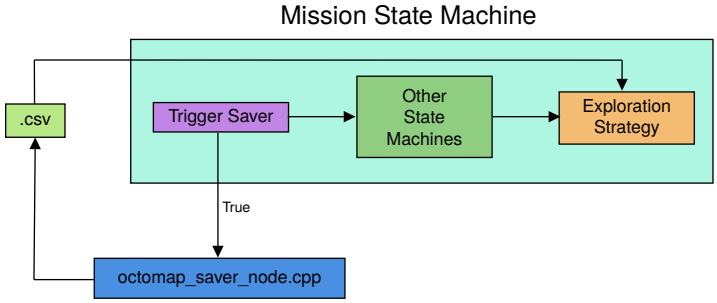


Figure 5.4.: Integration of custom C++ ROS node to deserialize the binary OctoMap messages.

5.2.2. Autonomous Exploration State Machine Architecture

After understanding how the OctoMap messages are deserialized, the first iteration of the general architecture for autonomous exploration can be defined. This architecture represents the initial design of the state machine hierarchy that will support the implementation of the exploration strategy. After understanding how the OctoMap messages are deserialized, the first iteration of the general architecture

[4]The first one is `octomap-python` and the seconds, more recent, is `pyoctomap`.

for autonomous exploration can be defined. This architecture represents the initial design of the state machine hierarchy that will support the implementation of the exploration strategy. A general architecture is proposed in Figure 5.5.

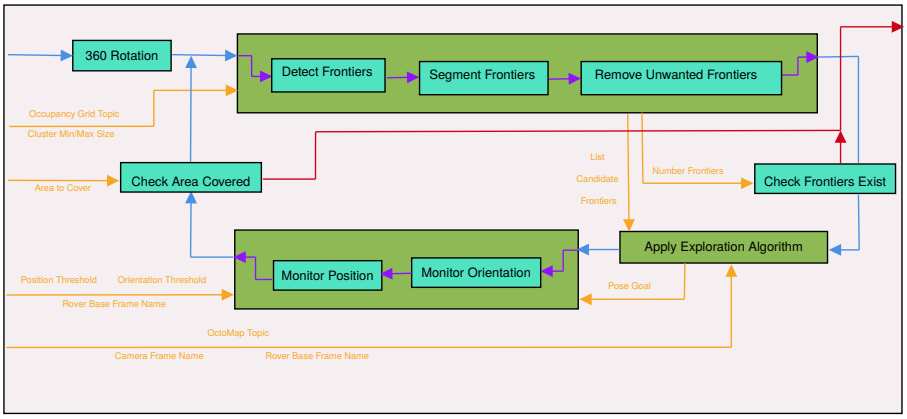


Figure 5.5.: General architecture of the initial autonomous exploration framework.

Here, the yellow arrows (→) represent variables provided by the user that are passed into state machines or other variables that are created by and passed between state machines. Blue arrows (→) indicate high-level transitions between hierarchical state machines, while purple arrows (→) correspond to low-level transitions between state machines within a hierarchical structure. Finally, red arrows (→) denote transitions that finalize the exploration process successfully. Of course, the general architecture represents a high-level overview, and it is expected that certain state machines within this structure will themselves contain additional nested state machines. Nevertheless, a description is provided that explain this architecture.

To begin, the rover performs a 360-degree rotation to maximize the amount of information available before moving. Since the rover is capable of in-place rotation, as described in Section 2.2, this maneuver is considered safe because no translational motion occurs during the process. After completing the rotation, the system transitions to a hierarchical state machine responsible for detecting and segmenting frontiers from the 2D occupancy grid. This process uses user-provided inputs, including the occupancy grid topic and the maximum and minimum cluster size parameters.

Classifying every individual grid cell as a separate frontier point would be inefficient, as the rover's stereo vision system has a sensing range much larger than the distance between consecutive grid cells. Therefore, frontier points are clustered into groups representing viable exploration frontiers. Additionally, this state machine removes invalid or unreachable frontier points—for example, those that are unreachable by the rover. If no frontiers are available, exploration concludes successfully, as indicated by the red transition arrow leading out of this state machine.

If viable frontiers do exist, the exploration algorithm selects the next frontier goal. This computation requires additional user-defined parameters, such as the OctoMap topic, the camera frame, and the rover base frame name. The selected frontier is converted into a pose goal, which is both passed to the next hierarchical state machine, responsible for monitoring rover motion, and published to the goal position ROS topic. The movement monitoring state machine operates passively, tracking progress without directly commanding motion. From this point, the autonomous navigation stack assumes control and drives the rover toward the pose goal. When the rover successfully reaches the target, the hierarchical state machine reports success, triggering the next transition.

Finally, a verification step evaluates whether the explored area exceeds a user-defined threshold. This option allows users to stop exploration based on a maximum coverage area, goal rather than waiting for all frontiers to be explored. This is especially important in large environments. If the coverage threshold is reached, the mission concludes successfully. Otherwise, the loop repeats until all frontiers have been explored or the defined area has been covered.

5.3. Implementation of Low-Level State Machines

Following the definition of the overall exploration architecture, this section focuses on the implementation of the low-level state machines that compose the hierarchical control structure. These implementations interact directly with the robot's foundational capabilities, them being perception, mapping, and navigation, through ROS topics. Each low-level state machine is responsible for performing a specific function.

5.3.1. Frontier Detection Pipeline

The first challenge of the frontier detection pipeline is to identify all grid cells that can be considered potential frontier candidates. As previously mentioned, frontier cells represent the boundary between known free space and unknown space in the occupancy grid. The process begins by scanning the 2D occupancy grid map, where each cell is classified as free, occupied, or unknown, depicted, by grey (■), black (■), and green (■) grid cells, respectively, in Figure 5.8a. A cell is labeled as a frontier candidate if it is free and has at least one adjacent cell that is unknown. The ROS topic publishing this information does it through a `nav_msgs/OccupancyGrid` message. For this, a state machine was created in RAFCON that subscribes to the ROS topic specified by the user containing the occupancy grid map. The state machine waits for the message and processes the received data to identify potential frontier cells.

Each cell in the map is checked: if it is free and at least one of its neighboring cells is unknown, it is classified as a candidate frontier. Besides, an important consideration must be made when interpreting the information from the occupancy grid. It is possible that, based on the 3D point cloud data, an unknown grid cell appears completely surrounded by free grid cells, Figure 5.6. Given the grid

resolution of 5x5 cm, such cells can be safely reclassified as free rather than unknown, since their small size and surrounding context make it unlikely that they represent unexplored space. Consequently, these cells are not treated as candidate frontiers.

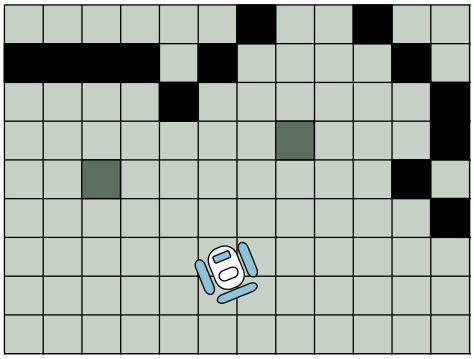


Figure 5.6.: Example of unknown grid cells fully surrounded by free grid cells, which can be safely reclassified as free.

The detected frontiers are then converted from grid indices to real-world coordinates using the map resolution and origin parameters. Finally, a `visualization_msgs/Marker` message is created, where each frontier cell is represented as a green cell in RViz, Figure 5.8b. This allows for the visualization of all candidate frontier points detected from the current occupancy grid. As an example, Figure 5.7 shows the implementation of such state machine in RAFCON.

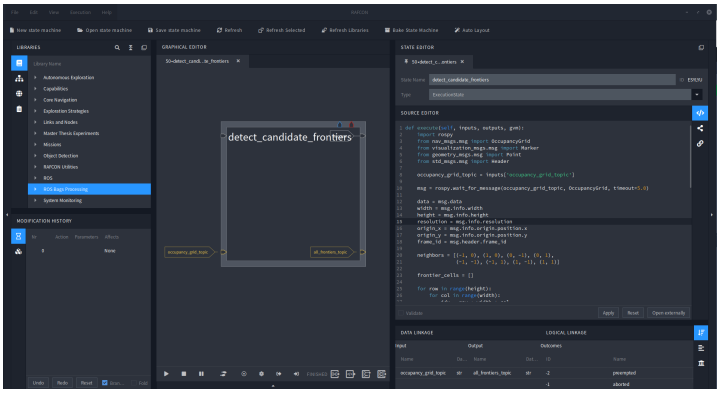


Figure 5.7.: Example of the state implemented in RAFCON for detecting all candidate frontiers.

Now that all possible candidate frontiers have been identified, they can be clustered for further processing. To accomplish this, a RAFCON state machine was

implemented to segment adjacent frontier cells into coherent clusters. The state subscribes to the ROS topic containing the previously detected frontiers. Each point in this message corresponds to a frontier cell in the occupancy grid. The algorithm constructs a graph based on Euclidean distance between points, where two points are considered neighbors if their squared distance is below a threshold derived from twice the map resolution. A BFS is then used to group connected frontier cells into clusters.

Each cluster is constrained by the maximum and minimum cluster size user-defined parameters. It is important that the cluster size can be adjusted, given that the range of the stereo vision cameras in different systems varies. Once all clusters are formed, they are published to a new topic as a `visualization_msgs/MarkerArray`, where each cluster is represented by a distinct color for easy visualization in RViz, Figure 5.8c. The resulting topic, serves as input for the next stage of the frontier detection pipeline.

On top of this, it is necessary to determine which point best represents each clustered frontier region. The centroid of the cluster is typically chosen because it provides a compact, spatially balanced representation of the entire frontier area [8]. The centroid of a frontier cluster composed of N grid cells is given by:

$$C = \left(\frac{1}{N} \sum_{i=1}^{N} x_i, \ \frac{1}{N} \sum_{i=1}^{N} y_i \right) \tag{5.2}$$

where (x_i, y_i) are the coordinates of the center of each grid cell and $C = (C_x, C_y)$ represents the centroid coordinates. This process is implemented through a dedicated RAFCON state machine that subscribes to the ROS topic containing the clustered frontiers. Upon receiving this message, the state machine iterates through each cluster, computes its centroid, and publishes the results as a new `MarkerArray` for visualization in RViz. The centroids are represented as spherical markers whose colors correspond to their originating clusters, ensuring visual consistency between clusters and their representative points, Figure 5.8d.

In addition to visualization, the state machine stores the computed centroids as output variables (`centroids_list` and `number_centroids`) that can be accessed by subsequent high-level state machines. These variables will serve as inputs for decision-making processes, such as selecting the next exploration goal or evaluating spatial coverage. Note that because this information is taken from the 2D occupancy grid there is no height coordinate. In addition, because the FOV of the camera is concave from the POV of the rover, it is almost certain that the centroid will fall within a known region of the map.

The results of a fully functional frontier detection pipeline are presented in Figure 5.8, captured from an experimental mission scenario. It can be observed that, due to the minimum cluster size constraint, some of the candidate frontiers detected in (b) are no longer present in (c). Furthermore, all centroids shown in (d) for this particular 2D occupancy grid lie within the known region of the map, confirming the consistency of the clustering and centroid computation processes.

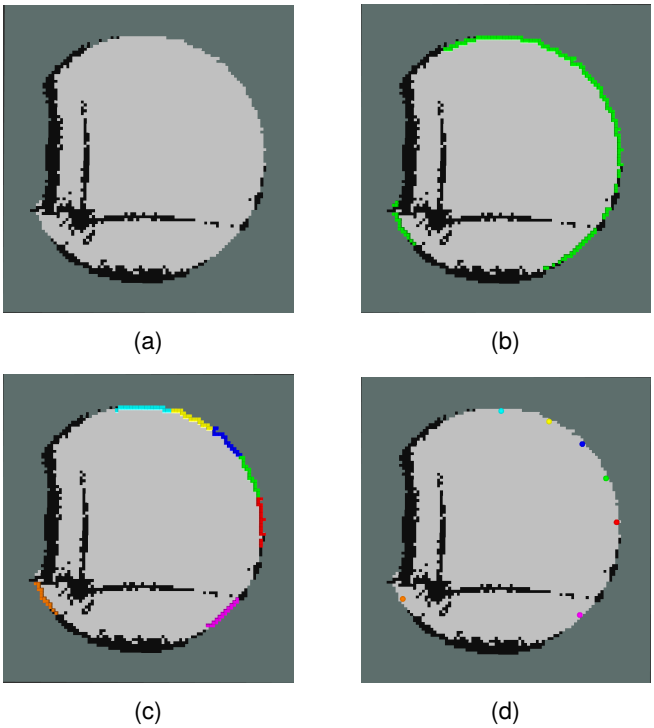


Figure 5.8.: Frontier detection pipeline from RViz: (a) generation of 2D occupancy grid, (b) detection of candidate frontiers, (c) clustering of frontiers, and (d) centroid computation for navigation goals.

5.3.2. Filtering Unwanted Frontiers

Although all computed centroids lie within the known region of the map, not all of them are guaranteed to be accessible by the rover. Consequently, additional state machines are employed to filter out unwanted frontiers from the centroid list generated by the frontier detection pipeline. Two main situations can render a centroid unsuitable for exploration:

- Inaccessible region due to occupancy: The centroid is located in a region that cannot be reached because it lies beyond occupied grid cells, Figure 5.9.
- Narrow passage constraint: The centroid is theoretically reachable—there exists a continuous region of free space connecting the rover and the frontier—but the available passage is narrower than the rover's required clearance, making traversal physically impossible, Figure 5.10, which is the case of the orange (●) centroid of Figure 5.8d.

To address this, two dedicated state machines were developed to filter out these unwanted frontiers before proceeding to the next stage of exploration. The first

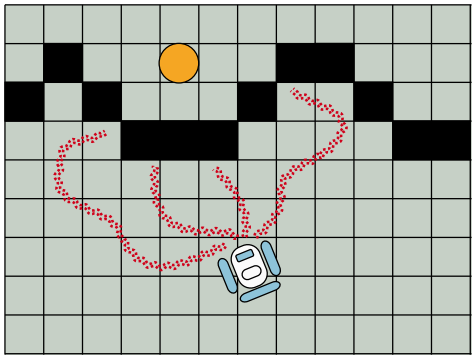


Figure 5.9.: Example of an inaccessible region due to occupied grid cells.

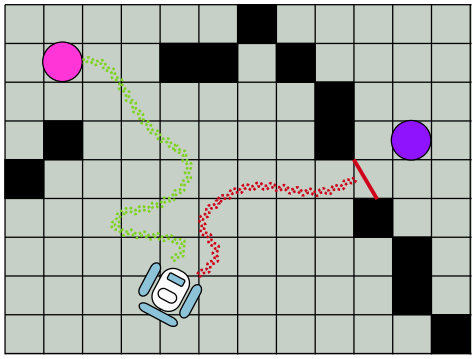


Figure 5.10.: Example of a narrow passage that prevents rover traversal.

state machine evaluates if the spatial there is a straight connection between each centroid and the rover's current position. If such an obstruction is detected, the corresponding centroid is discarded. The second state machine assesses the width of the free-space corridor leading to each remaining centroid. This is achieved by analyzing if the minimum width for a clear path between the rover and the centroid is bigger than a certain threshold that correspond to the rover's dimension with a safety factor being applied. The output of these two filtering stages is a refined set of valid frontier centroids, guaranteed to be both reachable and physically traversable by the rover.

With the frontiers detected and filtered, the next step is to describe how the rover's position and orientation are monitored while it navigates toward a goal position. As shown in the general architecture of the autonomous exploration framework (Figure 5.5), the subsequent stage involves applying the exploration algorithm. However, since this algorithm represents the core of this thesis and requires a more detailed explanation, the monitoring process is first presented. Afterwards, a dedicated section focuses on the exploration algorithm itself, detailing how the next frontier to explore is selected from the list of centroid frontiers obtained from the frontier

detection pipeline, after filtering out the unwanted ones.

5.3.3. Position and Orientation Monitoring

Monitoring the position and orientation of the rover is a crucial part of the autonomous exploration framework. In this context, the monitoring process does not control the rover directly; rather, it observes and verifies the progress of the autonomous navigation stack after a pose goal has been assigned. Each pose goal consists of two components. First, a position goal, which corresponds to the coordinates of the selected frontier centroid in the 2D occupancy grid, and then an orientation goal, that defines the desired heading of the rover when reaching the frontier.

Determining the appropriate orientation is a non-trivial task. Most works in autonomous exploration focus on moving the rover towards a frontier without explicitly specifying the desired orientation. However, orientation is important for maximizing sensor coverage and ensuring effective perception of the environment. In this implementation, the orientation goal is chosen such that the stereo vision camera is facing the frontier centroid. This ensures that when the rover reaches the target position, the camera is aligned with a line passes through the center of the rover and the frontier centroid. Figure 5.12 illustrates this concept.

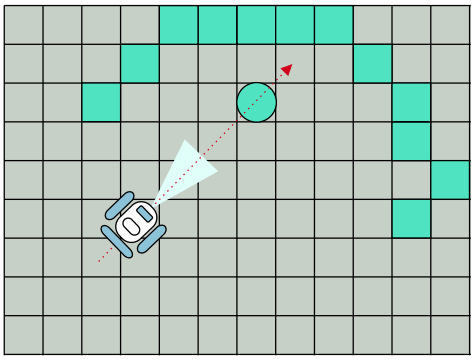


Figure 5.11.: Rover orientation aligned towards a frontier centroid. The arrow indicates the desired heading while approaching the target.

There are several passive approaches to monitor the rover's position. One straightforward method is to continuously track the rover's current pose at each timestamp and compare it with the given pose goal. The pose goal consists of two components, the position vector, (x, y, z), and the orientation represented as a quaternion, (x, y, z, w). This indicates that two different state machines can be created, each responsible for monitoring the rover's position and orientation, separately.

Through the `tf` Python package[5], the rover's position in the global map frame

[5]Available at: https://docs.ros.org/en/jade/api/tf/html/python/tf_python.html

can be retrieved at each timestamp. Knowing both the rover's current position and the frontier centroid, the Euclidean distance between them can be computed as:

$$d = \sqrt{(x_c - x_r)^2 + (y_c - y_r)^2} \tag{5.3}$$

where (x_r, y_r) represents the rover's position and (x_c, y_c) represents the coordinates of the frontier centroid in the 2D occupancy grid. The z coordinate is omitted since the centroid is defined within the 2D plane.

Then, a state machine can continuously monitor this distance, and once it falls below a user-defined position threshold, the state machine concludes successfully, indicating that the rover has reached the target. It is important to note that the rover's autonomous navigation stack also uses its own internal thresholds to determine when the goal position has been reached. Ideally, the user-defined threshold in the state machine should closely match the one used by the navigation stack to ensure consistent behavior.

If the threshold defined in the state machine is larger than that of the navigation stack, the state might terminate prematurely, before the rover's navigation system confirms goal completion. Conversely, if the state machine's threshold is smaller, it might continue running even after the navigation stack has already concluded that the goal has been reached, potentially leading to unnecessary delays.

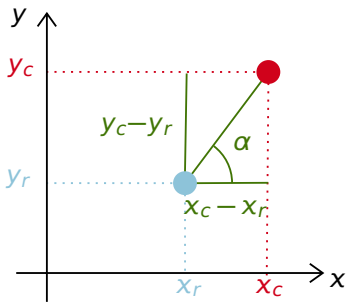


Figure 5.12.: Rover orientation aligned towards a frontier centroid. The arrow indicates the desired heading while approaching the target.

Regarding the orientation, it can be computed by determining the angle that makes the rover face the selected frontier centroid. This is done using the difference between the rover's current position and the centroid position in the global frame, as illustrated in Figure 5.12. The orientation angle, α, is computed as

$$\alpha = atan\left(\frac{y_c - y_r}{x_c - x_r}\right) \tag{5.4}$$

where α is then restricted to $[-\pi, \pi]$. After obtaining this angle, it is converted into a quaternion representation. This conversion is necessary because ROS uses quaternions to represent orientation in three-dimensional space, avoiding the singularities and discontinuities associated with Euler angles. The resulting

quaternion defines the orientation goal that can then be monitored, knowing the orientation of the rover at any timestamp.

Thus, three hierarchical state machines were developed for use within this pipeline: one that terminates successfully when a position threshold is reached, another that terminates successfully when an orientation threshold is reached, and a third that terminates successfully when both position and orientation thresholds are simultaneously satisfied. Using one or a combination of these hierarchical state machines yields similar outcomes when navigating toward a pose goal. Depending on the chosen configuration, the rover may first orient itself toward the selected frontier and then move to it, or it may perform both actions simultaneously.

5.3.4. Exploration Completion Checks

Two completion checks are employed to determine whether the exploration process shall continue. They are based on an area covered and frontier availability check. The area covered check computes the area of the environment that has been explored by multiplying the area of each grid cell, given by the grid cell resolution, by the total number of cells classified as either free or occupied in the occupancy grid. The frontier availability check determines wether there are still frontier centroids to explore by checking if the list of centroids obtained after filtering unwanted frontiers is greater than 0. If the user wants exploration to stop when there are no more frontiers to explore, it can bypass the state machine responsible for checking the covered area. This can be done either by changing the state transitions directly in RAFCON or by setting the area coverage threshold to a very high value.

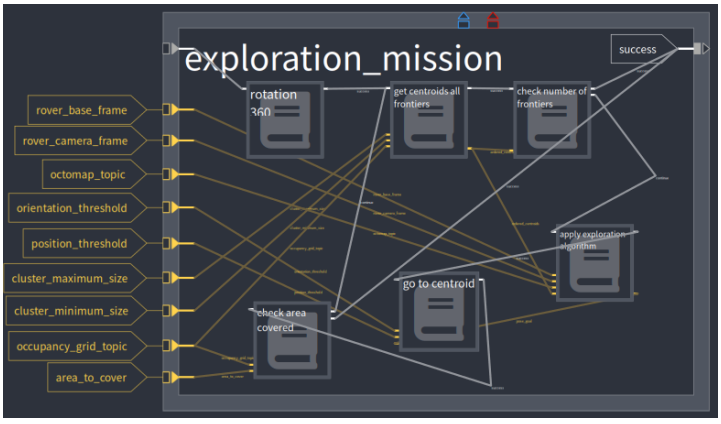


Figure 5.13.: Low-level state machine implementation of the autonomous exploration mission.

With this, Figure 5.13 illustrates the implementation of the low-level state machines thus far. This provides an exploration mission structure similar to the the general architecture of the exploration framework presented in Figure 5.5 (Subsection 5.2.2). The only hierarchical state machine, along with its corresponding

sub-state machines, that remains to be defined is the one related to the exploration strategy, which is presented in the following section. Note that some state machines correspond to execution states, *e.g.*, `check area covered`, while others are hierarchical states that contain either execution or additional hierarchical states. For instance, the hierarchical state `go to centroid`, Figure 5.14, is composed of two subordinate hierarchical states: `go to centroid pose orientation` and `go to centroid pose position`. Furthermore, `go to centroid pose orientation` itself, 5.15, consists of 4 distinct execution states. As well, note the user defined inputs to the left of the exploration mission hierarchical state machine.

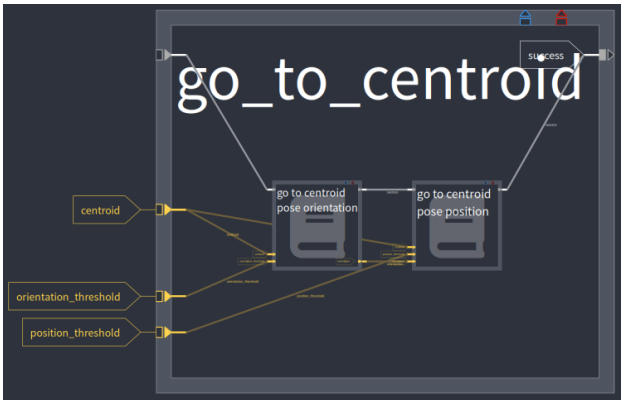


Figure 5.14.: Hierarchical structure of the `go_to_centroid` state machine.

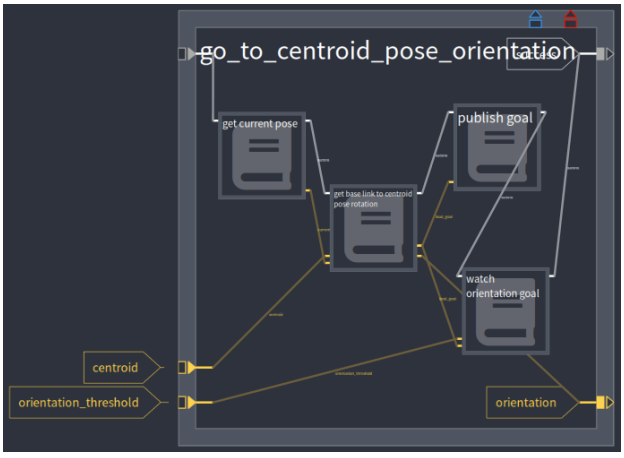


Figure 5.15.: Detailed view of the `go_to_centroid_pose_orientation` hierarchical state, composed of 4 execution states.

The `get current pose` state retrieves the current pose of the rover with

respect to the global map frame. This information is then passed to the `get base link to centroid pose rotation` state, which computes the orientation angle according to Equation 5.4 and converts it into a quaternion representation. Using this quaternion, the `publish goal` state machine publishes the corresponding orientation goal to the topic used by the autonomous navigation stack. Subsequently, the `watch orientation goal` state monitors the rover's current orientation angle and compares it to the desired orientation. Once a given orientation threshold is reached, the state machine exits successfully.

With the general architecture for exploration implemented in RAFCON comprising low-level state machines responsible for detecting potential frontier centroids through a frontier detection pipeline, filtering unwanted frontiers, and monitoring the rover's position and orientation, it is now possible to progress to the implementation of the exploration algorithm.

6

Design of Exploration Algorithm

The exploration strategy is the core this thesis work. On one hand, it shall be structured in a way that is easy enough to be implemented across different robotic platforms. On the other hand, it shall have sufficient complexity to advance the efficiency of autonomous exploration compared to existing open-source strategies found in the literature.

Accordingly, this chapter performs a trade-off study between existing open-source exploration strategies to define the exploration strategy to be implemented in this work in Section 6.1. Given the definition of the desired strategy, Section 6.2, Section 6.3, Section 6.4, and Section 6.5, address the different components of the strategy, mainly, the travel cost, information gain, utility function, and frontier coverage, respectively. Finally, Section 6.6 examines the effect of the ray casting step size on entropy accuracy across the ray casting sphere.

6.1. Exploration Strategies Trade-Off Study

This being the case, the first step is to define, a suitable exploration strategy that can be adapted and improved for implementation in this framework. For this, a trade-off study like the one presented for deciding the state machine framework to be used is performed.

As discussed in the literature review, two open-source exploration strategies stand out. The first is the Autonomous Explorer Node for Frontier Exploration[1], which selects the closest unexplored frontier as the next navigation goal. The second is an exploration strategy based on RRT, where the next frontier point, x_{f_p}, is chosen by balancing the navigation cost, N, and the information gain, I, through a scaling factor, λ, [85]. In the work of Umari and Mukhopadhyay [21], the information gain is defined as the number of unknown cells surrounding a frontier point within a certain radius in the 2D occupancy grid, while the navigation cost is given by

[1] Available: https://github.com/AniArka/Autonomous-Explorer-and-Mapper-ros2-nav2

the Euclidean distance between the rover and each frontier point. A reasonable conclusion is that developing a strategy that also employs a utility function, similar to the work of Umari and Mukhopadhyay, but improved the way either the navigation cost, the information gain, or both, are retrieved, would be beneficial.

Another important limitation of these strategies is that they do not consider the rover's orientation at the frontier point, only mentioning that the rover goes to the frontier point. This is better illustrated in Figure 6.1, where the blue camera FOV captures more information about the frontier cluster, when compared to the red camera FOV.

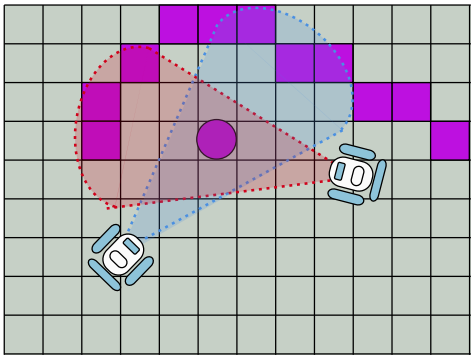


Figure 6.1.: Schematic of the effect of rover orientation at a frontier point on map coverage. An appropriate orientation or FOV range when reaching the frontier can significantly increase sensor coverage and exploration efficiency.

This opens the opportunity to develop an exploration strategy that explicitly accounts for a possible orientation that maximizes exploration efficiency at the frontier point. Given this analysis, and based on the reviewed literature, the following exploration strategies will be compared and evaluated. The first exploration strategy aims to improve the computation of both the navigation cost and the information gain when selecting the next frontier point, following approaches similar to those proposed by Bourgault et al. [47] and Dai et al. [50], named Improved Utility Function. The second strategy, Maximizing Entropy, focuses solely on maximizing the amount of new map area discovered while moving toward a frontier point. The third strategy introduces the consideration of the rover's orientation at the frontier, aiming to maximize sensor coverage and exploration efficiency. This strategy will be called Maximizing Frontier Coverage.

Finally, a Reinforcement Learning–based exploration strategy is included for completeness of the trade-off study, as it has been demonstrated in previous works, such as that of Leong [78]. A set of criteria is also defined to evaluate these strategies.

First is the **Computation Complexity**, *i.e.*, the computational cost of generating and selecting new frontier points. Lower complexity might mean a better algorithm performance on resource-constrained robotic platforms.

Second, the **Adaptability and Parametrization** assesses how easily the method can be tuned or extender. For example, by incorporating more terms in an utility function, or tuning the existing parameters through a scaling factor. This is also important, as one of the goals of this work is to develop a strategy that can be further extended by the community to achieve greater complexity.

The third criterion, **Improvement of State-of-the-Art Exploration Efficiency**, refers to the expected enhance in performance that the exploration strategy will have when compared to existing open-source exploration strategies.

Finally, the **Reproducibility and Open-Source Availability** criterion evaluates the extent to which the strategy can be reproduced or built upon using publicly available literature. This is aligned with the open-source nature of this work and supports future community-driven extensions.

Similarly to the state machine framework trade-off study, each strategy will be ranked on a scale from 1 to 5, with 1 considered the lowest and 5 considered the highest. The importance of each criterion in the work's context is attributed through relative weights as follows:

- Computation Complexity, 0.8. Efficient computation is important for real-time performance, especially on platforms with limited onboard resources. However, slightly higher computational costs can be tolerated if they result in substantially improved exploration performance.

- Adaptability and Parametrization, 0.9. The ability to easily tune or extend the exploration method is crucial for this work, as one of its primary objectives is to develop a strategy that can serve as foundational work to be expanded by the research community.

- Improvement of State-of-the-Art Exploration Efficiency, 1.0. This criterion measures the degree to which a strategy advances existing open-source methods in terms of exploration performance. It is the most important criterion.

- Reproducibility and Open-Source Availability, 0.7. While open-source accessibility is aligned with the goals of this work, it is assigned a slightly lower weight, as it primarily influences long-term research adoption rather than immediate technical performance.

Note that criteria 2 and 4 are distinct. Criterion 2 deals with the question 'Can this algorithm be adapted to different robotic platforms or modified easily?', whereas criterion 4 focuses on 'Can another researcher reproduce, execute, and build upon this work?'. The results of this trade-off study are presented in Table 6.1, where Reinforcement Learning is shortened to RL.

Criterion	Weight	Improved Utility Function	Max. Entropy	Max. Frontier Coverage	RL
Computation Complexity	0.8	4	5	4	2
Adaptability and Parametrization	0.9	5	3	4	5
Improvement of State-of-the-Art Exploration Efficiency	1.0	5	4	5	5
Reproducibility and Open-Source Availability	0.7	5	4	4	2
Weighted Sum		**16.2**	**13.5**	**14.6**	**12.5**

Table 6.1.: Trade-off analysis of advanced exploration strategies suitable for open-source implementation.

From the trade-off study, it can be concluded that the Improved Utility Function strategy achieves the highest overall score, with a weighted score of 16.2. This indicates that it offers the best balance between computational efficiency, adaptability, performance improvement, and open-source reproducibility. Nevertheless, the Maximizing Frontier Coverage strategy closely follows, with a score of 14.6, also scoring the highest with respect to improvements in the state-of-the-art, given the previously mentioned exploration strategies do not account for frontier coverage.

Moreover, the Maximizing Entropy strategy scores lower, with a weighted sum of 13.5, primarily due to its limited adaptability and parametrization. This is expected, as it focuses on maximizing entropy and lacks the flexibility of utility-based methods, which can incorporate multiple factors and, therefore, more easily tuned. Reinforcement Learning scores the lowest, which was expected, given the low score in terms of computation complexity, motivated by inference cost, compared to heuristic methods.

Given these results, both the Improved Utility Function and Maximizing Frontier Coverage strategies are considered suitable candidates for implementation in this thesis. The Improved Utility Function will serve as a baseline approach that refines classical open-source frontier-based exploration by optimizing the way navigation cost and information gain are computed. The Maximizing Frontier Coverage strategy, on the other hand, introduces a novel contribution by explicitly considering the rover's orientation at the frontier to maximize sensor coverage and mapping efficiency.

Consequently, an exploration strategy that integrates both approaches will be developed. This represents an even greater advancement over existing open-source exploration strategies, at it tackles two of the main gaps in current methods: an improved utility function between that balances navigation cost and information gain, and accounting for the rover's orientation at frontier points.

The results of the trade-off study allowed to determine that the implemented exploration strategy shall use an improved utility function that balances travel cost and information gain, while also accounting for the rover's orientation at frontier points. As a result, the first step is to determine how the travel cost and information

gain can be computed in a more accurate way.

6.2. Travel Cost

The travel cost, T_i, can be determined in two ways: by distance or by time. The distance-based approach considers only the Euclidean distance between the rover and the target goal. This can be easily computed as mentioned in Subsection 5.3.3. In addition, if the rover's linear velocity is known, the distance can be converted into travel time using $\Delta t = \frac{\Delta x}{v}$. Although this provides a simple measure of proximity, it fails to capture how the rover's orientation can influence travel time.

This concept is demonstrated in Figure 6.2. The blue centroid is located 15 cm away from the rover and requires a 180° rotation to orient itself correctly. Similarly, the green centroid is positioned at a distance of $20\sqrt{2}$ cm, requiring a 45° rotation. Given the rover's maximum angular velocity of 12.8°/s, maximum linear velocity of 0.13 m/s, and an occupancy grid resolution of 5x5 cm, the corresponding travel times can be determined. For the blue centroid, the rotation time is 14.06 s and the translation time is 1.15 s, resulting in a total travel time of approximately 15.22 s. In contrast, the green centroid requires 3.52 s for rotation and 2.18 s for translation, yielding a total travel time of 5.69 s. In light of this, the green centroid presents a lower travel cost when evaluated in terms of total travel time, yet a higher one when wrongfully considering only the Euclidean distance.

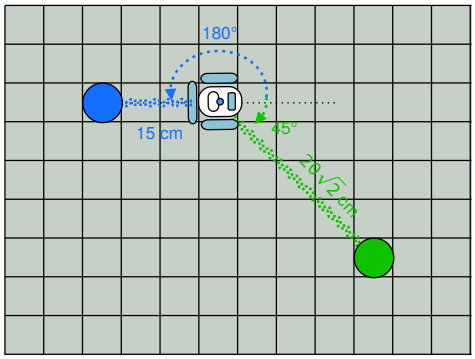


Figure 6.2.: Example illustrating the influence of rover orientation on travel cost computation.

This shows how computing travel cost by taking into account total travel time is more accurate than considering only the Euclidean distance. Even so, some assumptions are made to apply this method. First, it is assumed that the rover is always moves at its maximum linear and angular velocities. Second, the rover's trajectory from its initial position to the centroid is considered to be a straight line. Third, when computing the total travel time, rotation and translation are treated as separate, non-overlapping motions.

Finally, since this strategy is intended to be as plug-and-play as possible, it is

assumed that the stereo vision camera, specifically, the pan-tilt unit, remains fixed, and only the rover itself rotates. Rotating the pan-tilt unit compared to rotating the rover's body frame would yield a lower rotation time. However, not all robotic platforms have this feature. In fact, ongoing work at the LRM focuses on using the IMU data from the Intel RealSense Depth Camera D435i to correctly represent the rotation of the pan-tilt unit within the TF tree in RViz.

Some of these assumptions increase total travel time, while others decrease. It is expected that the rover's linear and angular velocities will vary during motion, and that its trajectory will not be a straight line. The results in Chapter 8 will demonstrate the extent to which the simplified travel cost estimation aligns with the rover's actual travel time. Note that because of this more complex approach, two additional user inputs are necessary. To be specific, the rover's linear and angular velocity.

6.3. Information Gain

The information gain, henceforth denoted as IG, quantifies how much uncertainty is reduced when a robot observes a previously unknown region. This uncertainty is typically represented by entropy H [48]. Hence, a greater information gain corresponds to a larger decrease in entropy, meaning the robot has reduced more uncertainty about the environment. This is given by:

$$IG = H_{\text{prior}} - H_{\text{posterior}} \tag{6.1}$$

where H_{prior} is the entropy before taking a measurement (how uncertain the rover is about the environment) and $H_{\text{posterior}}$ is the entropy after the expected measurement (how uncertain the rovers expects to be after observing). In an OctoMap, each voxel in 3D space has a probability p_i of being occupied as explained in Subsection 5.2.1. The entropy of one cell is:

$$H_i = -p_i \log(p_i) - (1 - p_i) \log(1 - p_i) \tag{6.2}$$

Remembering that a probability of occupancy of 1 means that the voxel is fully occupied, a probability of occupancy of 0 means that the voxel is fully free, and a probability of occupancy of 0.5 means that the state of the voxel is completely unknown, Equation 6.2 will yield 0 if the voxel is known, *i.e.*, its probability of occupancy is either 0 or 1, and yields the maximum value of 1 when the voxel is completely unknown. Thereafter, the total entropy over a region R is the sum of all cells' entropy:

$$H(R) = \sum_{i \in R} H_i \tag{6.3}$$

Therefore, for a potential sensor viewpoint, it is possible to estimate how much entropy would be reduced by observing a region., where the IG is then:

$$IG(x_f) = H(R) - H'(R|x_f) \tag{6.4}$$

where $H(R)$ is the current entropy of the observable region, and $H(R|x_f)$ is the expected entropy after taking a measurement from the frontier candidate x_f. In

practice, since exact measurement prediction is difficult, an approximation is used that considers the number of unknown cells visible from that point within the sensor range of FOV. Thus, a practical approximation used in many papers, such as the work of Dai et al. [50], is:

$$IG(x_f) \approx \sum_{i \in V(x_f)} H_i \tag{6.5}$$

where $V(x_f)$ is the set of cells visible from x_f. That is, the more unknown volume the rover sees, the more information gain it is expected. Accordingly, what is left to do is define the volume V that contains the set of voxels visible from x_f to apply Equation 6.5 and compute entropy and, subsequently, information gain.

Besides, note that the occupancy probabilities in an OctoMap can be expressed either in a full or binary form, where voxels are classified as free or occupied. Because this strategy is implemented in resource-limited rovers, a binary OctoMap is employed. This representation allows each voxel to be directly assigned of value of 0 or 1 and not compute the entropy value for each voxel, through Equation 6.2, which significantly decreases exploration time. This is supported by work in the literature that also use the binary occupancy representation in OctoMap for mapping, navigation, and exploration tasks [126, 127].

6.3.1. Ray Casting Within a Spherical Region

Since the rover is equipped with a stereo vision camera, a spherical region with a radius equal to the range of the stereo vision camera will be used as the volume V that will contain the set of visible voxels. Using the TF tree and the user input names of the camera frame and the rover body frame, it is possible to determine the position of the stereo vision camera when the rover reaches the frontier point. This position will serve as the origin of the ray casting sphere. The ray casting process has already been explained in Chapter 4.

The choice of a sphere as the volume V, is supported by the fact that rovers equipped with pan-tilt units are capable of rotating their stereo cameras to cover a wide range of directions. For instance, if this algorithm is being used on a rover to explore a cave, it might be useful for the rover to look up. With this, the volume V provides the most accurate representation of the observable space, when compared, for example, to just using the camera frustum, 6.3, rotated around the frontier point as the volume V.

Since the volume V is a sphere, spherical coordinates are used to determine the step increments of each ray. In this representation, the usual Cartesian coordinates (x, y, z) are replaced by (r, θ, ϕ), where r is the radius from the origin, θ is the azimuth angle in the xy-plane, and ϕ is the elevation angle measured from the z-axis, Figure 6.4.

Because the range of ϕ is $[-\frac{\pi}{2}, \frac{\pi}{2}]$, and the range of θ is $[-\pi, \pi]$, the azimuth step will always be twice the elevation step to ensure uniform distribution of rays across the entire sphere, even before deciding the actual step values.

A visual representation is useful for explaining how the sphere is positioned within the OctoMap for ray casting. Given this, Figure 6.5 clarifies how this occurs.

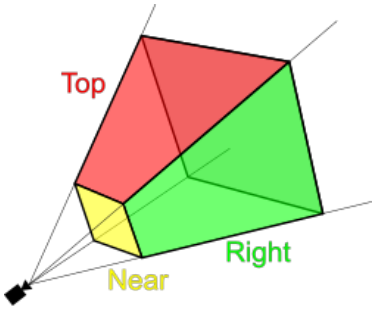


Figure 6.3.: Camera frustum view. *[Credits: Wikipedia.]*

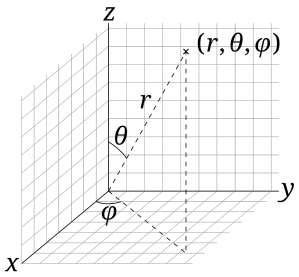
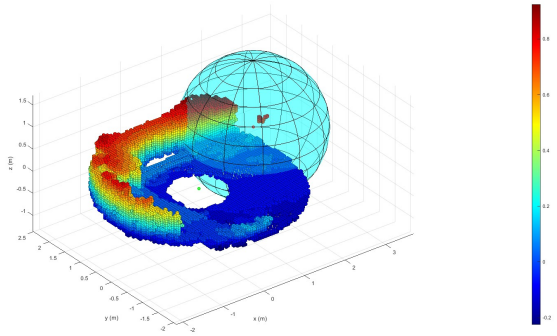
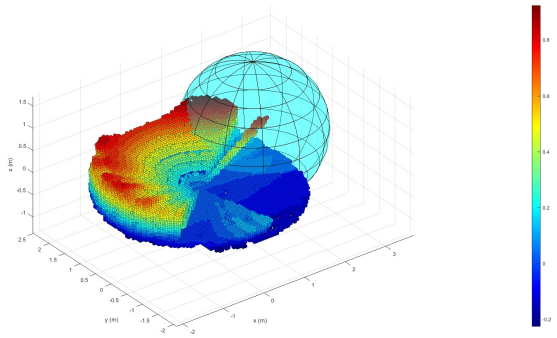


Figure 6.4.: Spherical coordinates representation. *[Credits: Wikipedia.]*

The OctoMap is height-colored from an experimental mission scenario in which the rover, represented by the green dot, performed a 360° rotation. The red dot represents the position of the stereo vision camera after applying the transformation from the respective frontier centroid to the camera frame. A 1.5 m radius sphere is projected around the camera position. Figure 6.5a shows only occupied voxels, while Figure 6.5b depicts both free and occupied voxels. It can be seen that there are voxels that fall within the projected sphere. During ray casting, the rays intersect these voxels, which allows to compute entropy.



(a) Ray casting sphere projected within the OctoMap showing only occupied voxels.



(b) Same configuration as in (a), but displaying both free and occupied voxels.

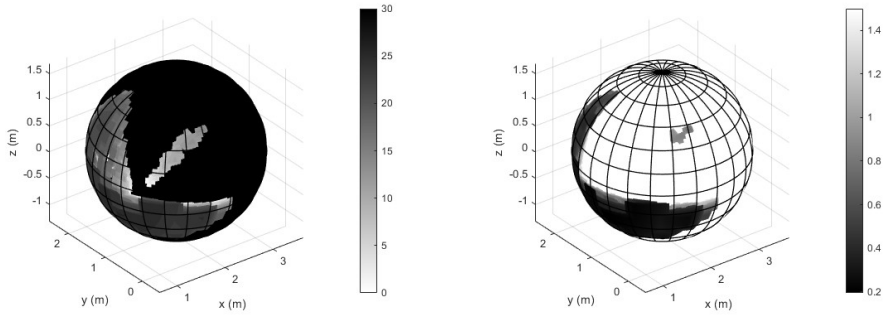
Figure 6.5.: Visualization of the ray casting sphere within the OctoMap. (a) Occupied voxels only. (b) Both free and occupied voxels.

From this, a method is defined to compute the total entropy inside this region in the most computationally efficient way possible. First, the entropy along each ray is computed, and then the entropy of all rays is summed to obtain the total entropy. This is done for all centroid after filtering unwanted frontiers. The centroid with the highest total entropy represents the centroid with the biggest information gain. Ray casting along a ray stops either when an occupied voxel is hit or when the distance between the tip of the ray and the sphere's origin is greater than the sphere radius. Note that with these constraints, the different between using the rotated camera frustum or the sphere as volume V becomes redundant.

In the example presented, the floor is composed of occupied voxels will with stop the rays and the sky is just unknown voxels which will always give maximum entropy along the ray. Note that under these constraints, the difference between using the rotated camera frustum or a sphere as V becomes negligible because occupied

voxels will always stop rays, independent of the shape of V.

By knowing the resolution of each voxel as well as the position of each occupied and free voxel of the OctoMap in the sphere, it is possible to infer the position of unknown voxels in the remaining available space. Now, with the whole volume of the sphere defined, entropy can be computed along each ray. For example, given a 1.5 m radius sphere and a voxel resolution of 5x5x5 cm, there are 30 voxels per ray. This means that, from Equation 6.2, the maximum entropy value along a ray is 30, which corresponds to just having unknown voxels in that ray. Because the number of rays changes with step size so will the total entropy for the same scenario. This technique can also be used to compute a depth from around the sphere, by only considering the distance to occupied voxels.



(a) Entropy map projected onto the spherical surface.

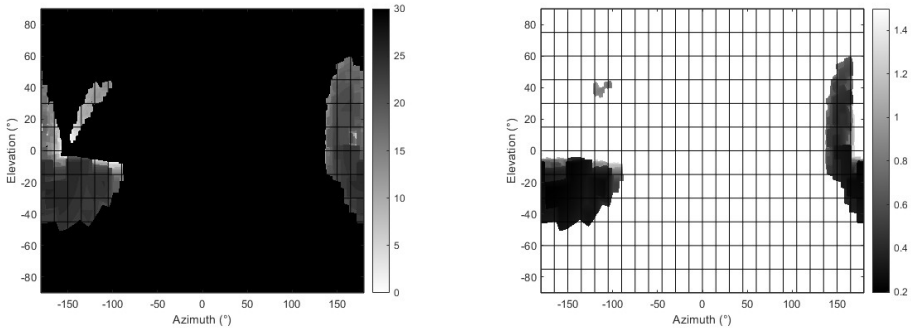
(b) Depth map projected onto the spherical surface.

Figure 6.6.: Projection of the entropy and depth maps from the scenario of Figure 6.5 onto a spherical surface.

Coupled with this, it is possible to compute a depth map an entropy map across the spherical surface of the sphere for each frontier point. These maps are a visual representation of the observed environment by the stereo vision camera at that point. Hence, Figure 6.6 projects the depth map an the entropy map of the same scenario as Figure 6.5 into the spherical surface of radius 1.5 m. Each cell on the spherical surface corresponds to a spherical quadrilateral, defined by the selected azimuth and elevation step sizes. These areas represent the discrete sampling areas of the spherical grid, where the corresponding depth and entropy values are mapped.

In the black-and-white gradient color bar of the entropy map, note that the maximum expected value of entropy, 30, dominates most of the spherical surface, meaning that, in this particular ray casting point, most of the environment is unknown. Furthermore, by comparing the entropy map with the depth map, it can be seen that the entropy map is similar to the depth map but includes additional features. This stems from the fact that the depth map accounts only for occupied voxels, whereas the entropy map considers both free and occupied voxels. For better visualization, the spherical surface can be projected in 2D, as illustrated in

Figure 6.18.



(a) 2D projection of the entropy map.

(b) 2D projection of the depth map.

Figure 6.7.: 2D projection of the entropy and depth maps from the scenario of Figure 6.5.

For this particular ray casting, a step of 0.5° was used for the azimuth, while a step of 0.25° was used for the elevation. Given the range of these angles, a total of 519841 rays were casted, which took 85.282 s for MATLAB to compute, for a total entropy value of 14915612. This represents 95.6% of the total possible entropy of 15595230. This correlates with the entropy map, which is predominantly filled with regions of maximum entropy. Thus, it is clear that attention shall be given to defining the correct elevation and azimuth step sizes that yield accurate ray casting results without compromising performance. This will be addressed in Section 6.6.

Since the maximum entropy value varies according to the step size of the angles it is important that for a set of frontier centroids, the same step size is used, to ensure that the entropy values can be properly compared. For now, since the way travel cost and information gain are computed has already been implemented, the next step is to determine how to balance them through an utility function.

6.4. Utility Function

When considering utility functions for autonomous exploration that balance travel cost and information gain, several approaches can be found in the literature. For one, González-Baños and Latombe [46] used an utility function for a given frontier centroid, x_i, that takes the distance to the centroid, d_i, and the information gain, IG_i, as:

$$x_i = IG_i \exp(-\lambda d_i) \tag{6.6}$$

setting the λ parameter as 0.2. This was later reproduced by Schuster et al. [97] as part of DLR's LRU participation in the SpaceBotCamp Challenge. Other work like the one from Stachniss et al. [123] use a similar approach without relying on an

exponent, where the utility, U_i, of a frontier centroid is given by:

$$U_i = IG_i - \alpha d_i \tag{6.7}$$

However, these utility functions use distance to centroid and not total travel time as the travel cost. A linear weighted difference could also be used, where the utility is given by:

$$U_i = \omega_1 \frac{IG_i}{IG_{max}} - \omega_2 \frac{T_i}{T_{max}} \tag{6.8}$$

where the minus sign expresses the penalty from travel cost. This represents a normalized linear trade-off between information gain and travel cost, where ω_1 and ω_2 are weighting coefficients such that $\omega_1 + \omega_2 = 1$. This might appear to be an easy approach to use at first glance. Yet, tuning the parameters to achieve a proper balance between information gain and travel cost can be challenging and requires careful consideration. Therefore, this approach is not adopted in the present work. However, given that this thesis is intended to be easily extendable by the community, it could be pursued as future work.

Hence, the chosen approach to balance the utility function is based on the work of Dai et al. [50], where the utility of each frontier centroid is given by a ratio between the information gain and travel cost:

$$U_i = \frac{IG_i}{T_i} \tag{6.9}$$

The frontier centroid with the highest utility is the one chosen as the target position goal. This ratio automatically penalizes frontiers that require a long total travel time, while favoring closer frontier with a high information gain, without the need to tune a weighing factor. Another motivation for this choice is that, experimentally tuning a weighing factor instead of in a simulated environment would be very time-consuming, taking time allocated for other critical steps. The implementation of the state machines in RAFCON used to select the next frontier goal according to this method is shown in Figure 6.8.

6.5. Frontier Coverage

The utility function that defines the rover's target position has been implemented. Still, as the trade-off study of Section 6.1 showed, another promising approach that is not present in the literature is to investigate how selecting the optimal orientation or set of orientations, when approaching a frontier centroid could improve exploration efficiency. For this purpose, the information obtained from the ray-casting process will be manipulated. The idea behind this frontier coverage method is to determine, based on the information obtained from ray casting, the range of angles through which the rover shall rotate when it reaching the frontier centroid to maximize additional information gain during coverage.

Of course, if the rover performs a 360° rotation every time it reaches a frontier centroid, it will cover the entire angle range from which information about the

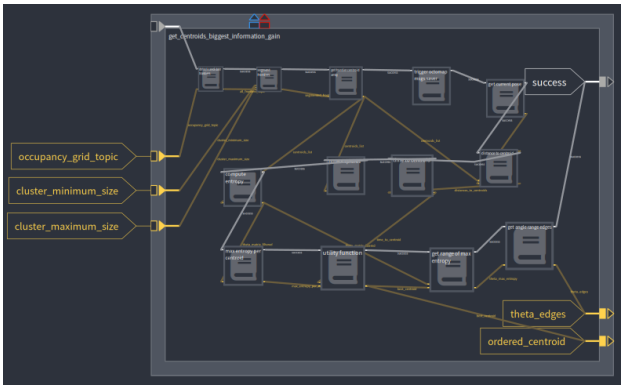


Figure 6.8.: Hierarchical state machine architecture used to compute travel cost, information gain, and the utility function that determines the next position goal.

environment can be gathered. However, this approach is inefficient, as it increases exploration time and redundant exploration without a proportional gain in new information.

At any instant, what is being seen by the stereo vision camera is limited by the camera horizontal and vertical FOV. In the case of the LRM, the Intel RealSense Depth Camera D435i has a hFOV of 87° and a vFOV of 58°, for a diagonal (total) FOV, using Equation 6.10, of 95.8°.

$$FOV_{\text{total}} = 2 \arctan\left(\sqrt{\tan^2\left(\frac{FOV_{\text{h}}}{2}\right) + \tan^2\left(\frac{FOV_{\text{v}}}{2}\right)}\right) \tag{6.10}$$

Hence, for each point in Figure 6.18a corresponding to the center of the stereo vision camera's FOV, an area can be defined that represents what the camera is seeing currently. Figure 6.9 is an example of such representation, where the red rectangle represents the iFOV of the camera. Given that the total observable sphere around the camera covers $180 \times 360 = 64800$ deg^2, and the rectangular angular area seen at any instant is $87 \times 58 = 5046$ deg^2, at any given time, the iFOV of the camera sees, approximately, 7.79% of the total ray casting sphere.

If the entropy values from ray casting are constrained by the camera's instantaneous field of view (iFOV), it is possible to compute an entropy map that, for any given camera position, represents the normalized total entropy within the visible region, *i.e.*, the sum of the entropy values of all rays falling inside the iFOV, as presented in Figure 6.10. This representation allows to determine the camera azimuth and elevation that correspond to the direction of maximum entropy. In other words, it identifies the orientation from which the camera is expected to observe the most information, which corresponds to the region of maximum entropy bounded by the iFOV of the camera. Other than that, Figure 6.9 shows that the rectangle

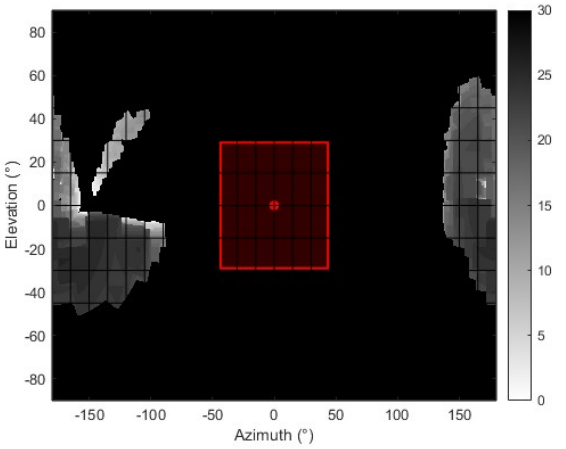


Figure 6.9.: 2D projection of the entropy map with the red rectangle representing the iFOV of the camera.

depicting the camera's iFOV can take several position where entropy is maximum. This is because the 2D projection of the entropy map makes it so that there a lot of maximum entropy regions adjacent to each other.

As a result, the set of optimal orientations at a frontier point used to efficiently maximize information gain through frontier coverage, after applying the utility function to go to the frontier point, comes from defining which regions in Figure 6.10 justify being observed based on their entropy FOV values.

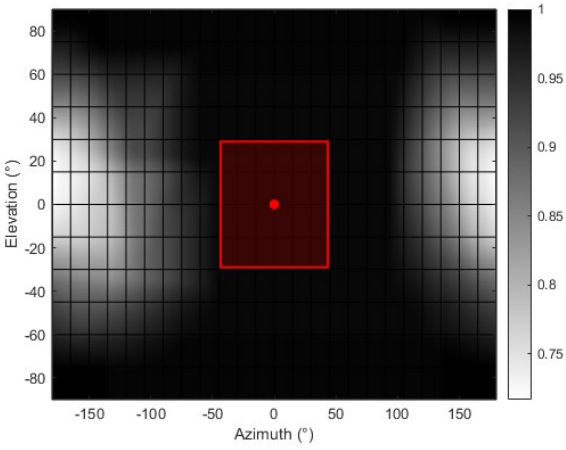


Figure 6.10.: Camera FOV entropy map showing, for each azimuth and elevation, the total entropy sum normalized within the camera's FOV.

Another important consideration is that not all robotic platforms have a rotating pan-tilt unit, which limits the entire entropy FOV map to a band along the 0° elevation, across the full azimuth range. Additionally, in the LRM case, there is no feedback on the RC servos of the pan-tilt unit, and the rotation of the frames in the TF tree is updated using the current time instead of the commanded timestamp. Over time, this difference makes the camera frame in the TF tree unreliable, eventually breaking the visual odometry, as the system misplaces the camera angle. Ongoing work by other students in the project is being carried out to use the IMU data from the Intel RealSense camera to better integrate the rotation of the pan-tilt unit in the TF tree. Consequently, the pan-tilt camera of the LRM is not able to change its elevation without breaking the TF tree.

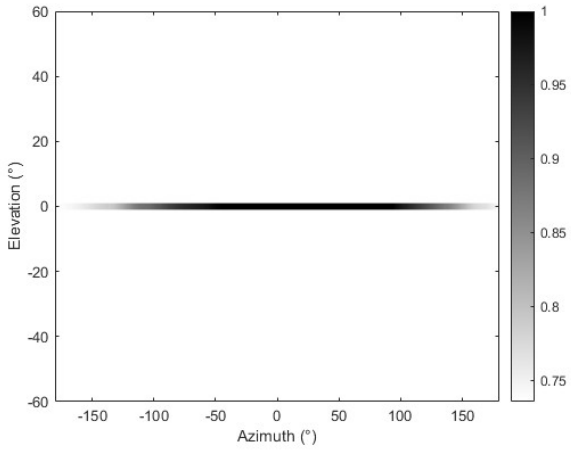
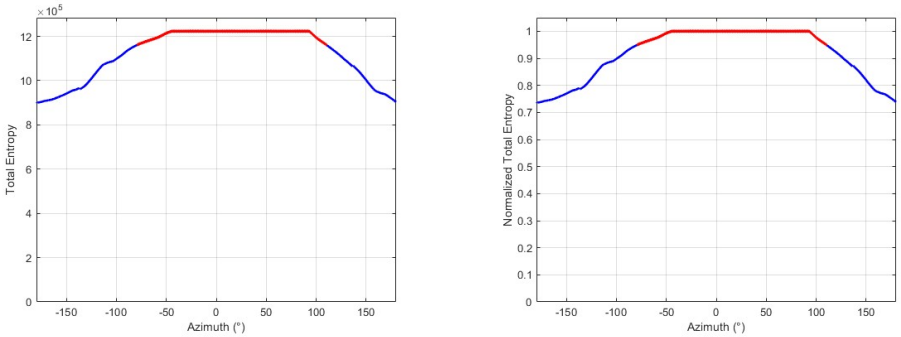


Figure 6.11.: Black-and-white gradient color camera FOV entropy map limited at 0° elevation.

This being said, frontier coverage is limited along the 0° elevation band, corresponding the the stripe of the camera FOV entropy map in Figure 6.10 that corresponds to the 0° elevation. This is illustrated in Figure 6.11. Because there is no elevation variation, the camera FOV entropy can be plotted as a function of azimuth with the actual camera FOV entropy values instead of the black-and-white gradient color, Figure 6.12a. These values can also be normalized, as Figure 6.12b shows.



(a) Actual camera FOV entropy values. (b) Normalized camera FOV entropy values.

Figure 6.12.: Representation of camera FOV entropy along the 0° elevation band as a function of azimuth, with the red dots depicting the azimuth angles above the 0.95 normalized threshold.

The normalized plot is particularly useful because it allows to define a threshold to determine which azimuth angles are worth observing. For instance, a 95% value in the plot indicates that, at that azimuth angle, the camera FOV captures 95% of the maximum possible entropy. For the sake of exploration efficiency, a value of 0.95 was defined as the threshold to which an angle considered worth observing. The choice of this value tries to balance how much meaningful information can be gathered about the environment, without doing redundant frontier coverage. Thus, any azimuth with a normalized FOV entropy above this threshold is considered a candidate orientation. For this particular mission scenario and camera position for a given centroid, the range of azimuth angles worth observing counterclockwise (CCW) is $[-78.5, 110]°$, for a total range of 188.5°. From this, the range of maximum normalized entropy is $[-44, 93]°$.

In Chapter 8, it will be shown that candidate orientations with a normalized value of or greater than 0.95 for frontier coverage always fall within a single range, rather than being spread across multiple separate ranges. This is due to the fact that the camera FOV entropy map ends up masking the individual entropy values of each ray into a sort of continuum, which smooths out difference between azimuth and elevation steps. This phenomena can be observed by comparing Figure 6.9 and Figure 6.10. Consequently, this assumption is made when developing the state machines for frontier coverage.

6.5.1. State Machine Design for Frontier Coverage

The state machine approach for frontier coverage requires several considerations. First of all, the system must known the range of azimuth angles above the desired threshold so that the rover can cover this region. The range is provided in an interval in a counterclockwise direction. The 0° azimuth corresponds to the x-axis in the global map frame. With this information it is possible to also compute the orientation

of the rover, in terms of azimuth, in the global map frame. Two things can occur: the rover's azimuth is either inside or outside the range. A state machines checks which case applies and triggers the corresponding transition downstream.

Further, such state machine, Figure 6.13, is responsible for computing the furthest and closest azimuth from the rover's current orientation. Here, the variable `theta_edges` carries the orientation range to be looked at computed previously at the utility function step.

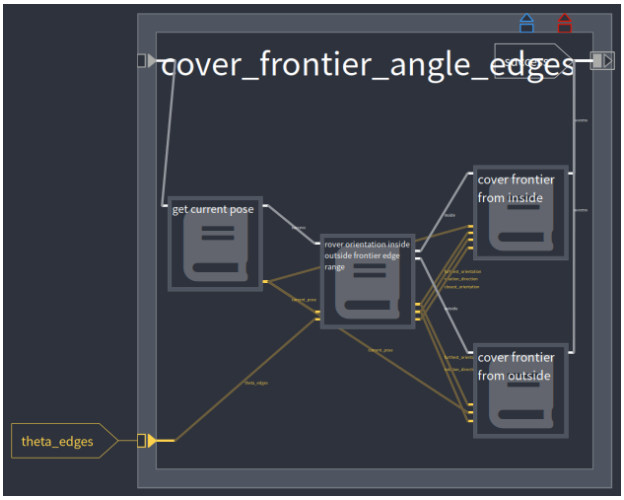


Figure 6.13.: Hierarchical state machine architecture used to compute travel cost, information gain, and the utility function that determines the next position goal.

Cover Range from Inside

On one hand, if the rover's orientation lies between the observable azimuth range, it first turns in the direction of the closest azimuth before changing its direction of rotation to the furthest azimuth. While this is happening, the same state machine responsible for monitoring orientation, presented in Subsection 5.3.3, is employed to trigger the switch in the rotation direction. Since only a rotation is considered here, without any translational motion, there are two possible ways to execute the rotation.

- **Pose goal approach:** The autonomous navigation stack of the rover receives an orientation command publishing a pose goal that orients the rover towards the desired azimuth angle. This approach is more complex but applied to all robotic platforms.
- **Twist vector approach:** The rotation is controlled directly by changing the rover's angular velocity along the z-axis. This approach is specific to the LRM architecture where a twist vector is used as input from the navigation stack.

The twist vector is the standard data format used as input from the navigation stack in the LRM. It consists of two separate vectors for linear and angular velocity, as outlined in Table 6.2.

Parameter	Type	Description
`linear.x`	float	Forward linear velocity along the x-axis
`linear.y`	float	Lateral linear velocity along the y-axis
`linear.z`	float	Vertical linear velocity along the z-axis
`angular.x`	float	Rotation the x-axis
`angular.y`	float	Rotation the y-axis
`angular.z`	float	Rotation the z-axis

Table 6.2.: Components of LRM twist vector used as input for the navigation stack.

Nonetheless, because the rover is not able to move vertically and only rotate in the xy-plane, the parameters `linear.z`, `angular.x`, and `angular.y` have no usage. The twist vector approach for covering the azimuth range when the rover is inside the range is presented below, Figure 6.14.

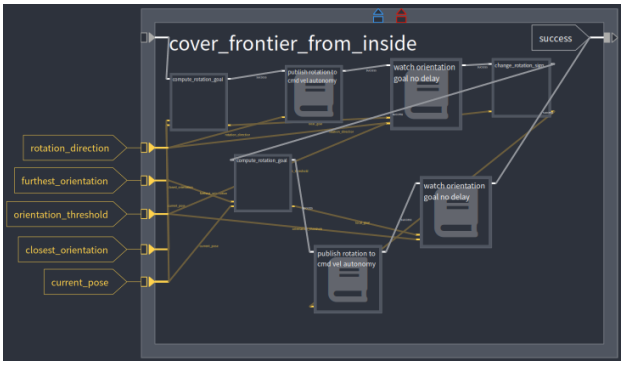


Figure 6.14.: Hierarchical state machine responsible for covering the entire azimuth range of the observable region from the inside.

Cover Range from Outside

On the other hand, if the rover's orientation lies outside the observable azimuth range, the state machine logic is a bit simpler. In this case, the rover just has to to rotate in the direction of the closest frontier, until it reaches the furthest frontier. The condition of rotating in the direction of the closest frontier ensures that the rover reaches the furthest frontier by moving within the observable azimuth range. The twist vector approach when the rover is outside the range is presented below, Figure 6.15.

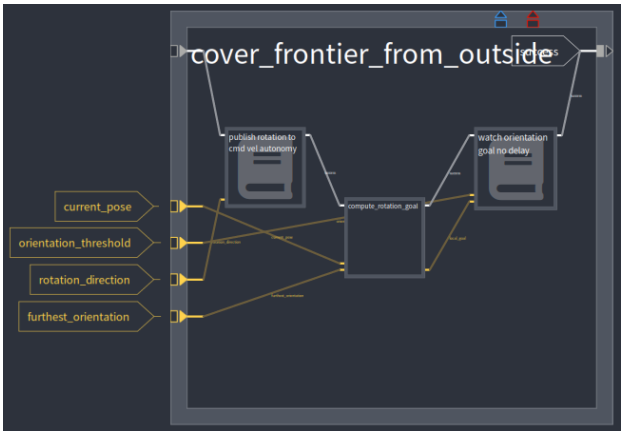


Figure 6.15.: Hierarchical state machine responsible for covering the entire azimuth range of the observable region from the outside.

6.6. Step Size Considerations

An important consideration is the size of the azimuth and elevation steps. Particularly, how they affect the normalized distribution of entropy across the camera FOV. Investigating how the step size influences the information gain used in the utility function to decide the next frontier position goal is more challenging, since the total entropy of the spherical region also depends on the step size, Equation 6.5.

Nevertheless, even though reducing the step size makes the entropy calculation less reliable, Chapter 8 shows that this does not affect the utility raking of possible frontier goals, which is critical for the exploration strategy to work. This analysis is necessary because a very small timestamp significantly increases the computation time of the total entropy because of the increasing number of rays, reducing the overall exploration efficiency of the exploration strategy. For instance, for a 1.5 m radius sphere and an elevation and azimuth step of 0.25° and 0.5°, respectively, the time it takes to complete the ray casting is 296.02 s. For comparison, the computation time to determine the closest frontier is 30.513 ms.

Following this trend, if the range of azimuth angles varies significantly with the step size, exploration efficiency is also compromised. The normalized camera FOV entropy plot for the 0° elevation can be used to verify this by using a very small step size as a reference, performing two checks: first, whether the range of azimuth angles changes, and second, computing the RMSE for plots with larger step sizes. The chosen reference corresponds to an azimuth step of 0.5° and an elevation step of 0.25°, with both step sizes being doubled in each subsequent computation. The normalized entropy results are shown in Figure 6.6.

Moreover, Table 6.3 summarizes the RMSE values computed with respect to the reference, as well as the CCW azimuth ranges for each step size. As expected, an increase in the step size leads to an increase in the RMSE and a decrease on the computation time. However, the azimuth range remain consistent until the elevation

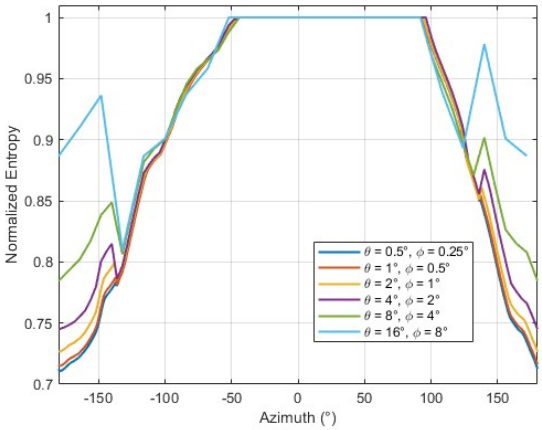


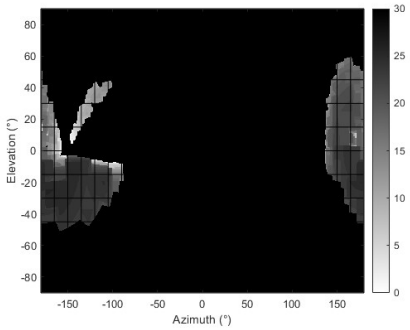
Figure 6.16.: Normalized camera FOV entropy at 0° elevation for different azimuth and elevation step sizes.

and azimuth reach a step size of 4° and 2°, respectively. Too, note that a big enough step size yields more than 1 azimuth range, such as the case of the biggest step size in the table. Another interesting conclusion of using this approach is the fact that even though the computation time decreases from 296.02 s, for the reference, to 1.34 s, for the biggest step size, the RMSE reaches a value of 0.0789. This is considered a small RMSE increase compared to the computation time decrease.

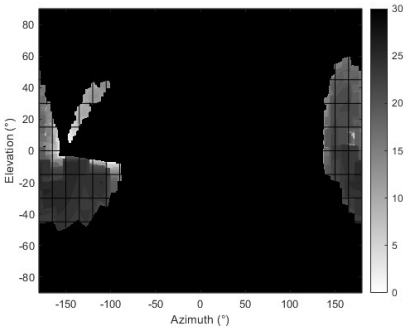
Step Size	RMSE	Computation Time	CCW azimuth range
$\theta = 0.5°, \phi = 0.25°$	-	296.02 s	$[-78.5, 110]°$
$\theta = 1°, \phi = 0.5°$	0.0019	69.67 s	$[-78, 110]°$
$\theta = 2°, \phi = 1°$	0.0074	25.70 s	$[-78, 110]°$
$\theta = 4°, \phi = 2°$	0.0166	7.83 s	$[-80, 108]°$
$\theta = 8°, \phi = 4°$	0.0357	3.14 s	$[-76, 100]°$
$\theta = 16°, \phi = 8°$	0.0789	1.34 s	$[-68, 92]°, [135, 145]°$

Table 6.3.: RMSE, computation time, and the CCW azimuth ranges of the normalized entropy with respect to a reference step size.

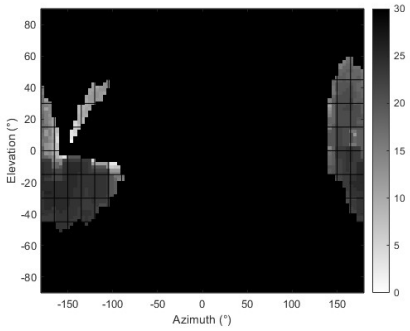
A visual representation of the effect of increasing the step size on the 2D projection of the entropy map is provided below. Based on the analysis of these figures and the values presented in Table 6.3, a step size of 4° for the azimuth, and of 2° for the elevation was selected as it provides a suitable balance between computation time and accuracy to the reference.



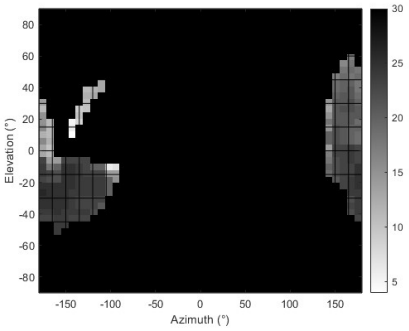
(a) 2D projection of the entropy map ($\theta = 0.5°$, $\phi = 0.25°$).



(b) Camera FOV entropy map ($\theta = 1°$, $\phi = 0.5°$).

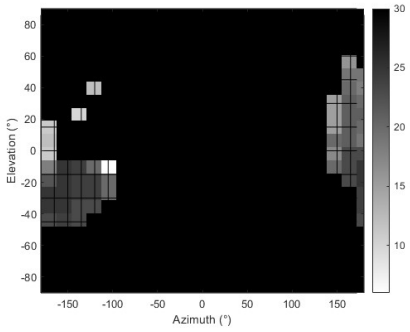


(c) Camera FOV entropy map ($\theta = 4°$, $\phi = 2°$).

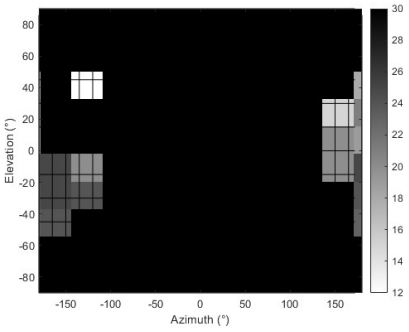


(d) 2D projection of the entropy map ($\theta = 8°$, $\phi = 4°$).

Figure 6.17.: Effect of step size increase in the 2D projection entropy map.



(a) Camera FOV entropy map ($\theta = 16°$, $\phi = 8°$).



(b) Camera FOV entropy map ($\theta = 32°$, $\phi = 16°$).

Figure 6.18.: Effect of step size increase in the 2D projection entropy map.

This chapter presented the exploration strategy that is employed on the LRM and is open-source through RAFCON. The exploration strategy consisted in using an utility function that balances travel cost and information gain to compute the next frontier goal, together with a frontier coverage based on a normalized entropy threshold along the 0° elevation. The selection of the ray casting step size was guided by a trade-off between computation time and entropy estimation accuracy, ensuring that the chosen parameters preserve accuracy while maintaining a computation time compatible with exploration efficiency. With this out of the way, it is possible to move to Chapter 7, where the object detection algorithm and 3D localization pipeline for OoI is discussed.

7

Detection and 3D Localization of Objects of Interest

This chapter is dedicated to explaining how the 3D localization of OoI in achieved within a state machine framework using real time CPU-based inference. For this, Section 7.1 delves into the formulation of the general architecture for object localization, including how camera intrinsic and extrinsic parameters are used to infer 3D coordinates from the 2D camera plane and depth information. Additionally, Section 7.2 investigates how a YOLOv7 model, saved in a PyTorch format, can be converted and deployed for CPU inference on an Intel NUC, as well as the step for training a custom object detection model from a default YOLOv7 model to detect colored cubes as a proof of concept. At the end, Section 7.3 exemplifies how the fully integrated pipeline is used to visualize the estimated 3D coordinates of detected objects in RViz.

7.1. Object Localization Pipeline

The first step in the 3D localization of an OoI is to detect the object. With respect to the LRM, object detection requires determining both the position and class of objects in the 2D plane of an RGB frame captured by the stereo vision camera. The second step involves using the camera's intrinsic and extrinsic parameters to localize the object within the environment overlapping the information from the RGB image with the corresponding frame in the depth image. Therefore, even before choosing the object detection algorithm to be used, it is necessary to characterize the object localization pipeline that makes this implementation possible in RAFCON.

Another important consideration is whether the hierarchical state machine responsible for the object detection shall be implemented within the autonomous exploration pipeline or run separately, in parallel. On one hand, running it in parallel allows a more open-source, plug-and-play solution, not limited to the autonomous exploration framework, but applicable to any robotic system that features 3D localization of OoI. This approach is favored by RAFCON's API, where the execution

of a state machine can be called directly through a Python script. Even so, this solution implies that the OBC, in the case of the LRM an Intel NUC, is capable of handling in parallel, both the object detection, computationally intensive due to inference, and the autonomous exploration algorithm, computationally intensive due to ray casting.

Moreover, as reported in Section 2.5, object detection algorithms typically rely on GPUs for accelerated inference, which are often unavailable on low-cost robotic platforms equipped only with a CPU. This represents a gap in the research, as the performance of CPU-only system for onboard inference remains understudied. On the other hand, integrating the object detection pipeline directly into the autonomous exploration pipeline ensures series execution, potentially reducing overall CPU load but making the object detection pipeline constrained to autonomous exploration applications.

All things considered, it was decided to develop the object localization pipeline as an hierarchical state machine that can be either run independently or be connected via state transitions into the autonomous exploration framework. Subsequently, the computational load will be evaluated to determine the most suitable option. Likewise, it is also possible that running inference on the CPU may take too long to support real-time object detection, rendering this implementation obsolete.

7.1.1. Real-Time Object Detection Threshold Considerations

To evaluate whether the object detection can run real time on the LRM, it is necessary to guarantee that the inference frequency is high enough so that no part of the environment is not seen as the rover moves. This is better observed in Figure 7.1, where the rover rotates CCW and three frames with the respective camera FOVs are shown. Only the middle frame captures the object, so if the inference takes as long as the interval between the first and last frame, the object will not be detected. In other words, given the maximum angular speed of the LRM and its hFOV, the inference frequency of the object detection algorithm must be high enough to ensure that objects entering the camera's FOV are detected before they leave the FOV.

The LRM has a maximum angular speed, ω_{max} of 12.8°/s and the hFOV of the stereo vision camera is 87°. Hence, the time it takes for the camera to rotate across its own FOV is $t_{\text{FOV}} = \frac{\text{hFOV}}{\omega_{\text{max}}}$, which yields $\approx$ 6.8 s. Therefore, to detect the object before it leaves the frame, the inference must run at a minimum frequency of $\approx$ 0.147 Hz. This frequency value appears to be low enough so that real-time object detection is possible even of the CPU. Nonetheless, if real-time inference is not possible, an alternative offline approach will be followed, where the camera frames and the corresponding rover poses are saved, allowing inference to be performed after exploration.

7.1.2. General Architecture for Object Localization

The proposed general architecture for object localization is presented in Figure 7.2. First, a ROS node is started, followed by a TF listener that continuously maintains a

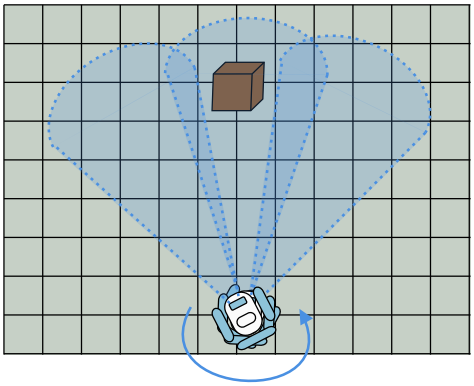


Figure 7.1.: Illustration of the real-time object detection challenge on the LRM Intel NUC.

buffer of the transformation between the rover's frames. The TF listener is stored in the global variable manager, a RAFCON capability that allows other state machines within the framework to access the transformation data in real time. Following this, is a hierarchical state machine that pairs and saves the closest matching timestamps from both the RGB and depth image frames to a local folder on the OBC. Moreover, it stores the pose of the rover's camera at that timestamp as a variable in RAFCON to be passed to another state machine, as it is a necessary extrinsic parameter used to compute the coordinates of the OoI in the 3D global map frame.

Thereafter, a state machine runs CPU-based inference on the RGB image frame. If at least one object is detected, the state machine outputs the bounding box coordinates and object class, then transitions to another one responsible for computing the object's 3D coordinates, checking whether the object is new or already known, and publishing this information to a ROS topic while visualizing it in RViz, through the `detected_objects` ROS topic.

Publishing to a ROS topic opens the possibility of integrating the detected object information with other nodes in the robotic platform. Then, because the RGB and depth image frames were saved in a folder for post-processing, they are deleted after each iteration to free up storage. By contrast, if no object is detected, a transition occurs to a state machine that deletes the existing RGB and depth image frames, and then the object detection loop is repeated.

Note that two additional user inputs, the ROS topic names for the RGB image and depth image, are required. Not only that, the state machine responsible for performing inference depends on other third-party packages to operate. Specifically, the official GitHub repository implementating YOLOv7 [122] and Intel OpenVINO, a toolkit for optimizing and deploying learning models on Intel hardware[1]. Their need is justified in Section 7.2, while the other state machines aforementioned are now examined more in detail.

[1]The GitHub repositories for the YOLOv7 implementation and OpenVINO can be found here.

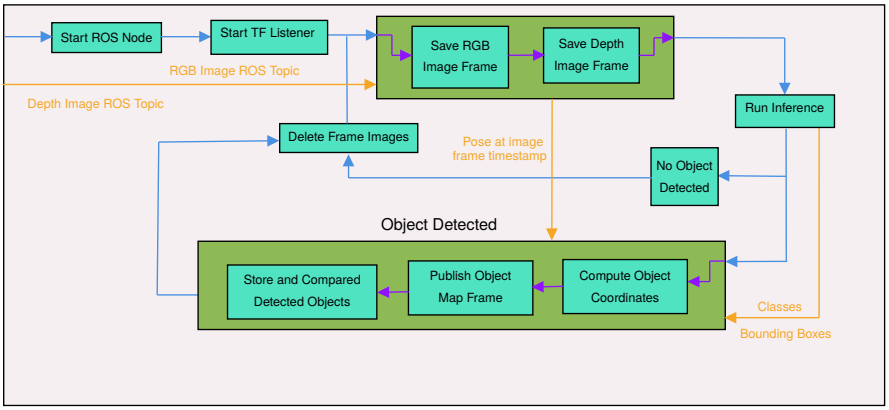


Figure 7.2.: General architecture for object localization.

Timestamped Camera Frames

For this state machine, both color and depth frames from the camera are captured and saved for subsequent processing. Two folders are created to store the images: one for the color image frame and one for the depth image frame. In the LRM case, it uses the messages from the ROS topics `camera/color/image_raw` and `camera/aligned_depth_to_color/image_raw` to obtain synchronized color and depth image frames. However, these names can be overwritten by the user inputs. The aligned depth image is used instead of the raw depth image to ensure that each depth pixel corresponds exactly to the same pixel in the color image frame.

This is necessary because the stereo vision camera is composed of a RGB camera and a depth camera, which do not share the same intrinsic parameters. The `aligned_depth_to_color` topic performs this transformation automatically.

This enables reliable 3D localization of the detected objects and simplifies subsequent image processing. Both use the `sensor_msgs/Image` message type.

At the same time, a `tf.TransformListener` is used to get the camera's pose relative to the map frame with the same timestamp the images were retrieved and store it in an output variable. The ROS messages for the color and depth image frames are converted to OpenCV[2]-compatible formats using NumPy arrays, and each image is saved as a `.png` file. An example of a color image alongside its corresponding depth image is presented in Figure 7.3.

Because the aligned depth image is being used, the depth values are transformed from the depth camera frame to the color camera frame of the stereo vision camera. By overlapping Figure 7.3b onto Figure 7.3a, it is clear that the depth and color images are properly aligned, with the depth information outlining the correct position

[2] OpenCV (Open Source Computer Vision Library) is a widely used, open-source software library that provides tools for computer vision and image processing.

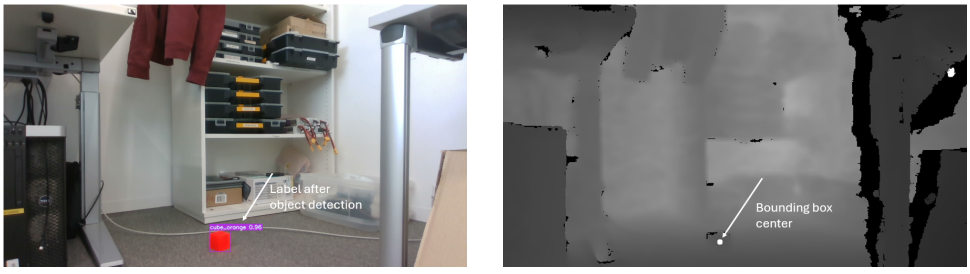
of objects in the depth image relative to their location in the color image.

(a) Color image. (b) Depth image.

Figure 7.3.: Same timestamp color image (a), and depth image (b) from LRM's stereo vision camera.

Localization in Global Map Frame

This state machine is triggered when at least one object is detected after inference, and its purpose is to localize the detected objects in 3D space using the depth information, bounding boxes, and the camera's intrinsic and extrinsic parameters at the time the image frames were captured.



(a) Color image. (b) Depth image.

Figure 7.4.: Color image showing the bounding box after inference (a), and the corresponding depth image showing the 5×5×5 cm cube and the corresponding bounding box center used to infer Z_c (b).

To begin, the center point of the bounding box, (u, v), is calculated, as this will serve as the reference to retrieve the depth value, Z_c, from the depth image. In practice, the center of the bounding box will correspond to the true center of the detected object in the image. This can occur due to object orientation, irregular shapes, and the fact, without knowing the entire object's geometry, the measured depth value corresponds to the surface facing the depth camera rather than the object's true center.

However, for small objects like the ones the LRM will localize, this difference is typically negligible. To mitigate this, the depth values are averaged within a small

3x3 pixel window centered around (u, v). A small 3x3 pixel window was chosen to prevent from including pixels that do not belong to the object, particularly if it is a small object or is very far away. The bounding box resulting from inference in the color image, and the center of the bounding box used to infer Z_c are presented in Figure 7.4. For this particular case, a 5x5x5 cm orange cube was detected, and the accuracy of the (u, v) position can be validated from looking at the depth information that clearly outlines the cube.

The coordinates (u, v) can be back-projected into the camera coordinate system as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{7.1}$$

where K is the camera intrinsic matrix defined as:

$$K = \begin{bmatrix} f_c & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

where f_x and f_y are the focal lengths, while c_x and c_y are the principal point. These parameters can be retrieved by echoing the ROS topic `/camera/color/camera_info`. Why this is the case and their proper definition as been previously explained in Chapter 4. Expanding Equation 7.1 gives:

$$X_c = \frac{(u - c_x) \cdot Z_c}{f_x}, \qquad Y_c = \frac{(v - c_y) \cdot Z_c}{f_y}, \qquad Z_c = Z_c \tag{7.3}$$

Now, using the camera's pose that was previously saved as the extrinsic parameters describing its position and orientation in the world frame and Equation 7.4, the 3D coordinates of the object, (X_w, Y_w, Z_w), can be determined in the global map frame.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = R \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + t \tag{7.4}$$

where R is the 3x3 rotation matrix and t is the 3x1 translation vector. Merging Equation 7.1 and Equation 7.4 leads to:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = R \cdot Z_c \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} + t \tag{7.5}$$

This equation summarizes the calculations performed by the state machine to transform the pixel coordinates and depth value into 3D world coordinates. The importance of using the camera pose corresponding to the timestamp when the images were captured rather than the current pose, lies in the fact that inference occurs between capturing the image and getting the object's 3D coordinates, and the rover keeps moving during this time which would lead to inaccuracies.

Handling New and Previously Detected Objects

A potential issue that may arise concerns how the 3D localization pipeline differs between detecting a new observed object or one that has already been seen. Since the class and 3D coordinates of objects are stored in the `detected_objects` ROS topic, the information in this topic can be used to tackle this issue.

A state machine is created to verify, after computing the 3D coordinates of the most recent detected object, whether an object of the same class already exists. This is done by comparing the Euclidean distance between objects of the same class and checking if it falls below a certain threshold. The threshold is necessary because inference runs in parallel with autonomous exploration and, as the rover moves, odometry drift gradually increases, which will change the estimated position of the object in the global map frame. To account for this, a threshold of 25 cm was defined. Chapter 8 concludes that this level of drift is not concerning for the mission scenario considered.

The main motivation of knowing with accuracy the object's positions it to enable future autonomous grasping for a fully autonomous mission pipeline. This will be possible because the proposed pipeline provides the grasping with a reliable region where each object is. Once there, the rover can disregard global odometry drift and rely on its local perception to know where the object is. However, this concerns future work and is beyond the scope of this thesis. Because the first instance the object was detected corresponds to the instance with lowest odometry drift due to lowest exploration time, the newly computed coordinates of a previous known object do not overwrite the previous ones.

This also opens up the possibility of creating a state machine that checks if a given object class has already been detected and implement it in the general architecture of autonomous exploration described in Section 5.2. This allows to easily guide exploration to find and stop when a desired object is detected, showing the modularity of using a state machine approach for autonomous exploration.

7.2. Implementation of YOLO-Based Object Detection

The only element left to explain is the implementation of the chosen object detection algorithm. Selecting the object detection algorithm did not require a trade-off study, as the literature widely recognizes YOLO as the state-of-the-art solution for real time object detection due to its balance between speed and accuracy, with plenty of supporting documentation[3]. Its theoretical foundations are laid out in Chapter 4.

Given the availability of an open-source GitHub repository for YOLOv7, as well as DLR's internal training toolkit for creating and training custom YOLOv7 models, this version was selected for implementation. Nonetheless, other models can be used as long as the specific state machine responsible for running inference and retrieving class and bounding box information is properly adapted.

[3]This includes both an official Ultralytics website and Ultralytics GitHub repository.

7.2.1. Custom Model Training

Based on the GitHub repository, the default YOLOv7 model is trained on the COCO[4] dataset (Common Objects in Context) and includes 80 object classes, covering a wide range of everyday objects such as bicycles, umbrellas, laptops, and teddy bears. In addition to its main task of object detection, YOLO is also able to perform image segmentation, which assigns a class label to each pixel in the image, classification, which predicts the overall class of an entire region, and pose estimation, which identifies the orientation of objects. Nonetheless, going beyond object detection is not considered here, as it would be time-consuming and falls outside the scope of this thesis.

In a mission environment, the objects of interest are more specific and do not correspond to any of the 80 classes. A more realistic object class would be rocks. Nonetheless, DLR's training toolkit initially requires training on a `.ply` file which is difficult to generate for rocks with different shapes, sizes, and colors. Further, given the similarities between such rocks and the surrounding environment, more knowledge in deep learning and model training would be required to properly train a detection model. Thus, it was decided to train a custom YOLOv7 model to detect 3D-printed 5x5x5 cm colored cubes as a proof of concept for the object detection pipeline.

Figure 7.5.: Proof of concept 5x5x5 cm colored cubes for custom YOLOv7 model.

Six different colored cubes were chosen to train the custom model, Figure 7.5. DLR's training toolkit automates creating the training dataset by first generating random images of the objects from the `.ply` files. These objects are placed in scenes with other objects and different backgrounds. The toolkit also automatically generates all bounding box annotations and depth information for each object in

[4]More information about the COCO dataset is available here.

every image. This is extremely useful since generating the data takes much longer than training the custom model. An example of the created synthetic data created is presented in Figure 7.6.

(a) Train batch 3. (b) Train batch 9.

Figure 7.6.: Examples of synthetic training data batches used for custom model training.

After a sufficient number of synthetic training data was created, the default YOLOv7 model was custom trained using one of DLR's GPU cluster for 40 epochs, meaning the entire training dataset was passed through the network 40 times, allowing the model to iteratively refine its internal parameters and improve prediction accuracy. The training toolkit also generates test data batches to validate the custom model training that show inference results with predictive bounding boxes, labels, and level on confidence, where as example is presented in Figure 7.7.

Moreover, the custom trained YOLOv7 training results are summarized below in Figure 7.9, over the 40 epochs. Their deep analysis goes beyond the scope of this work. Nevertheless, a brief assessment is made. Overall, the model converges with decreasing losses and increasing detection accuracy metrics. More specifically, regarding training losses for bounding box, objectness and classification, all decrease sharply during the first 10 epochs. A rapid decrease in bounding box loss shows that the model quickly learns to predict the object position, while the decrease in objecteness loss and classification loss, suggests improved discrimination between object and background and effective learning to assign correct class labels to detected objects, respectively. The validation losses for each mirror the same trend.

With respect to precision and recall, the precision curve rises sharply within the first few epochs. This indicates that the vast majority of predicted bounding boxes correspond to real objects. The recall curve shows more fluctuation early in training but also ends up stabilizing around 0.8. The mean average precision

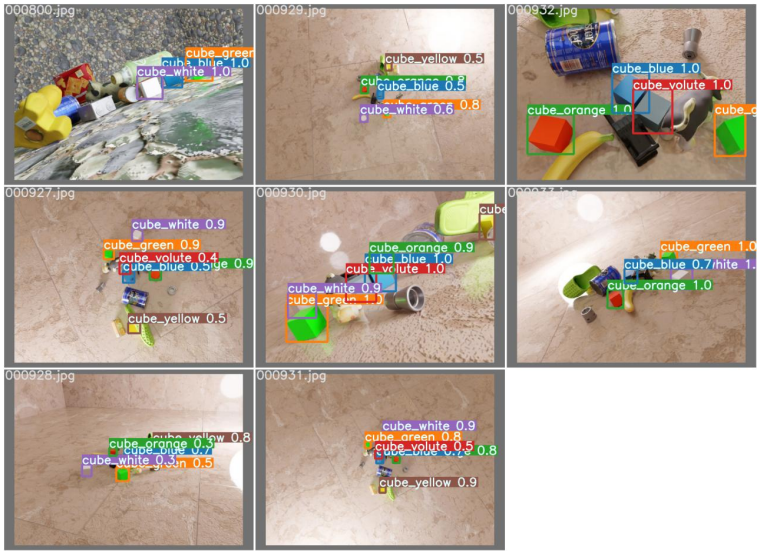


Figure 7.7.: Example of YOLOv7 validation output on a test data batch.

(mAP) is a widely used metric for object detection that combines both precision and recall across all classes, where a higher value is better, presented in Figure 7.10. mAP@0.5 measures the average precision when the predicted bounding boxes have an Intersection over Union (IoU) threshold of 0.5 with the ground truth. mAP@0.5:0.95 is a stricter metric that averages the mAP across multiple IoU thresholds from 0.5 to 0.95 in steps of 0.05. Both exhibit strong upward trends.

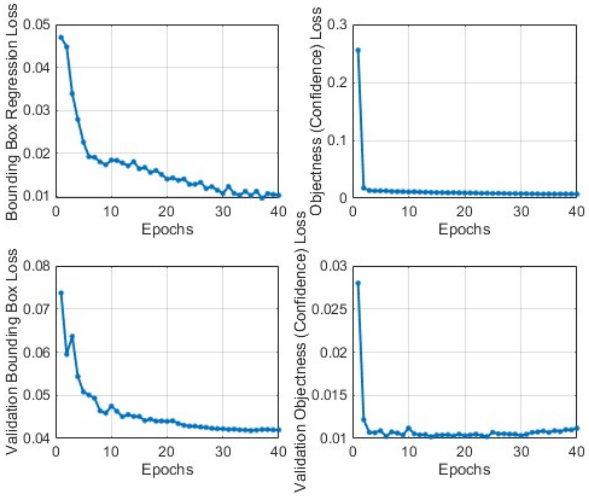


Figure 7.8.: Custom YOLOv7 model training results 1.

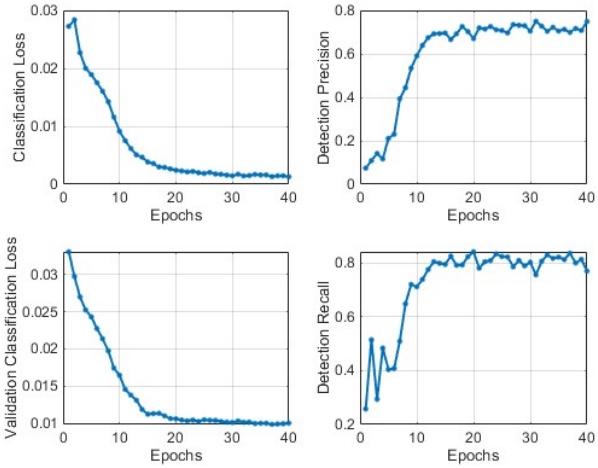


Figure 7.9.: Custom YOLOv7 model training results 2.

In fact, it was expected that the custom YOLOv7 model training presented no issues, since despite DLR's training toolkit pipeline combining several objects and background, the cubes have sharp edges and vivid colors that make them easily distinguishable.

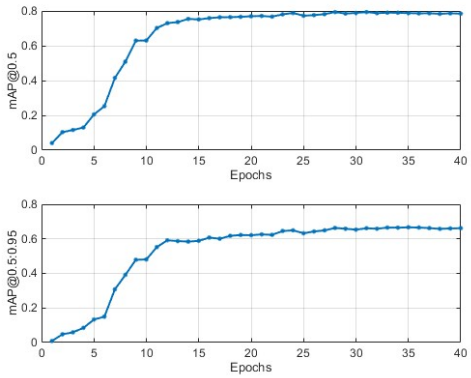


Figure 7.10.: Mean Average Precision training results for the two common variants.

After training, the custom YOLOv7 model is saved as a PyTorch file, the deep learning framework underlying YOLOv7. This file contains the model's learned weights as well as its architecture, allowing the trained network to be used for inference or further fine-tuning. Knowing that the experimental validation of this work will be done at the PEL laboratory at DLR Oberpfaffenhofen, a site designed to mimic lunar soil [128, 129], Figure 7.11, the custom YOLOv7 model can be validated

in a real-life scenario by capturing images of the cubes in this environment and running inference on a desktop equipped with a GPU. So, Figure 7.12 shows the validation of the object detection algorithm in a a lunar-analog site.

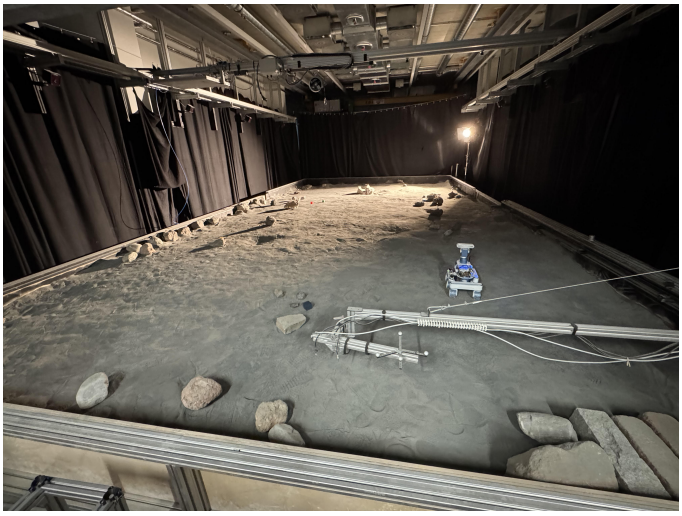
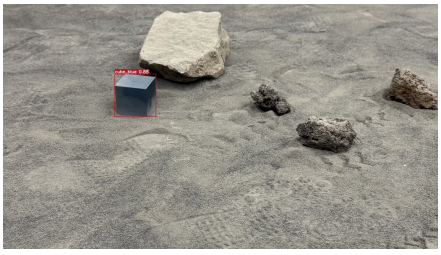
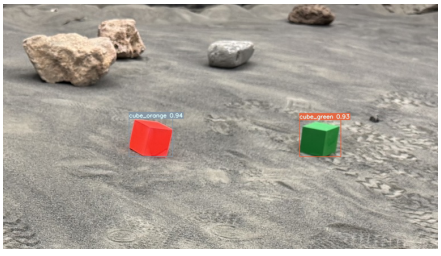


Figure 7.11.: PEL laboratory at DLR Oberpfaffenhofen.



(a) Scenario 1.

(b) Scenario 2.



(c) Scenario 3.

Figure 7.12.: Real-life scenario validation of object detection algorithm.

Now that the custom YOLOv7 model has been trained and validated, it is possible to move towards its implementation within the Intel NUC.

7.2.2. CPU Inference Deployment

To deploy the YOLOv7 model for CPU inference on an Intel NUC, the first step involves converting the PyTorch model, saved in the `.pt` format, into a format that is optimized for Intel hardware. The motivation for this is twofold. First, the `.pt` file is a PyTorch checkpoint, which stores the model architecture and weights. Pytorch is designed primarily for GPU acceleration, so when a model with this format runs on a CPU, it does not automatically take advantage of the low-level CPU optimizations.

As a result, inference can be significantly slower and consume more memory than necessary. Morever, DLR provides a Portable Computer Vision Pipeline (PCVP) with the option of running YOLOv7 directly. However, this approach has two limitations: it is not open-source, and the YOLOv7 implementation in the PCVP is designed to run only with CUDA on NVIDIA GPUs. Therefore, it is clear that the model must be converted to a CPU-optimized format.

The OpenVINO toolkit provides a workflow for this purpose. Matter of fact, documentation on how to convert and optimize YOLOv7 with the OpenVINO toolkit is available here. Thus, the steps described here were followed in the following way. The PyTorch model is converted into the Intermediate Representation (IR) format, which consists of an XML file describing the network architecture and a BIN file containing the learned weights. Once the model is converted, the OpenVINO runtime is used to load and execute it on the Intel NUC. More technically, the runtime abstracts the hardware specifics and allows the same code to leverage various CPU instruction sets, such as AVX2 or VNNI[5], which can significantly improve inference speed. In practice, the input image is preprocessed to match the model's expected dimensions and normalization, then passed to the OpenVINO InferRequest object.

The accuracy of the quantized model can also be compared with the original one. Nonetheless, this step is unnecessary if the model is able to meet the desired performance on the target CPU. For this, the pre-inference images of Figure 7.12 were used to run inference locally on the Intel NUC of the rover executed through a state machine in RAFCON. The results were similar which validates the model running on the CPU. At this point, the full loop for object detection implemented in RAFCON is provided in Figure 7.13. A timer variable is used in case the user whishes to wait a certain amount of time between each run of the object localization loop. However, real time object detection is obsolete if the 3D localization pipeline runs at a lower frequency that ≈ 0.147 Hz, as analyzed in Subsection 7.1.1. This point will be discussed next.

7.2.3. Real Time Inference Validation

The most effective way to validate real-time inference, *i.e.*, ensuring that the Intel NUC can perform inference fast enough to process every frame without skips, is to

[5]Information about Intel AVX2 and Intel AVX2 VNNI can be found here.

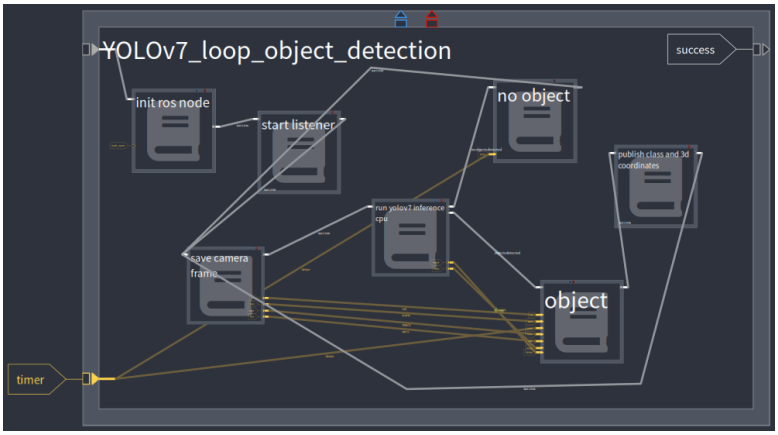
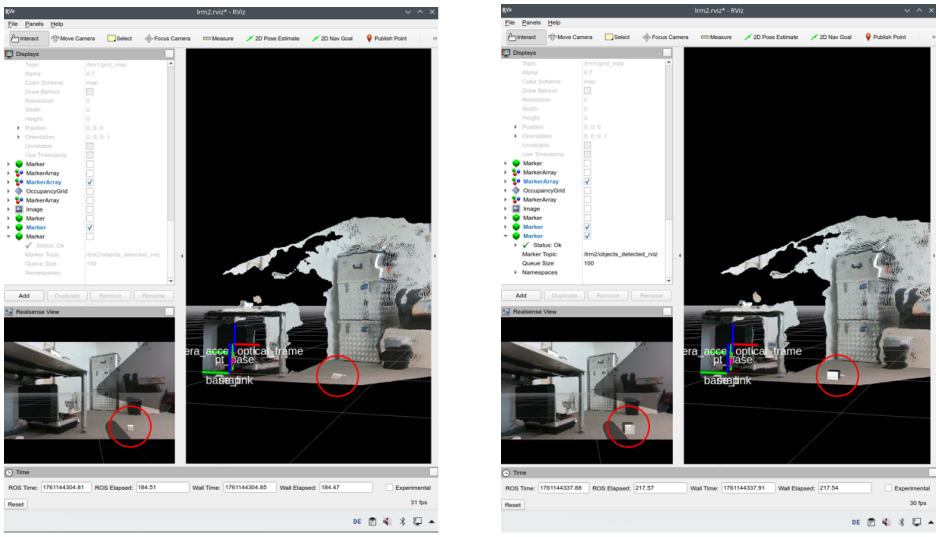


Figure 7.13.: Hierarchical state machine loop for 3D object localization.

measure the execution time of each iteration within the object localization loop. For this purpose, the time of both the inference and the complete object localization loop, including saving the color and depth frames and publishing the 3D coordinates of the detected object into RViz, are averaged over 10 iterations. If the average execution time per loop is below 6.8 seconds, real time inference is achieved. The averaged inference time was 4.4596 s, while for the complete loop was 5.81278 s. This value is lower than the 6.8-second threshold. Therefore, real time inference is possible and validated. This is bolstered by the experimental work where the rover did not miss any object detection, Chapter 8.

7.3. Fully Integrated Pipeline in RViz

It is important for researchers using this pipeline to visualize, in real time, the 3D coordinates of the objects computed by the rover within RViz. As previously mentioned, a ROS topic is responsible for storing the classes and 3D coordinates of detected object. A state machine can then use this information and publish it in the global map frame in RViz, using different colors or shapes to distinguish between objects. For instance, in this case, where different colored cubes are used, an if statement can change the color of the published markers according to the class of the object. This is possible because the model is trained to detect specific objects, meaning that the object classes are known a priori. This also helps to validate the accuracy of the object localization. Consequently, Figure 7.14 illustrates how the research would perceive the fully integrated pipeline in RViz, with a white cube being published, on the right, given the class and 3D coordinates of the detected object, also a white cube placed on the floor, clearly seen on the left.



(a) Reference cube placed on the floor pre inference.

(b) Detected object published in RViz with the computed 3D coordinates post inference.

Figure 7.14.: Visualization of the fully integrated pipeline in RViz. The hierarchical state machine publishes detected objects with their computed 3D coordinates (b).

8

Experimental Validation and Results

This chapter presents the experimental validation of the exploration strategy and 3D object localization pipeline developed. The chapter begins by detailing the experimental setup, including the testbed configuration, assumptions regarding rover operations, and how data is acquired and saved during testing in Section 8.1. Also, the parameters that serve as benchmarks to evaluate the performance of the implemented exploration strategy. Next, the standard operating procedure is outlined to ensure reproducibility across all experiments, followed by a description of the three experimental sets in Section 8.2.

The first concerns evaluation the exploration efficiency of the implemented strategy compared to available ones in the literature, the second one to validate the 3D object localization pipeline, and third the final integration testing combining exploration and object detection. Finally, Section 8.3 concludes this thesis work by presenting the results obtained from these experiments, providing post processed data with respect to the rover's performance, exploration efficiency, 3D localization accuracy, and computational load.

8.1. Experimental Setup

The experiments to validate the performance of the exploration strategy, coupled with object localization, will be conducted at the DLR's Planetary Exploration Laboratory (PEL), Figure 7.11, already partially described in Section 7.2. The laboratory features a 5x10 m testbed for navigation tasks, with an analog lunar soil surface that can be modified for different mission scenarios. Additionally, rocks are available that can be placed to serve as obstacles for the rover. A control room is used to monitor the rover's behavior.

8.1.1. Testbed Configuration

The first task is to decide the configuration of the testbed that the rover will need to explore. This configuration shall have obstacles, hills and depressions. Even though the autonomous navigation capabilities of the rover have been tested as part of another master thesis work [33], this was only demonstrated as part of one mission scenario of simply moving from point A to point B. As a results, having the rover exploring the PEL fully autonomously matures the autonomous navigation capabilities of the rover. It will be very important to understand the success rate of the rover when avoiding obstacles in different configurations, for example. Moreover, evaluating whether the small dimensions of the rover might a pose a challenge when traversing certain parts of the terrain is also valuable. Having all of this in mind, the following testbed configuration was designed, Figure 8.1.

Figure 8.1.: Chosen PEL configuration as part of the experimental setup at DLR Oberpfaffenhofen.

In this configuration, rocks are scattered throughout the testbed, while the colored cubes are randomly placed. The level of environmental complexity is considered high, as most autonomous exploration strategies found in the literature, particularly open-source approaches, are typically indoor simulated environments, where corridors represent the only obstacles. Such environments are less complex because point-like obstacles such as the rocks present in this testbed are harder to interpret by the autonomous navigation stack. It is expected that the local mapping system will recognize that larger rocks are non-traversable and appropriately classify them as obstacles, based on the work of Ricardez Ortigosa [33].

8.1.2. Rover and Testbed Miscellaneous Assumptions

As previously mentioned, the testbed has dimensions of 5x10 meters. The effective range of the camera used for visual odometry is approximately 5 meters. This implies that if the rover is positioned at the center of the testbed and performs a full 360° rotation, it is able to map the entire environment. In practical terms, this means that independent of the starting point of the exploration, only a small number of centroids need to be explored for the rover to map the entirety of the PEL. For this reason, the maximum range of the camera used to track features for visual odometry was limited to 2 meters. Further reduction of this range would negatively impact the visual odometry performance, as it would result in fewer detectable features to track.

Additionally, it will be ensured that the black curtains are closed. Although they will still be detected as obstacles by the stereo vision camera, their uniform appearance makes them comparable to a sky or space-like background, which provides a more realistic environment for the rover. They also contain fewer visual features to track compared to the walls and surrounding objects behind the curtains. This means that leaving the curtains open would artificially inflate the accuracy of the SLAM.

With respect to lighting, the PEL is equipped with a powerful light source designed to mimic sunlight. However, the goal of this rover is to serve as a modular, open-source robotic platform for students and researchers, to be used in different environments and lighting conditions. Therefore, although future work could investigate how lighting affects the autonomous navigation stack of the system, the experiments in this study will be conducted with the building lights turned on and the high-intensity light source switched off.

A final consideration concerns the trails left by the rover after each exploration run. Between tests, these track marks will be erased using a broom. This is necessary because the marks can also serve as visual features for the SLAM system. As the number of tests increases, so does the number of visible trails, which could provide additional, non-representative features and thereby artificially inflate the SLAM accuracy, similar to the effect that is expected to happen by leaving the curtains open.

State Machine Configuration Parameters

A set of parameters must also be defined, as they determine the behavior of certain state machines within the overall mission hierarchical state machine, introduced

during the implementation of the low-level state machines in Section 5.3. This is different from the user inputs necessary for compatibility with other robotic platforms. For instance, for position and orientation monitoring, Subsection 5.3.3, a position threshold of 8 cm and an orientation threshold of 4° where chosen, respectively. What is more, regarding the exploration completion checks, Subsection 5.3.4, the area to cover threshold was set to 10^6 m^2. This is because exploration is intended to stop when the PEL has been fully explored, *i.e.*, there are no more frontier centroids to visit. The cluster minimum and maximum size were set to 30 and 40, respectively. Also, since the maximum range of the camera is 2 meters, the ray casting radius shall be of the same value.

8.1.3. Data Acquisition

All experimental data are recorded using the ROS `rosbag` to enable post-processing and reproducibility of the results. The `rosbag` tool is a message recorder within the ROS framework, storing all desired published topics during runtime into a single file with the `.bag` extension. Each message is saved with its original timestamp. Once the data have been recorded, the `rosbag play` command can be used to replay the experiment. When executed, it republishes all the recorded topics with their original timing, effectively simulating a live system. This capability makes it possible to visualize the recorded data in RViz, as is happened in the experiment for further post processing and performance evaluation.

8.1.4. Benchmarks for Performance Evaluation

The remaining step in the data acquisition process is to specify the ROS topics to be recorded. These topics must be selected such that the recorded data provide sufficient information to evaluate the performance of the designed exploration strategy itself, as well as to enable comparison with other exploration approaches reported in the literature that will also be implemented on the rover. On that account, the ROS topics to be recorded result from being able to produce the following plots:

- **Traveled Distance vs Time:** This plot allows to assess how efficiently the rover covers the exploration area when compared to other strategies.
- **Computational Cost vs Time:** This plot quantifies the processing load on the OBC, in terms of CPU usage, RAM, and disk, as well as network sent and received over time throughout the mission. It helps determine whether the computational requirements of object localization coupled with the state machines, remain within the processing capabilities of the Intel NUC.
- **Entropy Difference Per Centroid vs Centroid:** Entropy difference before and after the rover has visited the respective frontier centroid is used as a metric to evaluate the information obtained at each exploration step.
- **Explored Volume and Area Covered vs Time:** These plots are the most important one as it represents how the volume and area of the mapped

environment increase over time. A more efficiency strategy explores more of the environment in the same amount of time as others.

- **3D Generated Point Cloud After Exploration:** The reconstructed point cloud provides a visual and quantitative representation of the final map obtained through SLAM. It allows the verification of mapping completeness and accuracy relative to the known geometry of the testbed.
- **Odometry Trajectory on 3D Point Cloud:** Overlaying the odometry trajectory on the generated 3D map allows the comparison between estimated and reconstructed paths. This visualization helps to identify drift or inconsistencies between trajectories.
- **Velocity, Acceleration, and Respective Covariance vs Time:** This plot provides a dynamic characterization of the rover's motion.
- **3D Coordinates of Objects in the 3D Point Cloud:** This plot visualizes the estimated 3D positions of the detected objects within the reconstructed point cloud. It serves to validate the accuracy of the object localization pipeline by comparing the predicted coordinates with the actual positions of the objects in the environment.

Lunar Rover Mini ROS Topics ROS Bags Recorder

Accordingly, the ROS topics that provide this information, in the LRM, are the following:

- `/system_stats`
- `/grid_map`
- `/octomap_full`
- `/odom`
- `/candidate_frontiers`
- `/segmented_frontiers`
- `/segmented_frontiers_centroids`
- `/centroids_sphere_view`
- `/utility_function`
- `/normalized_entropy`
- `/cloud_map`
- `/map_stats`
- `/detected_objects`

A rationale for each topic, along with a description of its usage, is provided in Appendix B. To record the selected ROS topics, a Python node was implemented within the LN Manager of the LRM, specifically responsible for this.

8.2. Experimental Procedure

After the experimental setup and experiment assumptions were made, attention is directed toward the experimental procedure. This involves not only defining a SOP that ensures consistent and reproducible execution of each experiment, thereby improving efficiency, but also enumerating the specific experiments to be conducted.

8.2.1. Standard Operating Procedure

The SOP defines a structured sequence of steps to ensure that each experiment is conducted consistently, safely, and reproducibly. Of course, individual experiments might deviate slightly from the SOP given their specificity. The importance of the SOP goes behind this thesis as it can be reused for other students working on the rover. The SOP goes as follow:

1. **Start the Normal Operations node in the LN Manager:** This initializes the Simulink interface, establishes communication with the rover, and launches the high-level GUI. Refer to Section 2.2 for details on why this is considered normal operations.

2. **Start the Autonomous Navigation node in the LN Manager:** This launches the ROS nodes for the Intel RealSense camera, VO, SLAM, local mapping, and navigation stack, as well as the OctoMap server and RViz visualization tool.

3. **Start the Autonomous Exploration node in the LN Manager:** This launches RAFCON to execute the exploration state machines and the OctoMap message saver for saving OctoMap 3D maps.

4. **Start the `LRM_object_detection` node in the LN Manager:** This starts the object detection loop for real-time object localization.

5. **Start the Logging node in the LN Manager:** This activates multiple monitoring and logging functions: `LRM_system_monitoring` publishes CPU, RAM, and disk usage; `LRM_rosbag_record` records the selected ROS topics; and `LRM_area_covered` continuously computes the area explored by the rover from the 2D occupancy grid.

6. **Switch to Autonomous Navigation mode in the high-level GUI or RAFCON:** This enables the rover to execute commands generated by the autonomous navigation stack. If this mode is not activated, the rover will not act upon the commanded linear and angular velocities.

7. **Start the Exploration Mission hierarchical state machine:** This starts the execution of the exploration mission on RAFCON. The state machine to be started depends on the experiment that is being conducted.

The LN Manager with all the aforementioned nodes of the LRM is displayed in Figure 8.2.

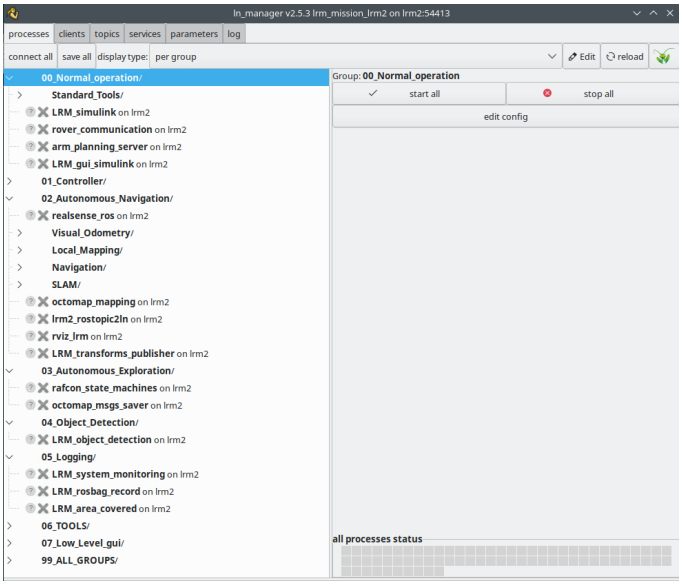


Figure 8.2.: LRM's LN Manager showing the main nodes along with their corresponding sub-nodes.

The post processing of each experiment is not described in detail here. In general, it consists in converting the data recorded in the ROS topics within the ROS bags into a format compatible with MATLAB to generate data plots.

8.2.2. First Experiment

The first experiment has two main objectives. The first is to evaluate the maturity of the autonomous navigation capabilities of the rover in complex scenarios. This is motivated by the fact that it was only demonstrated in one mission scenario going from point A to point B in the work of Ricardez Ortigosa [33]. The second is to compare the performance of the implemented exploration algorithm against open-source, state-of-the-art alternatives in a controlled environment to demonstrate the scientific relevance of this work.

To achieve this, the rover will be placed in 7 different initial poses within the PEL. At each pose, it will perform an initial rotation over an angle randomly selected in the range $[0, 2\pi]$, defining seven distinct mission scenarios. Using the low-level implementation of the hierarchical state machines, a set of frontier centroids is

computed for each scenario. Because this low-level implementation is common to all mission, independent of the exploration algorithm being used, this ensures that, although different exploration algorithms are applied to select the next target centroid, the set of candidate centroids remains identical across algorithms.

For each mission scenario, the exploration strategy under test is applied: the rover navigates to the selected centroid while mapping the environment and recording all relevant data in ROS bags. After reaching the target, the rover returns to its starting position. The experiment is then repeated with a different exploration algorithm, using the same initial conditions and candidate centroids. Because the selection of centroids depends on the exploration strategy, the rover is expected to choose somewhat different centroids for each algorithm. This approach enables a fair comparison between strategies, as differences in behavior can be directly attributed to the exploration algorithms rather than the environment or initial conditions.

The rationale for using seven distinct mission scenarios, rather than allowing the rover to map the entire PEL under each exploration algorithm, is twofold. First, terrain complexity may break the current capabilities of the autonomous navigation stack, making mapping the entire PEL potentially infeasible. By using smaller, controlled scenarios, the performance of the navigation stack can be evaluated in a more controlled environment.

Second, if the rover were to map the entire PEL under different algorithms, it would rarely encounter the same frontier centroids in the same context, preventing a direct comparison of decision-making between strategies. Also, given a camera maximum range of 2 m and the PEL dimension of 5x10 m, it is expected that mapping the entire PEL will not take more than 10 centroids to explore.

Thus, this approach preserves the ability to assess overall exploration efficiency: for each scenario, explored volume and time to reach the target centroid are recorded, which can then be compounded across all scenarios for the same exploration algorithm, effectively treating each mission scenario as an isolated instance within a continuous exploration mission. The exploration algorithms that will be compared are: *Closest Frontier*, which selects the closest frontier in terms of Euclidean distance for the rover to move to, *Random Frontier*, chooses a frontier randomly from the list of available frontiers, serving as a baseline for comparison, *Entropy-Based*, selects the frontier that maximizes expected information gain, *Utility Function*, chooses the frontier that maximizes the ratio of information gain and travel cost in terms of total travel time, and *Utility Function with Frontier Coverage*, which is similar to the utility function, but includes an additional term that defines the azimuth range that is worth observing at the frontier centroid.

Mission Scenario Starting Conditions

It was mentioned that after reaching the target, the LRM returns to its starting position so that the same initial conditions apply when running the same mission scenario with a different exploration algorithm. However, it is important to ensure that the initial conditions are in fact the same. When the Autonomous Navigation node is started, the odometry and map frames are initially aligned, so the rover's starting position coincides in both frames. Over time, odometry naturally drifts,

causing the odom frame to diverge from the map frame. The map frame, maintained by SLAM, remains anchored to the global map, providing a corrected estimate of the robot's position, while the odometry frame continues to drift.

To quantify this divergence, the Euclidean distance between the rover's positions in the odometry and map frames is computed. If this divergence exceeds a threshold of 5 cm when the rover returns to its original position, the experiment is repeated. Ensuring that the initial conditions are consistent in this way also prevents the coordinates of the frontier centroids to differ between exploration algorithms.

8.2.3. Seconds Experiment

The second experiment is designed to validate the 3D object localization pipeline. In this experiment, the object localization loop will be running, and the rover will be manually moved through the PEL using a gamepad for a fixed duration. The colored cubes are randomly placed across the PEL, Figure 8.1. For that reason, during this test, autonomous navigation is not employed. Nevertheless, VO and SLAM continue running, as it is still necessary to maintain and update the point cloud generated map and know the transformation between frames of the rover.

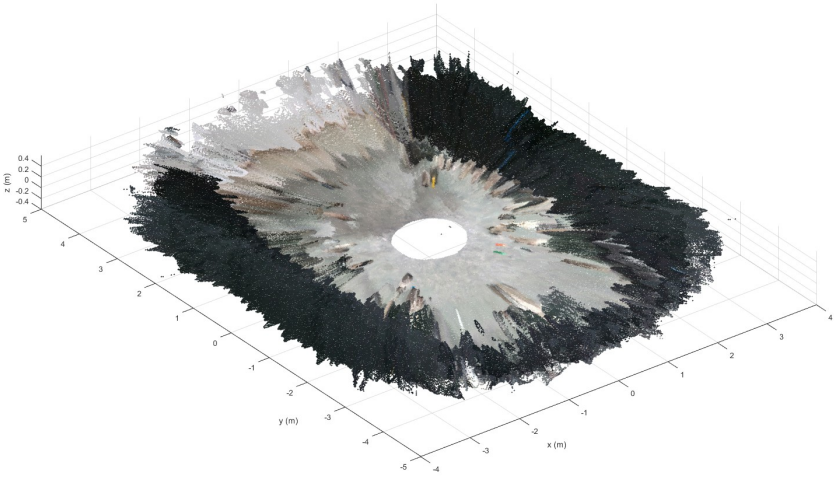
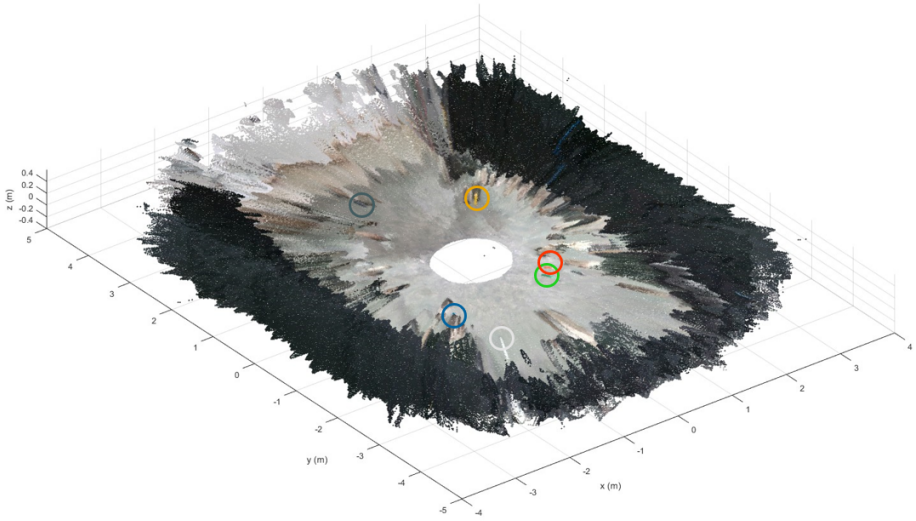


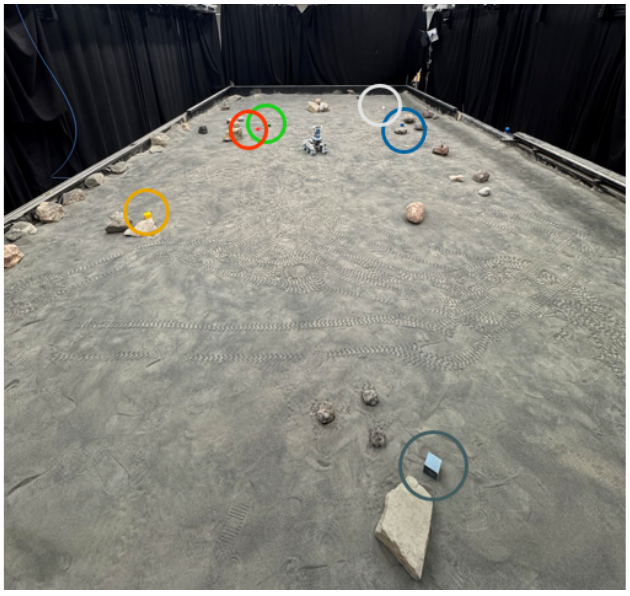
Figure 8.3.: Reference 3D map of PEL after 360° rotation.

The primary goal is to assess whether the rover can successfully detect all or some of the placed OoI and if their 3D coordinates are accurate. This is particularly important because object detection inference relies on a confidence threshold, which determines when an object is considered detected. Objects located further away have lower confidence scores, reducing the probability of detection. Of course, the random walk with the gamepad is biased towards object detection because this

is what this experiment is suppose to evaluate. Nevertheless, this is not the case for experiment 3. A confidence threshold of 0.85 was chosen for this experiment.



(a) Reference 3D point cloud map of PEL after 360° rotation with cube markers.



(b) PEL configuration experimental setup with cube markers.

Figure 8.4.: PEL experimental setup and corresponding 3D point cloud map with cube markers taken from the control room side.

Knowing if the object is detected far away or close to the rover also opens up future work research. For example, a strategy could initially use a low confidence threshold to detect a potential object, then guide the rover to approach the object while continuously monitoring the confidence score until it surpasses a higher threshold, *e.g.*, 0.9, ensuring that the object has been correctly identified. This would shift the mission scenario from exploring an unknown environment while detecting OoI in parallel, to actively guiding exploration with the goal of finding a specific OoI.

Accuracy of Estimated 3D Object Coordinates

The accuracy of the estimated 3D coordinates is evaluated as follows. A reference map of the PEL was created by placing the stereo vision camera at maximum range and having the rover perform a full 360° rotation at the center of the testbed. This generates a highly accurate 3D map of the environment, Figure 8.3. Given that the cubes are only 5x5x5 cm and the dimensions of the PEL are 5x10 m, they are difficult to visualize in the given plot. Theferore, Figure 8.4 depicts the PEL experimental setup and corresponding 3D point cloud map with cube markers for easier visualization. In Figure 8.4a the borders ar the PEL are well defined by the curtains in three of the four sides, while the remaining side is bounded for the control room wall and windows.

The 3D coordinates of the objects detected during the experiment are then overlaid onto this reference map to determine whether they fall within a reasonable radius of the actual position. A radius of 25 cm was chosen as an acceptable threshold. This threshold is the same for distinguishing two different objects apart, since afterwards the rover can rely on its local perception rather than global odometry when grasping the object in the future, as described in Subsection 8.49.

In this experiment, the system will also be monitored to provide real time information about the computational load on the Intel NUC, mainly how is the behavior of the frequency of the inference loop. In addition, the PEL is equipped with an ART (Advanced Realtime Tracking) system, which is an optical motion capture setup providing precise 3D position and orientation measurements of objects in real time. The ART system was not used in this study because the PEL was undergoing hardware and software reconfigurations, with the ART system unavailable when experiments were being conducted.

8.2.4. Third Experiment

The third and final experiment is intended validate both the developed exploration strategy and the 3D object localization pipeline in an integrated scenario. For this purpose, the rover will be placed at a random location within the PEL and will autonomously explore the environment. The exploration process will continue until no additional frontier centroids are available, indicating that the environment has been fully mapped. While exploration is happening, the object detection loop is running in the background. The 3D coordinates of the detected objects can be seen in RViz while the rover is exploring, and the ROS topic responsible for publishing the detected objects with a given timestamp can be analyzed later.

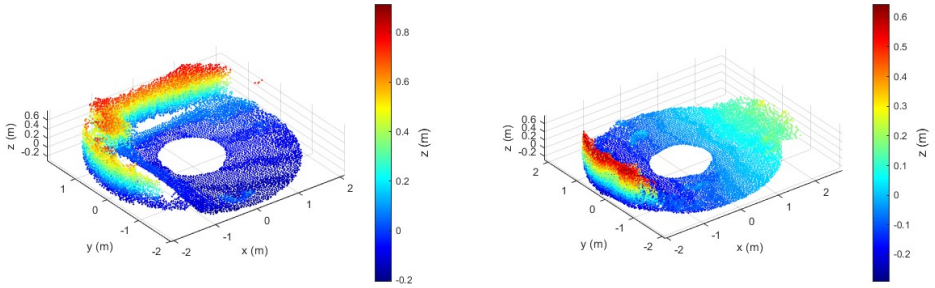
With the three experiments defined and characterized, the next step involves analyzing their results.

8.3. Results and Analysis

This section presents the outcomes of the experiments described in the previous section and provides an in-depth analysis of the obtained results. The analysis mainly focus on quantitative data from the benchmarks to evaluate performance of Subsection 8.1.4. After evaluating each experiment, a final discussion is also provided which summarizes the key findings.

8.3.1. First Experiment

Two of the 7 mission scenarios 3D point clouds are presented in Figure 8.5. All the mission scenarios are presented in Appendix C. Mission scenarios 1, 2, 3, and 6 both are a 360° rotation at different location in PEL. Mission scenario 3 is a 360° rotation right in the middle of the PEL. This is inferred given that there are no "walls" visible in the 3D point cloud of this mission scenarios. The "walls" are noticeable in other mission scenarios though, which are the black curtains surrounding the PEL.



(a) Mission Scenario 1.

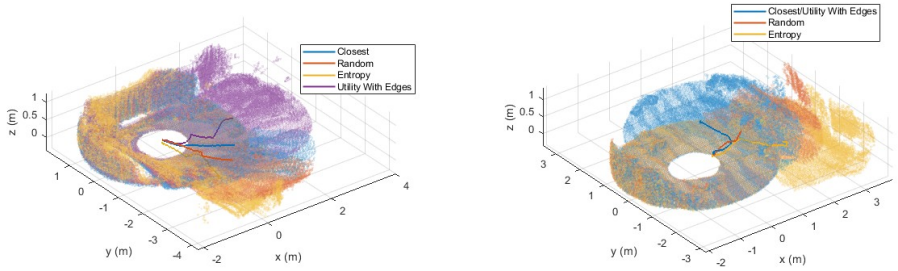
(b) Mission Scenario 7.

Figure 8.5.: Example of two mission scenarios for the first experiment.

Mission scenario 3 and mission scenario 5 have the rover pplaces in a slope, which can be observed by the height gradient in the plots. For mission scenario 5 and mission scenario 6, the rover only performed a 90° and 180° rotation, respectively. For mission scenario 7, after performing the 360° rotation, the rover moved forward at its maximum linear speed for 14 seconds.

First, the results are presented by overlaying the 3D point clouds generated with the corresponding rover odometry positions. If all exploration algorithms choose different centroids for the same mission scenario, the total number of different point clouds generated and respective odometry positions is 4. This is because when frontier coverage is added to the utility function, this does not influence the chosen

centroid. For instance, Figure 8.6 depict the overlaying 3D point cloud generated with the rover odometry positions for mission scenario 1, and mission scenario 7. All the overlaying 3D point cloud and odometry positions for each mission scenario are presented in Appendix D.



(a) Mission Scenario 1.

(b) Mission Scenario 7.

Figure 8.6.: 3D point clouds generated for each exploration algorithm with the corresponding rover odometry positions for the same mission scenario.

Moreover, it could be that different exploration algorithms choose the same frontier centroid as their next target. In such cases, the legend of each plot reflects this overlap. Also, regarding the utility-based strategy without edges, *i.e.*, without frontier coverage, and with edges, *i.e.*, with frontier coverage, both exploration algorithms choose the same centroid. As a result, the corresponding 3D point cloud is created in the same experimental run. Therefore, it is displayed only for the utility with edges. The same applies if any of the other exploration algorithms selects the same centroid as the utility-based strategy.

From the analysis of the figures, it is possible to conclude that visually, overall, the 3D point clouds generated by the utility-based strategy has more points that the others. As well, only mission scenario 1 and mission scenario 2 had four different frontier centroids being chosen. In contrast, mission scenario 4 had the same frontier centroid for the closest, entropy, and utility exploration algorithms. This is justified by the fact that the starting conditions of this mission scenario only had the rover perform a 90° rotation. With this, the starting 2D occupancy grid with segmented frontiers is the one shown in Figure 8.7, with the rover being located roughly at the intersection point between the yellow and the green segmented frontiers.

Even though frontier centroids are not directly displayed, it is clear that because the yellow or green centroids are much closer to the rover than the blue and red ones, the utility-based strategy computes that a much shorter travel time is required to reach one of the former centroids, biasing this strategy to the same centroid as the closest one.

However, these early conclusions are more qualitative than quantitative, and more

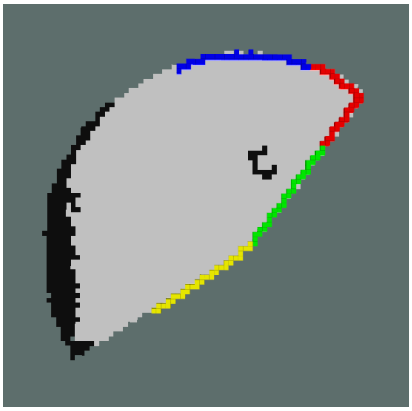


Figure 8.7.: 2D occupancy grid of mission scenario 4.

data shall be provided to confirm this. First of all, the computation time of each exploration algorithm is presented in Figure 8.8, for each mission scenario.

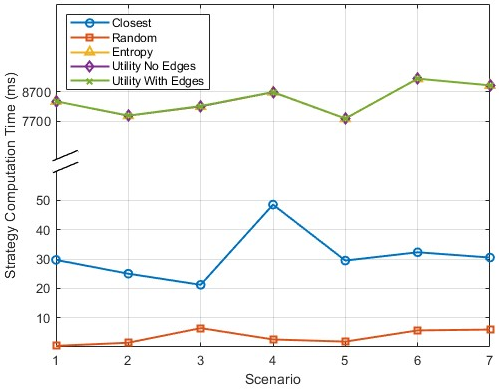


Figure 8.8.: Strategy computation time for each mission scenario.

Even though seven scenarios may be too few to draw definitive conclusions about small differences in computation time between strategies such as *Entropy*, *Utility No Edges*, and *Utility With Edges*, it can be inferred that these strategies take significantly longer than the *Closest* exploration strategy. This increased time is due to the ray casting computations, rather than the calculation of the utility function or frontier coverage. This is supported by the fact that the computation times of *Entropy*, *Utility No Edges*, and *Utility With Edges* are very similar, making them indistinguishable at this scale.

When it comes to real travel time, is is expected that the *Closest* strategy takes the smallest value at every mission scenario. The real travel time for each mission

scenario, Figure 8.10, shows that is, in fact, the trend, but with scenarios where the *Closest* strategy ends up taking longer than others.

When it comes to real travel time, the *Closest* strategy is expected to have the smallest values in every mission scenario. Figure 8.10 confirms this trend, although there are scenarios in which this one takes longer than some of the other strategies. Note that the rotation time for frontier coverage is also taken into account here. That is why the real travel times for the *Utility No Edges* strategy is always higher than the one for the *Utility With Edges* strategy.

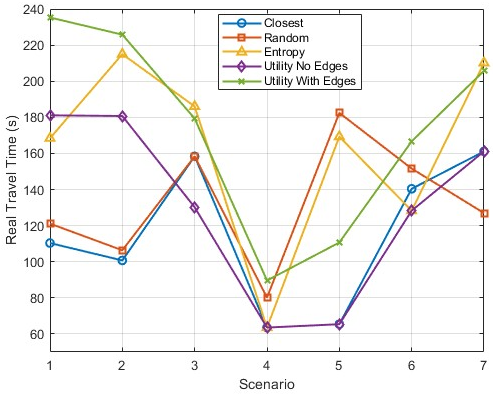


Figure 8.9.: Strategy real travel time for each mission scenario.

This difference is not because the algorithm to compute the expected real travel time is wrong, but because it only consideration orientation and Euclidean distance between the rover and the centroid, not accounting for how the cost map looks. For instance, if an obstacle is present, the rover may need additional time to get to the centroid. This difference is not because the algorithm used to compute the expected real travel time is incorrect. Rather, it arises because the algorithm only considers the rover's orientation and the Euclidean distance to the centroid, without accounting for the structure of the cost map. For example, if an obstacle is present, the rover may require additional time to reach the centroid.

Matter of fact, this occurs in scenario 6. As shown in Figure D.6, the odometry path of the *Closest* strategy follows a curved trajectory to reach the centroid, rather than a straight line, which increases the travel time. A similar situation occurs in scenario 3, Figure D.3, where the rover again takes a curved path to the centroid. In this scenario, the last *Closest* odometry message is corrupted, resulting in a straight line from the centroid to the origin, as if the rover returned home, which is not the case.

This opens the discussion on how the real travel time, overall, compares to the computed expected travel time. For this, Figure 8.10 showcases the difference between the real and expected travel time.

Based on the figure, it is evident that there is a substantial difference between the real and computed travel times, averaging around 55–60% and reaching a

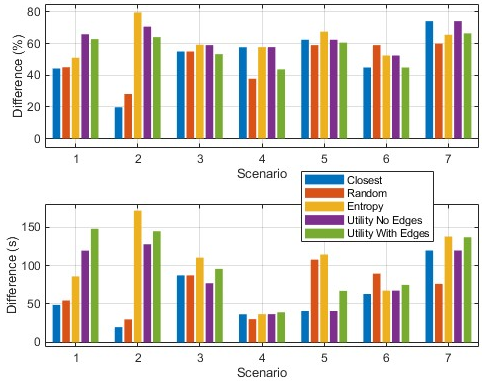


Figure 8.10.: Difference between real and expected travel time for each scenario.

maximum of 80%. In absolute terms, this difference reaches a maximum of 170 seconds. The closest prediction underestimated the travel time by approximately 20%, or 20 seconds, meaning that in all experimental runs, the rover took at least 20 seconds longer than the computed time to reach the centroid. As expected, the real travel time is always greater than the computed travel time.

With that in mind, the real travel time to centroid was plotted against the estimated time to centroid, 8.11. In this plot, the identity line indicates the overall relationship between the computed and actual travel times, showing how closely the estimates predict the real travel times and highlighting any systematic deviations. In other words, if the RMSE of the linear fit is 0 s, it means that the estimated travel times perfectly match the real travel times for all scenarios, with no deviation between the predicted and actual values.

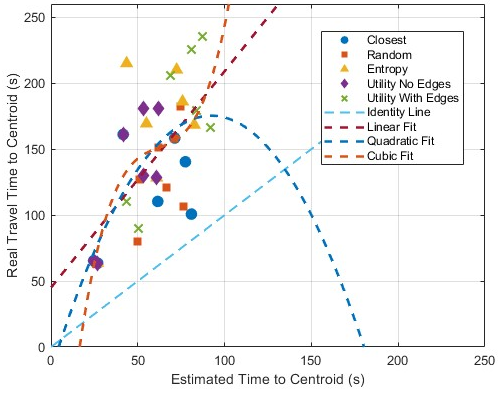


Figure 8.11.: Real travel time to centroid against estimated time to centroid.

According to the values on the plot, the RMSE for the identity line is 92.0 seconds. This is a very high RMSE value. Considering this, it was decided to create a way that would better adjust and correct the difference between both times. For this, linear, quadratic, and cubic fits were created, also depicted in Figure 8.11.

According to the plot, the RMSE for the identity line is 92.0 seconds, which is relatively high. Considering this, to better account for and correct the difference between the estimated and real travel times, linear, quadratic, and cubic fits were computed, as also shown in Figure 8.11. The respective RMSE values for the linear, quadratic, and cubic fits were 37.74 s, 36.81 s, and 35.81 s. From these RMSE values, it can be deduced that there is a significant improvement from the identity line to the linear fit, while the difference between the linear, quadratic, and cubic fits is less pronounced. Higher degree fits were avoided because they will overfit this particular data. Nevertheless, using the cubic fit to correct the expected travel times yields the results shown in Figure 8.12.

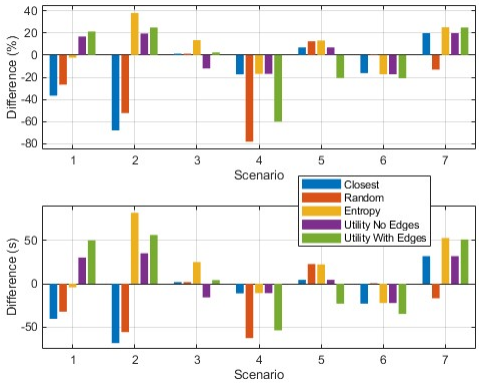


Figure 8.12.: Difference between real and expected travel time for each scenario with cubic fit correction.

Comparing Figure 8.12 with Figure 8.10, the cubic fit clearly improves the results. Nonetheless, some outliers remain, highlighting the need for a more accurate method to estimate travel times.

The same analysis in terms of comparing the real travel distance to centroid to the expected travel distance to centroid can be made. For this, Figure 8.13 illustrates the real travel distance to centroid, while Figure 8.14 highlights same difference between real and expected distance covered.

The real traveled distance to the centroid was computed using the rover's odometry positions at each timestamp, based on the cumulative Euclidean distance between consecutive position:

$$D_{\text{traveled}} = \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \tag{8.1}$$

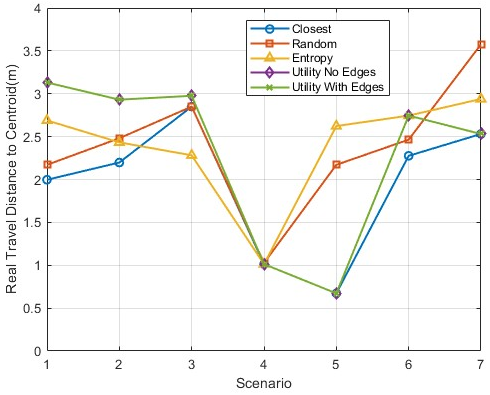


Figure 8.13.: Strategy real travel distance for each mission scenario.

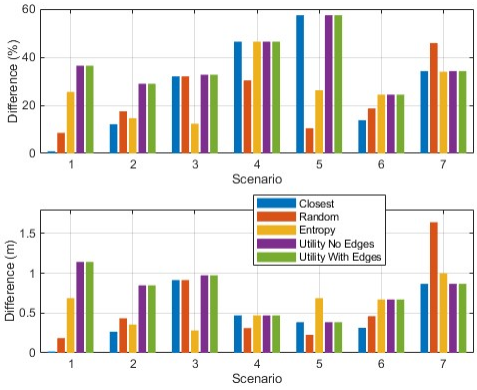


Figure 8.14.: Difference between real and expected travel distance for each scenario.

where (x_i, y_i, z_i) are the rover's coordinates at timestamp i, and N is the total number of odometry measurements along the trajectory. Here too there is a noticeable difference between both distances, with a RMSE of 0.72 meters. This is again justified by the fact that the cost map used by the autonomous navigation stack prevents the rover from moving along a straight line.

With the analysis of the difference between real and expected time and distance for all experimental runs, it is possible to move towards exploration efficiency results. The evidence for this can be seen with either entropy or voxel measurements. As previously explained in Section 6.3, entropy is a measurement of uncertainty about the state of the environment, where detected voxels are either free or occupied, which allow to define without doubt the state of the environment, related to information gain of a particular region of interest.

In other words, using Equation 6.2 their entropy is 0. Hence comparing these variations across different regions before and after the exploration, for a given exploration time, allows to conclude about the exploration efficiency of each strategy.

First and foremost, it is important to note that ray casting was performed across a sphere when using the entropy-based strategy. Nevertheless, since all data were recorded using ROS bags, the same ray casting, or counting the number of free and known voxels, can be applied to every centroid visited by any strategy, both before and after exploration. This allows the information gain associated with each centroid to be directly compared across all strategies.

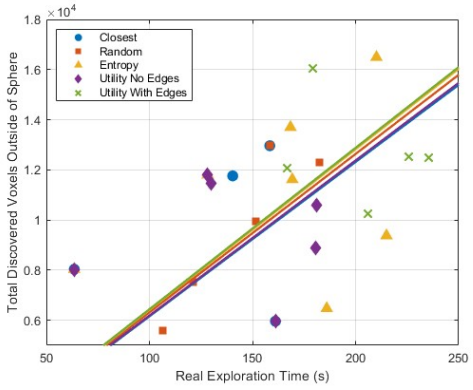


Figure 8.15.: Total discovered voxels outside of the spherical region of interest as a function of the real exploration time.

Strategy	Slope (voxels/s)
Closest	61.5677
Random	63.1643
Entropy	64.1149
Utility No Edges	61.8746
Utility With Edges	64.3889

Table 8.1.: Slope of discovered voxels per second for each exploration strategy outside of the spherical region of interest.

From this, it was decided to first examine if there is any correlation between the used strategy and how many voxels were discovered outside of the region of interest of each centroid, which is considered a sphere with the same radius of the ray casting sphere. Since both free and known voxels convey information about the environment, a higher number of discovered voxels indicates that a larger portion of the environment has been explored. It is expected that discovered voxels exist outside of the region of interest because when the rover moves to the frontier centroid, it might look into unknown regions of the environment, even unintentionally. Figure 8.15 provides this information, with the slope values being shown in Table 8.1. Note that the resolution of each voxel is 5x5x5 cm.

The slope values indicate that, for voxels discovered outside the sphere, there is no direct correlation between the exploration algorithm used and the volume of voxels discovered beyond the region of interest. This is expected since as the rover moves towards the centroid, its behavior is dictated by the autonomous navigation stack.

The next important finding concerns the total discovered volume as function of real exploration time inside the sphere of influence of each centroid. Note that real

exploration time not only entails real travel time but also the computation time for each exploration strategy. This is the most important data of this thesis work when it comes to proving that the implemented exploration strategy outperforms other state-of-the-art open source approaches. Figure 8.16 and Table 8.4 gives this data.

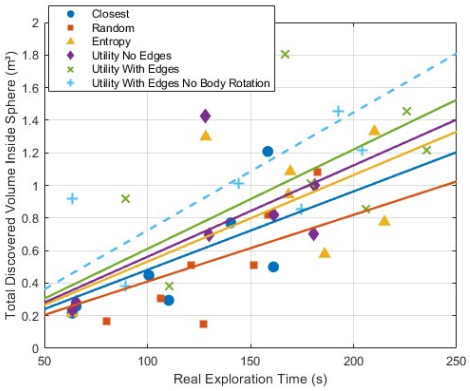


Figure 8.16.: Total discovered volume inside of the spherical region of interest as a function of the real exploration time.

Strategy	Slope (m^3/s)
Closest	0.0048
Random	0.0041
Entropy	0.0053
Utility No Edges	0.0056
Utility With Edges	0.0061
Utility With Edges No Body Rotation	0.0072

Table 8.2.: Slope of discovered volume per second for each exploration strategy inside of the spherical region of interest.

The slope values in the table confirm that this exploration strategy outperforms the others. It explores the environment more efficiently, covering a larger volume within the same amount of time. As important is the conclusion that the additional time required to perform ray casting, compared to just computing the closest frontier centroid, is outweighed by the significantly larger volume of the environment that this strategy enables the rover to discover.

The *Utility With Edges* performs the best, validating the trade-off study in Section 6.1 to choose the exploration strategy to be implemented. Moreover, even without the frontier coverage implementation, just using the *Utility No Edges* strategy to balance travel cost and information gain yields the second best results, with a purely *Entropy* strategy in third place.

Moreover, note that taking the *Closest* strategy as baseline, there is a maximum deviation of 27% compared to the *Utility With Edges* strategy for discovered voxels inside the spherical region of interest. This is much smaller for the discovered voxels outside the spherical region of interest, only 4.38%. This also validates this experiment, as the voxels outside the spherical region of interest are discovered as the rover moves and not by the choice of exploration strategy directly, which explain why their slope values are so similar.

An additional conclusion can be made when it comes to the correct application of the utility function. If only absolute discovered volume inside the sphere without accounting for the real exploration time was being considered, the *Entropy* strategy would always perform equal to or better than the *Utility No Edges* strategy.

Nonetheless, the utility function is employed correctly since the *Utility No Edges* strategy performs better than the *Entropy* strategy on overall exploration efficiency. a This also validates that the simple ratio between information gain and travel cost is enough to prioritize frontiers that offer the highest information gain relative to the effort required to reach them, when compared to more complex utility functions mentioned in Section 6.4. Still, it would be interesting to study if more complex utility functions would increase the exploration efficiency compared to this baseline.

The exploration efficiency can also be displayed as a function of total discovered voxels inside the sphere, similarly to Figure 8.15. This is portrayed in Figure 8.17.

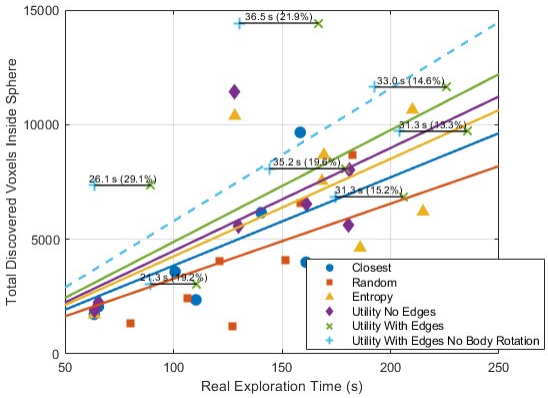


Figure 8.17.: Total discovered voxels inside of the spherical region of interest as a function of the real exploration time.

Not only this allows to reach the same conclusions when it comes to exploration efficiency, but also showcases an additional important consideration. As it was previously mentioned in Section 6.5, the pan-tilt of the LRM is fixed to avoid breaking the TF tree. Therefore, when performing frontier coverage, the rover has to rotation its body, which takes longer than simply rotation the pan-tilt unit.

Knowing the frontier coverage time from the recorded data, it is possible to deduce how much efficient this exploration strategy would be if the frontier coverage is made by rotating the pan-tilt, with no body rotation. An approximation was used assuming that a 180° rotation of the pan-tilt unit takes 6 seconds. The results of this optimization are shown in Figures 8.17 and 8.16, where both an absolute and a percentage decrease in time can be observed, ranging from 21.3 to 35.2 seconds and from 13.3% to 29.1%, respectively. With this approximation, a new improved value of 0.0072 m^3/s is reached in terms of exploration efficiency.

The same can be said when considering the area covered information given by the 2D occupancy grid, highlighted in Figure 8.18. Here, an interesting conclusion is that the *Closest* strategy performs better than the *Entropy* strategy.

Additionally, a final difference in terms of the entropy before and after exploration in the OctoMap can also be computed. A greater difference in entropy would mean that more of the state of the environment was known after exploration when

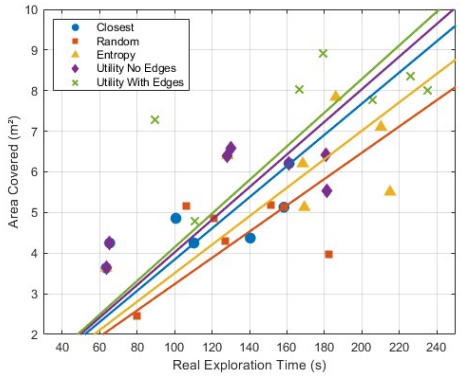


Figure 8.18.: Total area covered as function of the real exploration time.

Strategy	Area Covered (m^2/s)
Closest	0.0384
Random	0.0324
Entropy	0.0351
Utility No Edges	0.0402
Utility With Edges	0.0414

Table 8.3.: Slope of area covered per second.

compared to before. This is outlined in Figure 8.19. Once again, the *Utility With Edges* strategy performs the best, followed by the *Utility No Edges* and *Entropy*, respectively. Interestingly, the *Closest* strategy performs worst than the *Random* strategy. This could be because when moving to the closest frontier centroid, the rover covers less terrain, so less of the environment outside of the sphere of influence of the centroid is mapped.

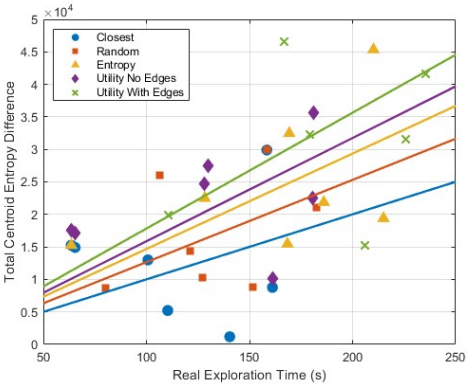


Figure 8.19.: Total centroid entropy different as function of the real exploration time.

Strategy	Slope (entropy bits/s)
Closest	100.0466
Random	126.5417
Entropy	146.7778
Utility No Edges	158.7715
Utility With Edges	178.2121

Table 8.4.: Slope of entropy bits difference per second.

One of the considerations in the first experiment was that each individual scenario could be treated as an isolated event within a single exploration mission, allowing the explored information to be compounded for each mission scenario. Hence, cumulative results for both volume and area, for each exploration strategy are

demonstrated in Figure 8.20 and Figure 8.21, respectively. Also, in each figure, the difference in volume and area between exploration strategies is also depicted.

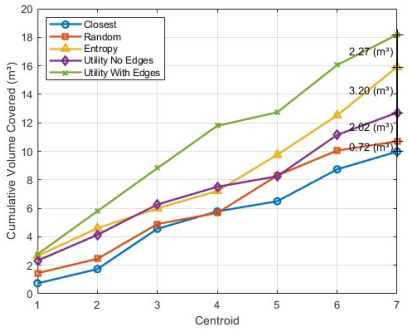


Figure 8.20.: Cumulative volume covered for a full mission scenario.

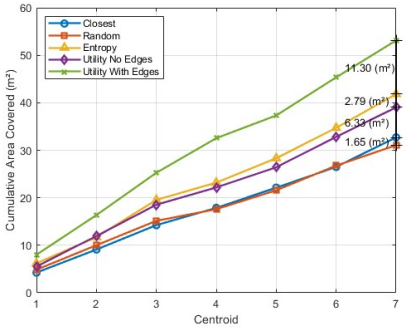


Figure 8.21.: Cumulative area covered for a full mission scenario.

Both the cumulative volume and area covered allow to conclude that the *Utility with Edges* strategy ends up gathering more information about the environment for a full mission scenario. In addition, the *Entropy* strategy outperforms the *Utility No Edges* strategy. This has to do with the fact that real exploration time is not being accounted for here.

Lastly, given that the rover was able to navigate complex environments in each of the seven mission scenarios, this experiments also allowed to mature the autonomous navigation stack of LRM. Not only that, but this experiment represented a total run time of approximately 18 hours by the LRM over a three-week period, the longest rover operations to date, with the wheels having to be calibrated 3 times during this period through the low-level GUI. The wheels were calibrated when a visual offset of approximately 4–5° was observed between the wheel's position and its actual zero position.

A final analysis can be made that concerns the range of frontier coverage. For this, the range of normalized entropy greater than 0.95 was computed for every possible frontier centroid, even the ones that did not had the highest utility. Figure 8.22 depicts this range for all frontier centroids, with the actual covered ones by the *Utility With Edges* strategy highlighted.

The average range is also presented, around the 0°, with a range covering 165.311 °. Is it interesting that this value is almost half the total available range. given that if a frontier centroid is place between the regions of known and unknown space. This somewhat aligns with was expected since, if the environment is not too complex, a frontier centroid lies between regions of known and unknown space, therefore, only around half of the surrounding area contributes efficiently to information gain, according to this strategy.

Finishing this subsection, the cumulative number of voxels for the *Utility With Edges* strategy can also be computed for the seven mission scenarios, Figure 8.23.

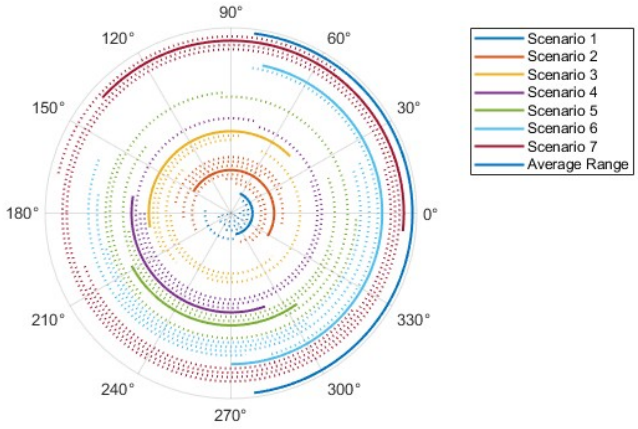


Figure 8.22.: Angle range of normalized entropy values above 0.95 to be covered.

Here, the last total number of voxels and timestamp from one previous mission scenario equals the starting values for the next.

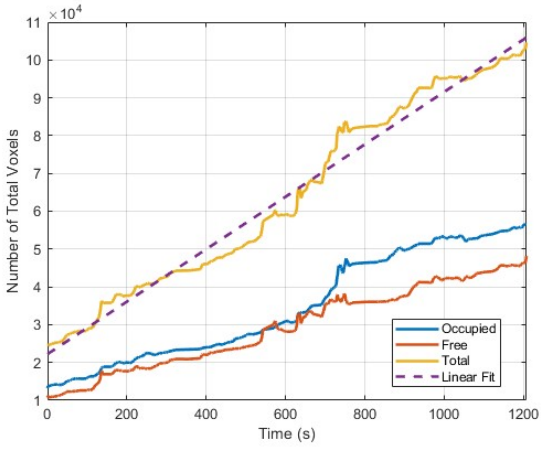


Figure 8.23.: Cumulative total number of voxels over time using the *Utility With Edges* strategy.

The linear fit slope for the total number of voxels yields 69.3341 voxels/s, which corresponds to 0.0087 m^3/s. This value will be compared with the one obtained during the full autonomous mission of mapping the PEL in the third experiment.

8.3.2. Second Experiment

The second experiment was designed to evaluate the accuracy of 3D object localization, as well as the computational load on the Intel NUC during the object localization loop, using the 3D point cloud map of the PEL shown in Figure 8.3 as a reference. Figure 8.24 illustrates the generated 3D point cloud while the rover is moved with the gamepad and the object localization loop is running. Additionally, the odometry positions of the rover are shown in purple, and the estimated coordinates of the cubes, obtained from the ROS topic publishing the 3D coordinates of each cube at a given timestamp, are also displayed. The position and orientation of the rover at the moment each detection occurred are indicated with arrows colored to match the corresponding detected cube. In a cleaner way, Figure 8.25 shows the same odometry position and estimated 3D coordinates of the cubes without the generated 3D point cloud.

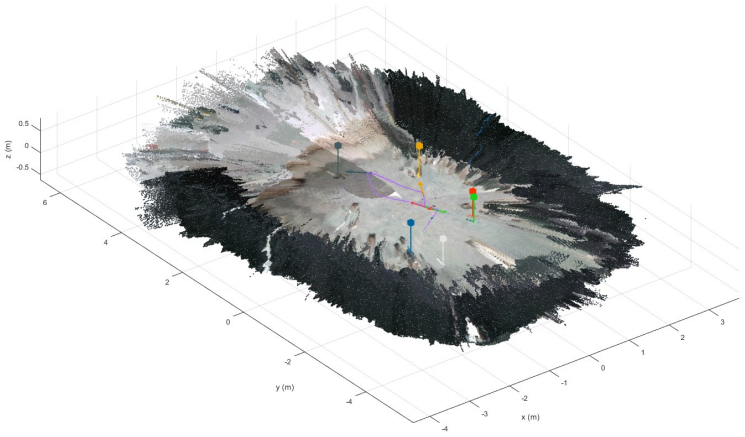


Figure 8.24.: Generated 3D point cloud data with object localization depicting the estimated 3D coordinates of OoI.

Visually, it can already be concluded that the generated 3D point cloud accurately portrays the PEL. Moreover, the estimated coordinates of the cubes are close to their positions in the generated 3D point cloud. However, there is a distinction between the 3D coordinates being accurate relative to the generated point cloud and being accurate relative to the reference 3D point cloud, which serves as a sort of ground truth.

Thus, by plotting the estimated 3D coordinates within the reference 3D point cloud and comparing them to the centers of the cubes in the reference, it is possible to visualize if the actual cube positions fall within the threshold region of 0.25 m radius around the estimated coordinates, Figure 8.26. For better visualization, the data is shown on the xy-plane.

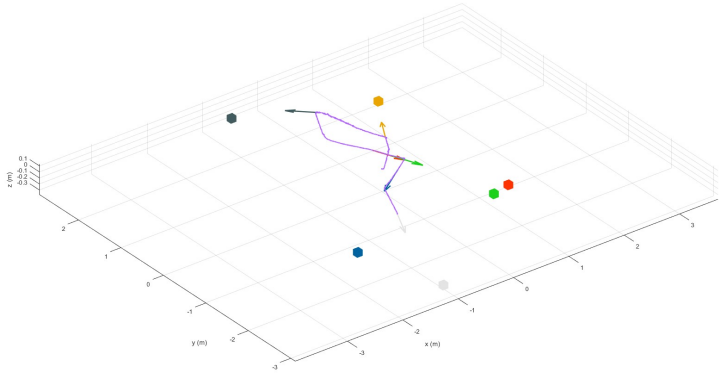


Figure 8.25.: Object localization with estimated 3D coordinates of OoI and odometry positions.

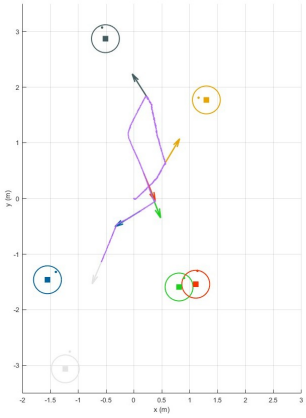


Figure 8.26.: Visualization of estimated cube positions. The threshold region of 0.25 m radius around each estimated coordinate is shown, with dots representing the actual ground truth of the rovers in the reference 3D point cloud.

From the analysis of the figure, it is possible to conclude that only the white cube falls outside the threshold region. This is reasonable, since, according to the odometry positions, it is the last cube to be discovered, meaning that greater odometry drift has accumulated up to this point. Also, in this case, there are no duplicate cubes, which validates even more the object localization loop. It is also possible to compute the distance between each cube center and corresponding

point, as well as the distance between the rover and the estimated cube position, when detection occurred. This is exhibited in Table 8.5 and Table 8.6, respectively.

Cube	Distance (m)
Blue	0.2052
Green	0.1836
Orange	0.2419
Volute	0.2088
White	0.3202
Yellow	0.1456

Table 8.5.: Distance between each cube center and estimated coordinates.

Cube	Distance (m)
Blue	2.0759
Green	1.8222
Orange	2.1919
Volute	1.3287
White	2.1326
Yellow	1.3826

Table 8.6.: Distance between rover position and estimated cube coordinates at the time of detection.

With this information the average distance between each cube center and estimated coordinates is 0.2175 m, while the average distance that the rover detects the object is 1.8223 m. This distance is determined either by how far the rover is when the confidence threshold is finally exceeded as it moves forward facing the object, or simply when the rover turns and the object comes into view.

When it comes to computation load on the Intel NUC, Figure 8.27 represents the data rate of both net sent and received during this experiment. The plot shows peaks and dips, but their frequency seems too random so that any relevant conclusions can be drawn.

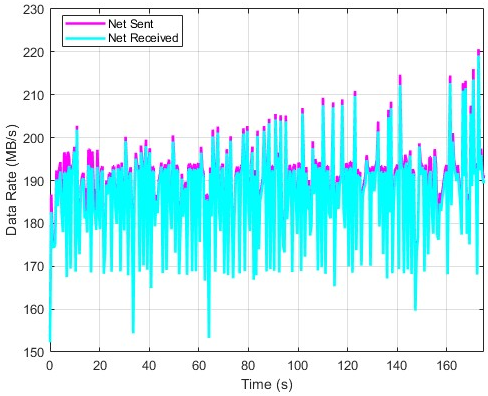


Figure 8.27.: Data rate for both net sent and net received.

Besides, the same cannot be said when is comes to CPU, RAM, and Disk usage, Figure 8.28. Here, disk usage remains constant throughout the experiment, while CPU and RAM exhibit a sinusoidal pattern. Moreover, while CPU usage oscillates around the same value, RAM usage keeps slightly increasing as exploration carries

on. Too, not that CPU usage spikes around 90%, an extremely high value. In addition, the periodic behavior of the CPU and RAM usage can be attributed to the object localization loop, where spikes represent inference being run of the OBC.

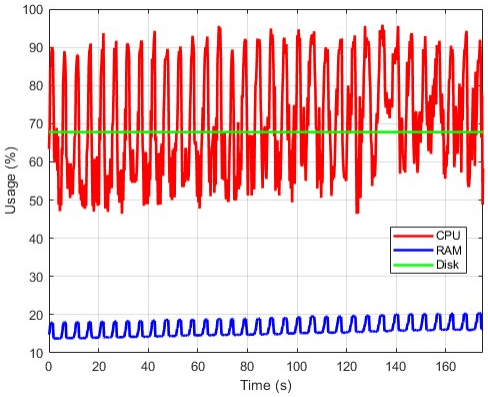


Figure 8.28.: CPU, RAM, and disk usage during experiment.

This hypothesis can be confirmed by fitting the CPU and RAM usage into a sinusoidal wave, and comparing their frequencies with the average frequency of the object localization loop, measured over 10 iterations in Subsection 7.2.3, which is 6.8 s or a frequency of 0.147 Hz. The CPU and RAM fitted sinusoids are shown in Figure 8.29 and Figure 8.30. The frequency of the CPU fitted sinusoid is 0.1879 Hz, while the RAM fitted sinusoid is 0.1822 Hz. Both are slightly higher than the threshold value of 0.147 Hz, which ensures real time object detection and localization is possible. Furthermore, both values are similar to the complete loop averaged inference time over 10 iterations of 5.81278 s or 0.172 Hz.

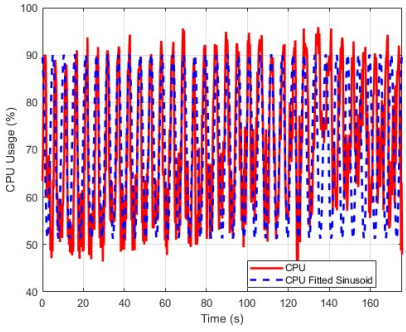


Figure 8.29.: Fitted sinusoidal wave for CPU usage.

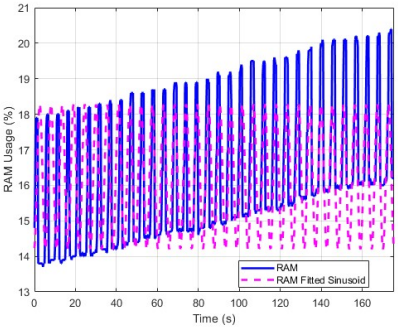


Figure 8.30.: Fitted sinusoidal wave for RAM usage.

8.3.3. Third Experiment

Finally, the third experiment, involving the full mission scenario with the *Utility With Edges* exploration strategy running in parallel with 3D localization of OoI, can be conducted. This allowed to validate the two different hierarchical state machines.

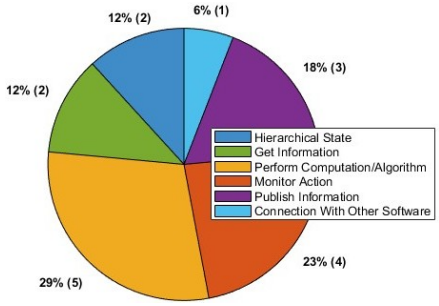


Figure 8.31.: State machine distribution for the object localization loop.

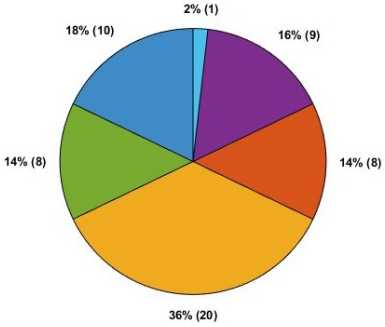


Figure 8.32.: State machine distribution for the full mission pipeline using the *Utility With Edges* strategy.

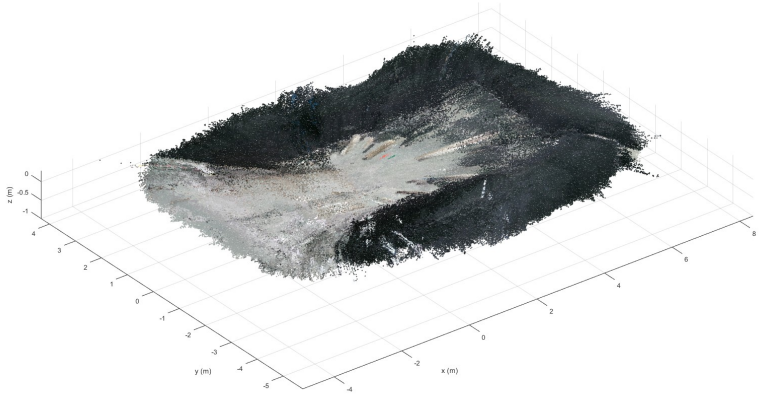
Each of the state machines belongs to one of these categories: *Hierarchical State*, *Get Information*, *Perform Computation/Algorithm*, *Monitor Action*, *Publish Information*, and *Connection With Other Software*. Thus, the number of different state machines used are presented for the object localization loop, Figure 8.31, and the full mission pipeline with the *Utility With Edges* strategy, Figure 8.32. Overall, 56 state machines were necessary for complete autonomous exploration, while 17 state machines where necessary for the 3D localization loop of OoI, for a total number of 73 state machines.

When trying to compute the 3D point cloud on MATLAB from the `.ply` generated file after exploration by RTAB-Map, a 22751398x6 matrix was created which stored the positions and RBG colors of each point of the point cloud. In comparison, the second experiment produced a 6497468x6 matrix.

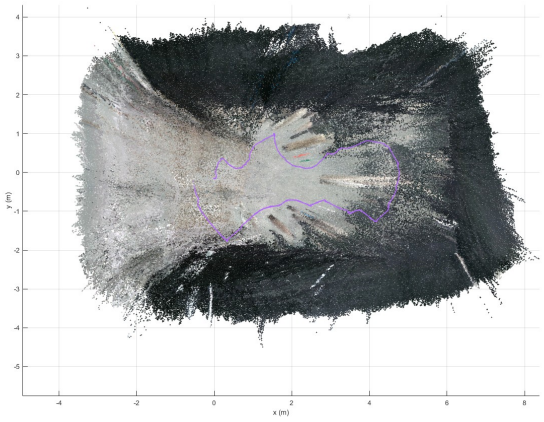
This means the third experiment 3D point cloud is 3.5 times denser than the one generated during the second experiment. For this reason, when trying to compute the 3D point cloud MATLAB crashes. Thus, the original point cloud of this experiment was reduced by 35% before plotting in MATLAB. The results for a 3D and 2D-plane view with overlapping odometry are shown in Figure 8.33a and Figure 8.33b, respectively. Reducing the density of the point cloud by 35% does not compromise its information.

In the 2D-plane plot the z coordinate of the odometry position was set high enough to tell it apart from some of the points in the point cloud. Several conclusion can be drawn from the analysis of both plots. First, it is clear from the odometry position that the autonomous navigation stack successfully avoids obstacles while moving towards a position goal. This can easily be inferred from the way the

odometry contours both rocks and cubes in its trajectory.



(a) 3D view of 3D point cloud.



(b) 2D-plane view of 3D point cloud with odometry position.

Figure 8.33.: Third experiment generated 3D point cloud.

Second, it can be seen that the accuracy of the PEL map is poorer than the one generated in the second experiment. This has to do with the rover that in the third experiment, the rover traveled a larger distance, which contributed to an increased drift in odometry, which compromised the accuracy of the point cloud. Even though the black curtains surrounding the PEL blend more with the testbed, the rectangular dimensions of the PEL are still noticeable.

Nevertheless, when comparing to the testbed setup of Figure 8.1, noticeable

features stand out. Particularly. the orange, green, and blue cubes, as well as the rocks. The white cube is blended with the points of the point cloud representing the black curtains.

Moreover, the yellow and volute cubes are not shown, as the part of the point cloud on this side of the PEL appears to be sloped upward. This also coincides with the last part of the PEL that was mapped by the LRM, which indicates an odometry drift along the z-axis. This is addressed further in the OctoMap representation of the PEL. It appears that the PEL extends to the left side. This is justified since this is the side of the control room are there are no black curtains here.

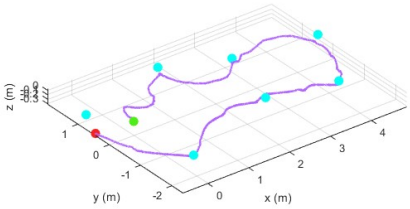


Figure 8.34.: 3D view of odometry during exploration.

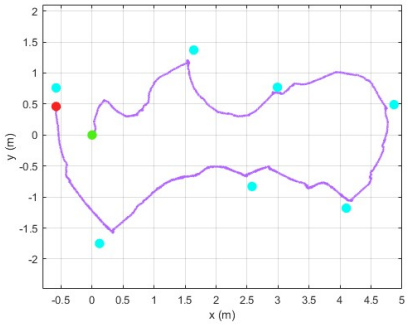


Figure 8.35.: xy-plane view of odometry during exploration.

The second analysis focuses on just the odometry data. Figure 8.35, Figure 8.34 and Figure 8.36, show the odometry trajectory of the LRM during the exploration mission, presented in a 3D view, in the xy-plane, and in the yz-plane, respectively. Also, blue dots in the figures represent the chosen frontier centroid, the green dot the starting position, and the red dot the final position.

It can be clearly observed when an obstacle was encountered by the LRM and the autonomous navigation reacted, as the odometry trajectory shows a curved path between points. This further validates the autonomous navigation capabilities of the LRM. Moreover, the total traveled distance by the rover according to odometry was 16.86 meters. On top of that, Figure 8.36 highlights the odometry drift along the z-axis, already noticed from the analysis of the point cloud. In fact, the rover estimates that is has moved -40 cm along the z-axis when, in fact, its trajectory should be approximately flat.

What is more, a total of 7 frontier centroids were explored to map the PEL. Some frontier centroid seem "above" the odometry position. This is because the frontier centroids are first selected as candidates from the 2D occupancy grid, where there are only x and y. This being the case, as odometry shifts from a from trajectory, the frontier centroids might appear "above" or "below" the rover's odometry. Also, it can be seen that in the rover's trajectory, the points where the rover reaches the centroid and computes the next one to go to do not coincide exactly with the centroid itself. This comes from the position threshold defined by the user.

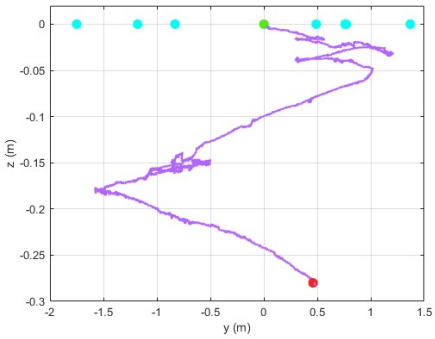


Figure 8.36.: yz-plane view of odometry during exploration.

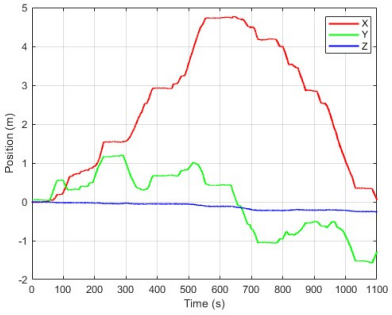


Figure 8.37.: Linear position over exploration time.

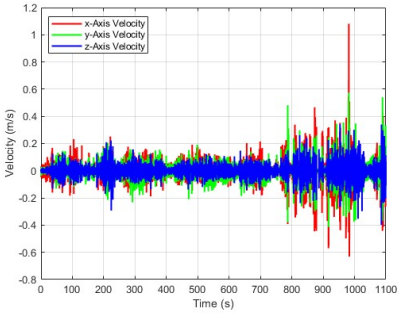


Figure 8.38.: Linear velocity over exploration time.

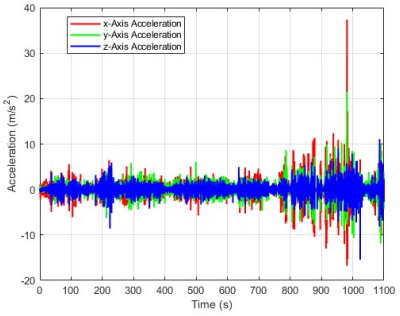


Figure 8.39.: Linear acceleration over exploration time.

Complementary to this, the position, velocity, and acceleration along the three axes in both linear and angular terms can also be presented, Figure 8.37, Figure 8.38, and Figure 8.39 for linear, respectively, and Figure 8.40, Figure 8.41, and

Figure 8.42 for angular, respectively. For both linear and angular motion, the position curves appear significantly smoother than the corresponding velocity and acceleration. This behavior arises because velocity and acceleration are obtained through differentiation of the position data, which amplifies measurement noise and small variations.

As a result, the derived signals become more irregular and spiked, sometimes appearing saturated at certain points. For instance, the angular acceleration in the roll axis reaches values exceeding 600 m/s^2, which is unrealistically high for the LRM. Such values likely result from a combination of sensor noise and limited sampling resolution.

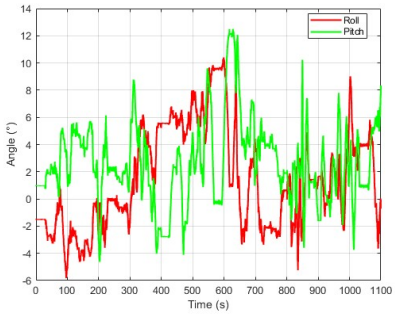


Figure 8.40.: Angular position over exploration time.

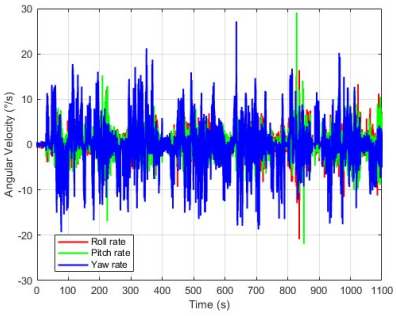


Figure 8.41.: Angular velocity over exploration time.

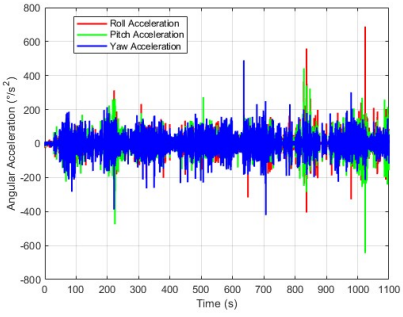


Figure 8.42.: Angular acceleration over exploration time.

Within the same odometry framework, the variance of the position (x, y, z) can also be computed and plotted. There are depicted in Figure 8.43. The variance of s a statistical measure of how much the position values deviate from their mean expressed as:

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{8.2}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ is the mean position, and $(x_i - \bar{x})^2$ represents the squared deviation from the mean.

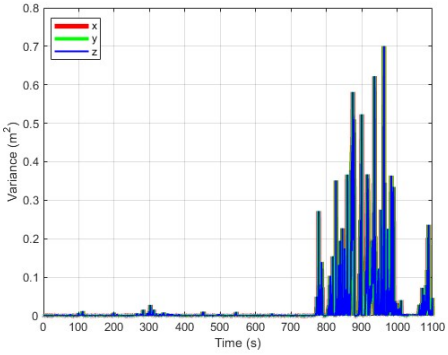


Figure 8.43.: Variance of position during exploration.

Regarding position variance, it is interesting that its values remain close to zero for the first 750 seconds of exploration. After this period, the variance peaks at approximately 0.7 m^2 and stays like this for around 250 seconds, until it reaches 1000 seconds. It then briefly returns to near zero before peaking again at around 0.23 m^2, and subsequently drops to zero once more. The return of the variance to near-zero values after significant fluctuations may correspond to a loop closure event, where the LRM revisits a previously mapped area and the odometry estimates are corrected, temporarily reducing positional uncertainty.

Too, the OctoMap will be presented as a direct ROS topic echo of the cloud map visualized on RViz. The OctoMap created of the PEL laboratory is presented in Figure 8.44. This representation of the environment using 5x5x5 voxels is fine enough that, from the image, it is possible to notice some of the traits of the terrain that make up the PEL.

For this reason, a depiction of the PEL setup, Figure 8.45, and the OctoMap, Figure 8.46, is provided, with magenta circles identifying the regions of the PEL setup that can be clearly recognized in the OctoMap.

By inspecting the images, it can be seen that the voxels in the OctoMap. Regions 1 to 5 are more noticeable as they represend piles of rocks in the PEL. Region 6 is more spread out, therefore, the OctoMap is also less pronounced around this area.

By inspecting the images, it can be observed that the voxels in the OctoMap correspond to distinct regions of the PEL. Regions 1 to 5 are more prominent, as they represent piles of rocks in the environment, while Region 6 is more spread out, resulting in a less pronounced OctoMap representation in that area. From all, region 4 is the less noticeable. This is justified by the fact that only one rock exists in this region. Besides, the black curtains are shown both to the left and the right of the OctoMap.

Special attention must be given to the two outlier regions, A and B, in Figure 8.46. These small clusters of voxels are placed much higher than the others, and seem to

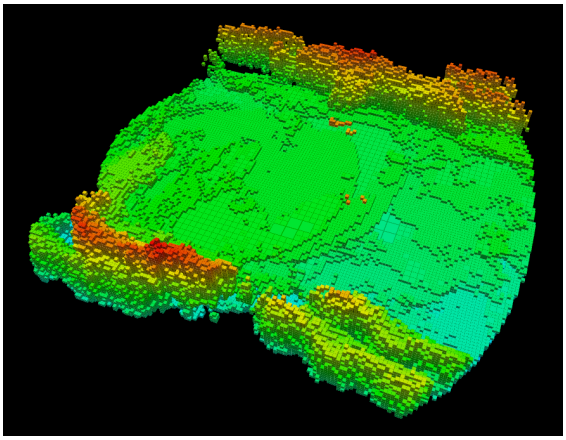


Figure 8.44.: OctoMap of PEL generated after exploration.

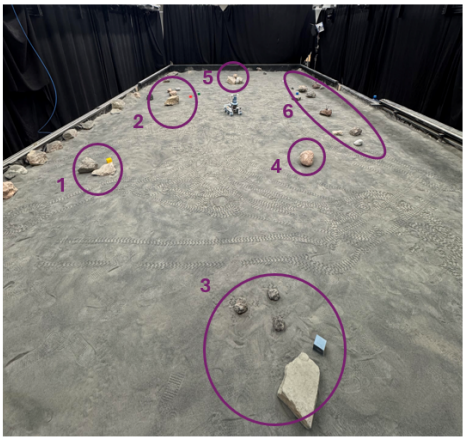


Figure 8.45.: Setup of the PEL laboratory. Magenta circles highlight distinctive regions of the environment.

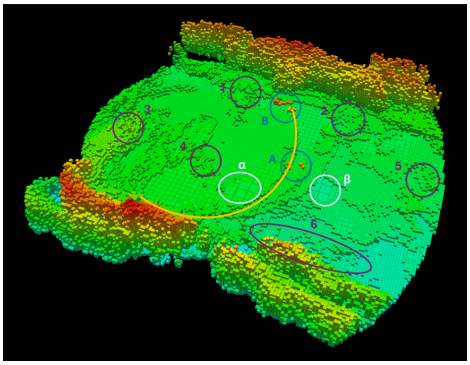


Figure 8.46.: Corresponding OctoMap representation of the PEL laboratory. Magenta circles highlight the same distinctive regions of the environment.

be some of sort of noise. This was caused by interference from the PEL roof lights, which affected the stereo vision camera. Even though this noise did not interfere with the PEL mapping, it could have affected the local cost map by introducing false obstacles. This being the case, this work would benefit from a more in-depth study on how light conditions affect the overall mapping capabilities of the LRM.

Another remark as to do with an odometry drift more evident along the z-axis. For one, the rover started by performing a 360° rotation, This first semi OctoMap before the rover moves to the first centroid is evident by the yellow curve. After this, the rover starts moving and the ground drifts to lower values, as noticeable by the color

gradient between Region 5 and Region 6. If there was no drift in odometry, this transition would be smoother as the PEL setup does not have this downhill slope.

Another remark concerns an odometry drift that is more evident along the z-axis. Initially, the rover performed a 360° rotation, which resulted in a partial OctoMap before it moved toward the first centroid, as indicated by the yellow curve in Figure 8.46. After this, as the rover continued to move, the ground level in the map appeared to drift downward, which can be observed from the color gradient between Region 5 and Region 6. If no odometry drift were present, this transition would appear smoother, since the PEL setup does not actually exhibit such a downhill slope.

On the other hand, when the rover starts moving in the opposite direction toward Region 3, the drift causes the ground level to appear shifted upward. This can also be explained if the stereo vision camera is tilted slightly upward or downward, and this misalignment is not reflected in the TF tree. However, both the stereo vision camera in the LRM and the camera frame in RViz were perfectly horizontal.

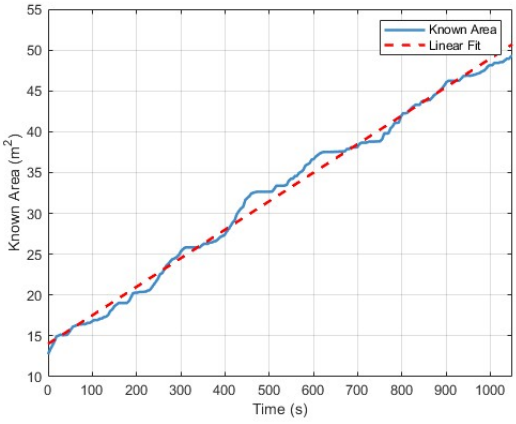


Figure 8.47.: Known area over time using the *Utility With Edges* strategy.

Lastly, regarding the OctoMap representation, Regions α and β also deserve attention. Even though the voxel resolution is 5×5×5 cm, the map is recursively subdivided into smaller voxels only where necessary on the octree structure, while uniform regions are represented by larger cubes. This behavior is also visible throughout the map, although less pronounced in certain areas.

Overall, the generated 2D occupancy grid and 3D point cloud visualized in RViz from this experiment are presented in Figure 2.9, and Figure 2.8, respectively, for a total exploration time in the ROS bag files of 18 minutes and 33 seconds. However, finishing the setup and starting the state machine took around 60 seconds, as it can be seen, from Figure 8.37, from the firs 60 seconds there is no rover movement. Taking around 10 seconds to compute centroid utility, taken from averaging the values of Figure 8.8 for this strategy, the total real exploration time is 17 minutes and 43 seconds. Using for $t = 0$ the actual instant when the state machine was started

yields the following known area as a function of time plot, Figure 8.47.

At the end of the exploration, the total known area was 49.23 m^2. This is slightly smaller than the PEL dimensions of 50 m^2, which is justified by a missing part of the PEL in the 2D occupancy grid. Also, the value of 49.23 m^2 is positively inflated since the black curtains, and the space between the PEL and the black curtains appears as occupied in the 2D occupancy grid. The slope of the linear fit of these points gives 0.0384 m^2/s. Using this value as reference, comparing to the slope of the same strategy from experiment 1, it has an error of 7.8%. This value allows to conclude that even though only 7 mission scenarios were tested in the first experimental, it is a good approximation in a full exploration mission in terms of exploration efficiency values.

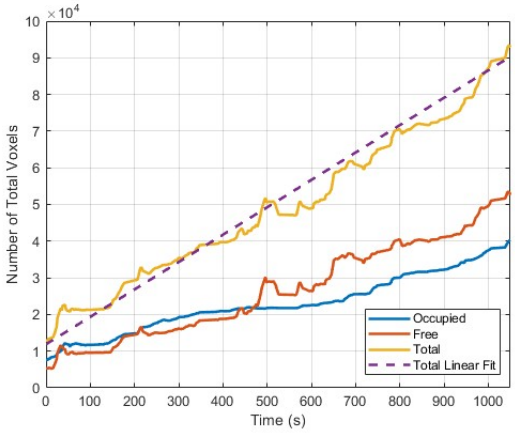


Figure 8.48.: Free, occupied, and total known voxels over time using the *Utility With Edges* strategy.

The same can be made in terms of the OctoMap, where Figure 8.48 shows the number of free, occupied, and total known voxels over time. The value at the end of exploration of total known voxels is 134181, which given the resolution of 5x5x5 cm, is a total volume of 16.7726 m^3. Also the linear fit slope for the total known voxels is 74.6798 voxels/s, which in m^3 is 0.0093 m^3/s. Using this as reference as comparing to the cumulative total known voxels for the same strategy in the first experiment, Figure 8.23, gives an error of 7.16 %. This error suggests that using the cumulative values of each mission scenario as a fully autonomous exploration is a good approximation, also validating the comparison in exploration efficiency between methods in the first experiment.

This slope value is greater which suggests that the rover ends up exploration more in the environment in a continuous mission, rather that in separate mission scenarios added together. Another interesting conclusion is the fact that there are dips in the plot, which indicate that over the course of exploration, the rover "loses" information about previously known voxels.

When it comes to the object 3D localization, Figure 8.49 allows to conclude that

the rover failed to localize the 3D volute cube at the end of the exploration, and given its odometry drift, the blue cube was considered two different cubes, 27 cm apart from each other.

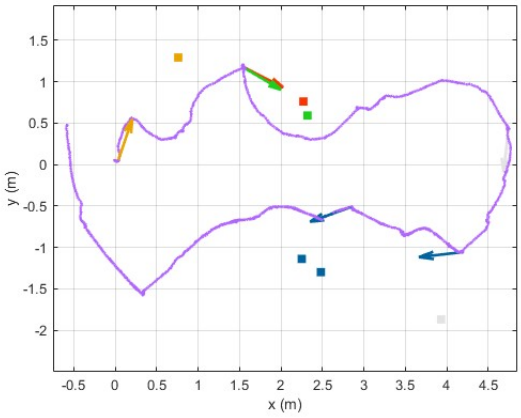


Figure 8.49.: Visualization of estimated cube positions.

Since the images in the objection detection are deleted post inference during exploration to save space, it is not possible to infer if the object detection did not occur due to an overload in CPU usage, or because at this point the rover estimated it was around 0.25 m into the ground, and the algorithm was not able to compute a proper transformation between the camera and the cube.

8.4. Discussion and Conclusion

The experimental validation performed in the PEL demonstrated the maturity of the LRM as an open-source, modular platform capable of autonomous exploration coupled with real-time object detection and 3D localization. Three main experiments were conducted. First, evaluate of the maturity of the autonomous navigation capabilities of the rover in complex scenarios and compare the performance of the implemented exploration algorithm against state-of-the-art approaches in the same domain. Second, validate the accuracy of the 3D object localization pipeline and its computation load on the OBC. Third, validate the fully integrated autonomous exploration mission comprised of the implemented exploration strategy and 3D object localization to map an unknown environment and detected objects of interest.

The first experiment compared the proposed *Utility*-based exploration strategy, both with and without frontier coverage, and *Entropy* maximization, against established methods, including the *Closest*-frontier strategy, and a *Random* baseline for control. Results indicated that the proposed method achieved a higher exploration efficiency, as measured by the slope of discovered volume per second, with values reaching 0.0061 m^3/s compared to the 0.0048 m^3/s for the baseline

Closest-frontier approach. In fact, all the exploration algorithms developed as part of this thesis work, which include the experimented *Entropy* and *Utility No Edges* strategies perform better than the baseline. Thus, these findings align with the hypothesis that using a simple ratio between expected information gain and travel cost yields a more efficient exploration behavior when choosing the next desired frontier centroid.

That said, analysis of the actual travel distances and times, when compared to the expected ones, revealed measurable differences. The RMSE between estimated and real travel times was approximately 92 seconds for the identity line, reduced to 35.8 seconds when a cubic correction fit was applied. Similarly, a 0.72 m RMSE was observed between expected and traveled distances. These deviations were primarily attributed to navigation constraints imposed by the local cost map, non-linear rover motion due to obstacle avoidance, and measured computation time between each transition of the full mission hierarchical state machine.

Despite these variations, the rover consistently reached all target centroids without manual intervention, validating the robustness of its navigation stack. Even if so the rover covered, for each centroid in every mission scenario, a distance ranging from 0.6 to 3.5 m, which is lower than the full mission scenario of the third experiment.

The second experiment, focusing on 3D object localization, confirmed the accuracy and reliability of the implemented pipeline. Using 3D-printed colored cubes as OoI, the rover accurately localized all expect one target within a 0.25 m radius threshold relative to their ground truth positions in the reference 3D point cloud. The white cube fell outside this margin, which was explained by accumulated odometry drift during after a greater exploration time. Nonetheless, the maximum distance between the reference and estimated cube position was 0.3202 m, while the minimum distance between the rover and the cube at the moment of detection was 1.3287 m.

Given that this minimum detection distance is approximately 4.15 times greater than the maximum positional error, it is safe to assume that, when the rover moves to the estimated position. it will still be able to detect the cube and rely on its local perception rather than global odometry when grasping the object in future work.

This result verifies that the perception pipeline, built on YOLO inference and depth-based information from camera intrinsic and extrinsic parameters, is capable of real time detection and 3D localization running only on the LRM CPU, achieving an average loop inference rate of 0.1879 Hz from the CPU fitted sinusoidal wave. The observed periodic oscillations in CPU and RAM usage confirmed synchronization with inference.

The third experiment, integrating exploration and object localization, validated the complete autonomous mission pipeline, including the hierarchical state machine framework employed in RAFCON. The rover successfully executed all planned centroids, avoiding obstacles autonomously. The final 3D OctoMap closely reflected the true structure of the PEL testbed, with a large number of regions from the PEL testbed present in the generated OctoMap. Some mapping noise, notably two voxel clusters above the ground, were identified as being caused by reflections from the PEL roof lights, suggesting that lighting conditions affect stereo vision

accuracy. Additionally, a *z*-axis drift was detected in the OctoMap, likely due to small odometry accumulation errors, rather than hardware misalignment, as both the stereo camera and respetive camera frame in RViz were confirmed to be aligned. Despite these limitations, the rover's total traveled distance of 16.86 meters matured rover operations.

This experiment also represented approximately 18 hours of cumulative operation, the longest runtime achieved by the LRM project to date, with the wheels having to be calibrated 3 times. This also validated other hardware choices in the LRM. Given the low-cost nature of the hardware, performing three calibrations over 18 hours of operation is considered good, and there was no overheating or component failure. The only downside was that, at full capacity, the battery lasted only 1.5 to 2 hours, significantly less than the initially expected 3-hour autonomy.

Additionally, it was clear that developing autonomous exploration strategies using state machines significantly simplifies the process. Initially, considerable time is required to design each state machine. However, over time, most of the building blocks, *i.e.*, state machines for creating hierarchical state machines to perform a specific task are already available. The main challenge then shifts from writing the code for each state machine to choosing its architecture. This not only applies to autonomous exploration strategies but also to other tasks performed by the rover. For instance, object grasping can reuse the object localization loop. For instance, object grasping can reuse the object localization loop, since both need to detect and locate the position of the OoI.

Overall, the validation experiments provide strong evidence that the proposed utility-based exploration algorithm, developed using a state machine framework, provides a scalable approach on low-cost robotic platforms, without relying on GPU acceleration for real time object localization or closed-source software. Nevertheless, several limitations emerged. Environmental lighting was shown to affect the generated OctoMap in a non-predictive way. The odometry drift mainly along the *z*-axis highlights the benefit of fusing VO with IMU data for better mapping accuracy.

9 Conclusions

This chapter concludes this thesis by addressing the research questions outlined in Section 9.1 and by presenting recommendations for future work in Section 9.2. This work set out to design and validate an open-source, modular framework for autonomous exploration and 3D object localization on low-cost robotic platforms such as the LRM. The goal was to integrate exploration and object localization within a state machine-based framework that is open, scalable, and comparable in performance to high-end systems, while performing better in terms of exploration efficiency when compared to existing ROS-based open-source strategies in the literature.

The LRM was further improved to autonomously map unknown environments, detect and localize OoI. This integration advanced its autonomy and established a foundation for a total autonomous mission pipeline, including object grasping. The proposed architecture, combining ROS and RAFCON, enabled managing the execution of autonomous tasks related to perception, exploration, and navigation.

In conclusion, the results validate the feasibility of an open-source, modular framework for autonomous exploration and perception on low-cost rover platforms. The developed pipeline successfully integrates high-level HFSMs, low-level state machines, and perception and exploration related tasks into an architecture capable of real time operation and exploration. These findings confirm the LRM's ability to perform complex autonomous missions, demonstrating the potential of low-cost, open-source systems for advanced robotic research and education.

9.1. Answering Research Questions

How can autonomous exploration strategies be implemented within an open-source, scalable and modular state machine-based framework for mobile robotic platforms?

- **What are the computational and architectural limitations that limit the implementation and ready-to-use deployment of exploration strategies on different mobile robotic platforms?**
  Implementing ready-to-use exploration strategies on different robotic platforms

is constrained by four principal factors. First, the onboard computational load, CPU vs. GPU, limits both the complexity of real time inference and the time available for the exploration algorithm to compute. In this case, performance is strongly bounded by the ray casting time. On one hand, robotic platforms with integrated GPUs benefit from a very high object localization loop frequency thanks to accelerated inference. On the other hand, CPU-only systems often require model quantization tailored to the specific CPU.
Second, middleware and communication impose latency challenges. Exchanges between ROS nodes nad HFSM states, such as control commands, and monitoring and listening to ROS topics, can delay transitions between state machines. Third, 3D mapping using a full OctoMap, where each voxel stores an occupancy probability between 0 and 1, scales poorly in terms of memory usage and query cost when computing individual entropy values. A binary OctoMap representation can be used to reduce the computation cost during ray casting without significantly affecting results.
Fourth, sensor quality and odometry drift, escalated by poor lighting or the lack of fusion between VO and IMU measurements, directly reduce map accuracy and must be mitigated through sensor fusion or developing an exploration strategy that prioritizes map accuracy.
Mitigations that proved effective in this work include running heavy computations in parallel across different HFSM states, limiting the OctoMap to a binary representation with constrained voxel resolution, enabling ray casting to reduce on unknown grid cell in the 2D occupancy grid, choosing azimuth and elevation step sizes for ray casting that balance accuracy and computation time, and applying frontier filtering to reduce the set of candidate exploration points.

- **How can frontier- and utility-based exploration algorithms be adapted for implementation within a state machine framework?**
  Frontier- and utility-based methods are implemented by decomposing the algorithm into smaller, single-purpose state machines, which serve as the building blocks of HFSM states. This implementation can be divided into a low-level state machine layer, which handles the pipeline of frontier detection, followed by clustering and filtering based on the 2D occupancy grid information, followed by the application of the exploration algorithm based on the list of candidate frontier centroids.
  In this case, this includes using an utility function defined as the ratio between information gain, estimated through ray casting around a sphere of radius equal to the range of the stereo vision camera, and trave cost, computed as the time it would take the rover to get to the centroid based on its pose. Once utility scores are computed, the position of the candidate frontier centroid is published as a navigation goal to the autonomous navigation stack. Apart from the HFSM, the employed state machines handle actions such as triggering actions, monitoring system parameters, gathering information, interfacing with other software, or performing computations for the algorithms.
  The state machine framework architecture allows the definition of global

variables, simplifying how they are accessed within individual state machines. Hence, practical adaptations, such as storing the TF listener in the global variable manager, are beneficial because a new buffer does not need to be created each time a state machine queries a transform in the TF tree, reducing processing time between state machines. They also provide an easy way to incorporate user-defined inputs, making this approach more adaptable to other robotic platforms.

How can the performance of the designed exploration techniques and real time object detection be guaranteed and evaluated across varying operating conditions?

- **What criteria and benchmarks best evaluate the performance and robustness of autonomous exploration techniques?**
  A comprehensive evaluation of exploration strategy performance requires considering parameters such as information gain, travel cost, map accuracy, and computational load. In terms of robustness, the evaluation focuses on how closely the estimated values match the real ones. The key criteria include exploration efficiency, the volume of the environment discovered over time, the RMSE between expected and actual travel distances and times, and the computational load on the OBC in terms of CPU, RAM, and disk usage.
  Additional metrics include the success rate, defined as the proportion of mission scenarios completed without manual intervention, and assessing the total number of voxels that have been discovered both inside and outside the spherical region of interest around each centroid. Because these performance metrics are often plotted as a function of time, a proper way to compare exploration strategy is compare the slopes of each criteria, where steeper slope indicates better performance of the exploration strategy. When exploration time is not considered, the cumulative volume covered as additional centroids are explored serves as a strong indicator of the overall information gain achieved by each strategy, independent of time.
  Although not implemented in this work, an ART system would allow to compare the estimated odometry position by the rover and the actual ground truth, providing more conclusive evidence regarding mapping accuracy. As well, when using repeatable scenarios as benchmarks, it is important to ensure identical starting configurations in terms of ground-truth positions.

- **How does the implemented exploration algorithm compare to state-of-the-art alternatives in the same domain?**
  Using the first experiment within the PEL as a reference to address this research sub question, the most complex exploration algorithm implemented, the *Utility With Edges* strategy which comprises utility-frontier algorithm with frontier coverage, outperforms the *Closest* strategy by 27% in terms of exploration efficiency, measured as the discovered volume per second of exploration time. Moreover, the *Entropy* strategy, which selects the centroid with the highest entropy sum based on ray casting around each centroid,

already shows an improvement of approximately 10.4% in the same domain compared to the *Closest* strategy.
The *Entropy* strategy is already more complex when compared to a similar one on the state-of-the-art which performs the sames entropy computation but just uses a radius around the centroid in the 2D occupancy grid summing the entropy values of each ray in the 2D plane, which is not as complex as ray casting around a 3D sphere. Also, the *Entropy* strategy is already more advanced than the most complex open-source method available in the state-of-the-art prior to this thesis. The referenced approach performs similar entropy computations but rely on a 2D radius around the centroid, summing the entropy values of rays in the 2D plane, rather than conducting full 3D ray casting around a spherical region.
The same conclusion regarding the implemented exploration algorithm and comparison with the state-of-the-art of exploration efficiency is reflected in the total centroid entropy difference as a function of real exploration time, as well as in the total area covered over time. The *Utility With Edges* strategy and *Utility No Edges* strategy always rank first and second, respectively. This is followed by the *Closest* strategy. All in all, the implemented exploration algorithms demonstrate higher complexity than state-of-the-art approaches while also achieving better exploration efficiency.

- **What are the limitations and challenges of achieving real time object detection on a CPU-based platform like the Intel NUC?**
  Real time object detection on CPU-only hardware faces limited compute for accelerated inference, which is translated in fewer frames processed per second compared to GPU-accelerated systems. In principle, real time object detection is possible if the inference frequency is high enough so that no part of the environment is not seen as the rover moves.
  This is defined by the time it takes the camera to rotate across its field of view FOV, given as the ratio between the hFOV and the camera's maximum angular speed. In practice, real time performance requires adopting lightweight YOLO variants. For instance, using OpenVINO to perform model quantization can reduce model size and improve inference speed on Intel hardware with just a CPU.
  Another issue concerns CPU usage, where high CPU load can lead to increased latency and reduced overall performance. However, the second experiment showed that peak inference reached approximately 90% CPU usage without compromising rover operations. Lastly, the majority of the computational load in the object localization loop is due to inference, accounting for 77% of the total loop time, while saving the color and depth frames and computing 3D coordinates from camera intrinsics and extrinsics account for the remaining 23%.

To what extent can integrating autonomous exploration techniques, coupled with 3D object localization, increase the maturity of the autonomous capabilities of the Lunar Rover Mini?

- **How can hierarchical finite state machine architecture be designed in a modular way that supports future development?**
  A modular HFSM design relies on developing each execution state machine with a specific purpose, organized into clear libraries. The advantage of storing states as libraries is that it promotes reusability and easy integration with other state machines developed in the future. Additionally, it allows for easier troubleshooting, as the libraries have already been validated.
  Since the middleware framework is ROS, each state interacts with ROS topics to perform its designated function. Particulary concerning the LRM, the HFSM architecture is employed in RAFCON, which is deployed as a node in the LN Manager of the rover. Its libraries are well-defined, which allows other users to continue to build on top of them. Besides, having less hard-coded variables may require more time during setup, but it ensures modularity and allows the HFSM to be integrated with other robotic platforms relatively easily.

- **What experimental validation methods can be used to assess the performance of the Lunar Rover Mini when generating unknown maps of the environment and detecting objects of interest?**
  A successful experimental validation campaign includes both benchmark scenarios, used to compare performance between different exploration strategies, and full mission scenarios to understand the behavior of the rover over a longer time. These include experiments where the behavior of the rover in the same mission scenario is tested for different exploration strategies, while recording exploration efficiency.
  Further, tests to conclude the accuracy of the 3D localization of OoI. These consist on having proof of concept objects with known ground truth located randomly in the environment, and having the rover move in the environment while the coordinates of the objects are being computed. Then, it can be concluded the different between the estimated and real position of the objects. More importantly, if this different is bigger than a theoretical circular region around the object where the rover will always be able to detect it. Furthermore, a test with the full autonomous exploration and 3D object localization pipeline, where an ART system can be used to save the ground truth position of the rover for later comparison with its estimated odometry.
  Finally, evaluating how changing environmental conditions, for instance lightning or the terrain configuration influences the results.

9.2. Future Work Recommendations

This report allowed to develop an understanding of possible future work that builds upon the research presented here, which are presented below.

More complex utility function: In this work, the utility function used was a

simple ratio between expected information gain and travel cost. However, several other utility functions in the literature use weighting factors to balance these two components. Thus, it would be interesting to explore how using different utility functions, or tuning the weights of the same utility function, affects selecting the next centroid, and what effect does that have on overall exploration efficiency. Also, it might be worth exploration adding another term to the utility function which takes into account map accuracy.

Fusing IMU data with VO for increased robustness: As mentioned when describing the LRM, it only relies on VO to estimate motion, which is prone to drifts. Given that the LRM is equipped with two IMUs, one on the middle board PCB, and one on the Intel RealSense camera, integrating IMU data to improve overall robustness and accuracy of the SLAM pipeline. As well, integrating the IMU data from the camera within the TF tree would allow to properly estimate the pose of the camera frame when rotating the pan-tilt, which is currently not accounted for. This would allow to improve the exploration efficiency when using *Utility With Edges* strategy, since the rover's body would not need to rotate, only the pan-tilt unit.

Using ART during experiments: Since the PEL is equipped with an ART system, it is possible to use this system to track the rover's position while it is moving. This allows to compute the RMSE between the ground truth and the estimated odometry, making it easier when testing exploration strategies that prioritize map accuracy.

Real time pose estimation: As it was described, achieving real time object detection on a CPU was already challenging. Nonetheless, extending this to pose estimation would increase the complexity of the object localization loop. Specifically, pose estimation is important for object grasping, for the end effector to understand the best way to approach the object. However, running inference for pose estimation is expected to take longer than simple object detection, which means it might not be able to run in real time.

Autonomous object grasping: Another recommendation is building on top of this autonomous exploration strategy by implementing autonomous object grasping for a fully autonomous mission pipeline. Since the 3D coordinates of OoI are published to a ROS topic, this pipeline would start by having the rover moving towards those coordinates. From there, it would rely on its local perception for the autonomous grasping.

Testing open-source implementation on other robotic platforms: A key part of this work was making sure that the implemented exploration strategy within the HFSM was open-source and as ready-to-use as possible on other robotic platforms. To achieve this, the HFSM was designed so that user inputs define specific parameters, topics, and thresholds. This allows easy adaptation without changing the code or transitions in each state machine library. This study would allow to identify ways in which the state machines can be even more modified to be more user friendly.

Testing in different more complex environments: One of the conclusions about experiments in the PEL was the fact that lighting conditions affected the OctoMap. None only that, but the tested for the LRM was designed to mimic lunar soil. Nevertheless, the LRM's goal is to be used as a robotic platforms for

researchers and students in different environments. Therefore, it would be beneficial to also study the rover's behavior when it comes to mapping the environment in such conditions. This would also allow to make conclusions about the mechanical design of the rover, mainly if the wheels slip in other terrains.

References

[1] NASA. *The Apollo Program*. Accessed: 16.07.2025. URL: https://www.nasa.gov/the-apollo-program/.

[2] L. A. Vedeshin. 'The First Soviet Experiments on Remote Sensing and Contact Study of the Moon (on the 50th Anniversary of the Moon Landing of Lunokhod 1 Self-Propelled Vehicle)'. In: *Izvestiya, Atmospheric and Oceanic Physics* 57.12 (2021), pp. 1800–1802. ISSN: 1555-628X. DOI: 10.1134/S0001433821120264.

[3] R. A. Cook and A. J. Spear. 'Back to Mars: The Mars Pathfinder mission'. In: *Acta Astronautica* 41.4 (1997). Developing Business, pp. 599–608. ISSN: 0094-5765. DOI: 10.1016/S0094-5765(98)00069-1. URL: https://www.sciencedirect.com/science/article/pii/S0094576598000691.

[4] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. K. Herd, R. Hueso, Y. Liu, J. N. Maki, G. Martinez, R. C. Moeller, A. Nelessen, C. E. Newman, D. Nunes, A. Ponce, N. Spanovich, P. A. Willis, L. W. Beegle, J. F. Bell, A. J. Brown, S.-E. Hamran, J. A. Hurowitz, S. Maurice, D. A. Paige, J. A. Rodriguez-Manfredi, M. Schulte and R. C. Wiens. 'Mars 2020 Mission Overview'. In: *Space Science Reviews* 216.8 (2020), p. 142. ISSN: 1572-9672. DOI: 10.1007/s11214-020-00762-y.

[5] M. Winnendael, P. Baglioni and J. Vago. 'Development of the ESA ExoMars rover'. In: (Aug. 2005).

[6] S. Ulamec, P. Michel, M. Grott, U. Böttger, S. Schröder, H.-W. Hübers, Y. Cho, F. Rull, N. Murdoch, P. Vernazza, O. Prieto-Ballesteros, J. Biele, S. Tardivel, D. Arrat, T. Hagelschuer, J. Knollenberg, D. Vivet, C. Sunday, L. Jorda, O. Groussin, C. Robin and H. Miyamoto. 'Science objectives of the MMX rover'. In: *Acta Astronautica* 210 (2023), pp. 95–101. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2023.05.012. URL: https://www.sciencedirect.com/science/article/pii/S0094576523002448.

[7] M. Vayugundla, T. Bodenmüller, L. Burkhard, M. J. Schuster, B.-M. Steinmetz, M. Sewtz, N. Borgsmüller, F. Buse, W. Stürzl, R. Giubilato *et al.* 'The DLR Autonomous Navigation Experiment with the IDEFIX Rover: Software Architecture, Autonomous Navigation Features and Preliminary Operations Concept'. In: *2025 IEEE Aerospace Conference*. IEEE. 2025, pp. 1–20.

[8] B. Yamauchi. *A Frontier-Based Approach for Autonomous Exploration*. Tech. rep. 1997.

[9] F. Amigoni and A. Gallo. 'A Multi-Objective Exploration Strategy for Mobile Robots'. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 3850–3855. DOI: 10.1109/ROBOT.2005.1570708.

[10] E. Uslu, F. Çakmak, M. Balcılar, A. Akıncı, M. F. Amasyalı and S. Yavuz. 'Implementation of frontier-based exploration algorithm for an autonomous robot'. In: *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*. 2015, pp. 1–7. DOI: 10.1109/INISTA.2015.7276723.

[11] A. Topiwala, P. Inani and A. Kathpal. 'Frontier Based Exploration for Autonomous Robot'. In: (June 2018).

[12] D. Holz, N. Basilico, F. Amigoni and S. Behnke. 'Evaluating the Efficiency of Frontier-based Exploration Strategies'. In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. 2010, pp. 1–8.

[13] J. Faigl and M. Kulich. 'On benchmarking of frontier-based multi-robot exploration strategies'. In: *2015 European Conference on Mobile Robots (ECMR)*. IEEE, Sept. 2015, pp. 1–8. ISBN: 978-1-4673-9163-4. DOI: 10.1109/ECMR.2015.7324183.

[14] U. Jain, R. Tiwari and W. W. Godfrey. 'Comparative study of frontier based exploration methods'. In: *2017 Conference on Information and Communication Technology (CICT)*. IEEE, Nov. 2017, pp. 1–5. ISBN: 978-1-5386-1866-0. DOI: 10.1109/INFOCOMTECH.2017.8340589.

[15] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[16] Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye. 'Object Detection in 20 Years: A Survey'. In: *Proceedings of the IEEE* 111.3 (Mar. 2023), pp. 257–276. ISSN: 0018-9219. DOI: 10.1109/JPROC.2023.3238524.

[17] M. J. Schuster, S. G. Brunner, K. Bussmann, S. Büttner, A. Dömel, M. Hellerer, H. Lehner, P. Lehner, O. Porges, J. Reill, S. Riedel, M. Vayugundla, B. Vodermayer, T. Bodenmüller, C. Brand, W. Friedl, I. Grixa, H. Hirschmüller, M. Kaßecker, Z.-C. Márton, C. Nissler, F. Ruess, M. Suppa and A. Wedler. 'Towards Autonomous Planetary Exploration'. In: *Journal of Intelligent & Robotic Systems* 93.3 (2019), pp. 461–494. ISSN: 1573-0409. DOI: 10.1007/s10846-017-0680-9.

[18] S. G. Brunner, F. Steinmetz, R. Belder and A. Dömel. 'RAFCON: A Graphical Tool for Task Programming and Mission Control'. In: *RoboCup 2016: Robot World Cup XX*. Ed. by S. Behnke, R. Sheh, S. Sarıel and D. D. Lee. Cham: Springer International Publishing, 2017, pp. 347–355. ISBN: 978-3-319-68792-6.

[19] M. Voellmy and M. Ehrhardt. ‘ExoMy: A Low Cost 3D Printed Rover’. In: Oct. 2020.

[20] A. Arka. *Autonomous Explorer and Mapper for ROS 2 Nav2*. https://github.com/AniArka/Autonomous-Explorer-and-Mapper-ros2-nav2. Accessed: 2025-09-14. 2025. URL: https://github.com/AniArka/Autonomous-Explorer-and-Mapper-ros2-nav2.

[21] H. Umari and S. Mukhopadhyay. ‘Autonomous robotic exploration based on multiple rapidly-exploring randomized trees’. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017, pp. 1396–1402. ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8202319.

[22] D. C. Conner and J. Willis. ‘Flexible Navigation: Finite state machine-based integrated navigation and control for ROS enabled robots’. In: *SoutheastCon 2017*. IEEE, Mar. 2017, pp. 1–8. ISBN: 978-1-5386-1539-3. DOI: 10.1109/SECON.2017.7925266.

[23] R. Balogh and D. Obdržálek. ‘Using Finite State Machines in Introductory Robotics’. In: 2019, pp. 85–91. DOI: 10.1007/978-3-319-97085-1{_}9.

[24] W. CARRIER III. ‘Soviet rover systems’. In: *Space Programs and Technologies Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics, Mar. 1992. DOI: 10.2514/6.1992-1487.

[25] E. G. Cowart. ‘Lunar Roving Vehicle: Spacecraft on Wheels’. In: *Proceedings of the Institution of Mechanical Engineers* 187.1 (June 1973), pp. 463–491. ISSN: 0020-3483. DOI: 10.1243/PIME{_}PROC{_}1973{_}187{_}132{_}02.

[26] J. Matijevic. *Sojourner: The Mars Pathfinder Microrover Flight Experiment*. 1997. DOI: 2014/21704. URL: https://hdl.handle.net/2014/21704.

[27] L. A. D’Amario. ‘Mars exploration rovers navigation results’. In: *The Journal of the Astronautical Sciences* 54.2 (2006), pp. 129–173. ISSN: 0021-9142. DOI: 10.1007/BF03256481. URL: https://doi.org/10.1007/BF03256481.

[28] A. R. Vasavada. ‘Mission Overview and Scientific Contributions from the Mars Science Laboratory Curiosity Rover After Eight Years of Surface Operations’. In: *Space Science Reviews* 218.3 (2022), p. 14. ISSN: 1572-9672. DOI: 10.1007/s11214-022-00882-7. URL: https://doi.org/10.1007/s11214-022-00882-7.

[29] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. K. Herd, R. Hueso, Y. Liu, J. N. Maki, G. Martinez, R. C. Moeller, A. Nelessen, C. E. Newman, D. Nunes, A. Ponce, N. Spanovich, P. A. Willis, L. W. Beegle, J. F. Bell, A. J. Brown, S.-E. Hamran, J. A. Hurowitz, S. Maurice, D. A. Paige, J. A. Rodriguez-Manfredi, M. Schulte and R. C. Wiens. 'Mars 2020 Mission Overview'. In: *Space Science Reviews* 216.8 (2020), p. 142. ISSN: 1572-9672. DOI: 10.1007/s11214-020-00762-y. URL: https://doi.org/10.1007/s11214-020-00762-y.

[30] M. Winnendael, P. Baglioni and J. Vago. 'Development of the ESA ExoMars rover'. In: (Apr. 2005).

[31] N. Patel, R. Slade and J. Clemmet. 'The ExoMars rover locomotion subsystem'. In: *Journal of Terramechanics* 47.4 (2010), pp. 227–242. ISSN: 0022-4898. DOI: 10.1016/j.jterra.2010.02.004. URL: https://www.sciencedirect.com/science/article/pii/S0022489810000182.

[32] S. Bekkers, M. T. Bettendorf, J. Reill, R. Giubilato and A. Wedler. 'The Lunar Rover Mini: towards a Versatile, Open-Source Mobile Robotic Platform for Educational and Experimental Purposes'. In: *17th Symposium on Advanced Space Technologies in Robotics and Automation*. Oct. 2023. URL: https://elib.dlr.de/202313/.

[33] A. Ricardez Ortigosa. 'Systems Integration with Autonomous Navigation of the Lunar Rover Mini for a Space Demo Mission'. PhD thesis. TU Berlin, 2023. URL: https://elib.dlr.de/200804/.

[34] Sam Bekkers. *Sensor Fusion for Visual-Inertial Simultaneous Localisation and Mapping*. 2024. URL: https://resolver.tudelft.nl/uuid:06bf2239-db11-41ee-bab2-a90e95b3ceb8.

[35] M. Conenna. *Elasto-Kinematic Calibration of the Lunar Rover Mini 6DOF Robotic Arm*. 2024. URL: https://elib.dlr.de/212024/.

[36] N. Patel, R. Slade and J. Clemmet. 'The ExoMars rover locomotion subsystem'. In: *Journal of Terramechanics* 47.4 (2010), pp. 227–242. ISSN: 0022-4898. DOI: https://doi.org/10.1016/j.jterra.2010.02.004. URL: https://www.sciencedirect.com/science/article/pii/S0022489810000182.

[37] Florian Schmidt. *Links and Nodes Manager*. Accessed: 17.07.2025. 2020. URL: https://gitlab.com/links%5C_and%5C_nodes/links%5C_and%5C_nodes.

[38] J. 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.

[39] M. Calonder, V. Lepetit, C. Strecha and P. Fua. 'BRIEF: Binary Robust Independent Elementary Features'. In: *Computer Vision – ECCV 2010*. Ed. by K. Daniilidis, P. Maragos and N. Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792. ISBN: 978-3-642-15561-1.

[40] M. Labbé and F. Michaud. 'RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation'. In: *Journal of Field Robotics* 36.2 (Mar. 2019), pp. 416–446. ISSN: 1556-4959. DOI: 10.1002/rob.21831.

[41] F. Dellaert. 'Factor Graphs and GTSAM: A Hands-on Introduction'. In: 2012. URL: https://api.semanticscholar.org/CorpusID:131215724.

[42] P. Hart, N. Nilsson and B. Raphael. 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[43] W. Gao, M. Booker, A. Adiwahono, M. Yuan, J. Wang and Y. W. Yun. 'An improved Frontier-Based Approach for Autonomous Exploration'. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE, Nov. 2018, pp. 292–297. ISBN: 978-1-5386-9582-1. DOI: 10.1109/ICARCV.2018.8581245.

[44] D. L. da Silva Lubanco, M. Pichler-Scheder, T. Schlechter, M. Scherhaufl and C. Kastl. 'A Review of Utility and Cost Functions Used in Frontier-Based Exploration Algorithms'. In: *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*. IEEE, Nov. 2020, pp. 187–191. ISBN: 978-1-7281-8981-9. DOI: 10.1109/ICRAE50850.2020.9310862.

[45] Y. Sung, D. Dixit and P. Tokekar. 'Online Exploration of an Unknown Region of Interest with a Team of Aerial Robots'. In: (Nov. 2018).

[46] H. H. González-Baños and J.-C. Latombe. 'Navigation Strategies for Exploring Indoor Environments'. In: *The International Journal of Robotics Research* 21.10-11 (2002), pp. 829–848. DOI: 10.1177/027836490202101083 4. URL: https://doi.org/10.1177/027836490202101083 4.

[47] F. Bourgault, A. Makarenko, S. Williams, B. Grocholsky and H. Durrant-Whyte. 'Information based adaptive robotic exploration'. In: *IEEE/RSJ International Conference on Intelligent Robots and System*. IEEE, pp. 540–545. ISBN: 0-7803-7398-7. DOI: 10.1109/IRDS.2002.1041446.

[48] H. Carrillo, P. Dames, V. Kumar and J. A. Castellanos. 'Autonomous robotic exploration using occupancy grid maps and graph SLAM based on Shannon and Rényi Entropy'. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015, pp. 487–494. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139224.

[49] S. Entropy, R. Entropy, I. P. A. Bromiley, N. A. Thacker and E. V. Bouhova-Thacker. 'Shannon Entropy, Renyi Entropy, and Information'. In: 2004. URL: https://api.semanticscholar.org/CorpusID:17565994.

[50] A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas and S. Leutenegger. 'Fast Frontier-based Information-driven Autonomous Exploration with an MAV'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020, pp. 9570–9576. ISBN: 978-1-7281-7395-5. DOI: 10.1109/ICRA40945.2020.9196707.

[51] D. L. da Silva Lubanco, M. Pichler-Scheder and T. Schlechter. 'A Novel Frontier-Based Exploration Algorithm for Mobile Robots'. In: *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*. 2020, pp. 1–5. DOI: 10.1109/ICMRE49073.2020.9064866.

[52] C. Gomez, A. C. Hernandez and R. Barber. 'Topological Frontier-Based Exploration and Map-Building Using Semantic Information'. In: *Sensors* 19.20 (2019). ISSN: 1424-8220. DOI: 10.3390/s19204595. URL: https://www.mdpi.com/1424-8220/19/20/4595.

[53] P. Li, C.-y. Yang, R. Wang and S. Wang. 'A high-efficiency, information-based exploration path planning method for active simultaneous localization and mapping'. In: *International Journal of Advanced Robotic Systems* 17.1 (2020), p. 1729881420903207. DOI: 10.1177/1729881420903207. URL: https://doi.org/10.1177/1729881420903207.

[54] B. Yamauchi. 'Frontier-based exploration using multiple robots'. In: *Proceedings of the Second International Conference on Autonomous Agents*. AGENTS '98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 47–53. ISBN: 0-89791-983-1. DOI: 10.1145/280765.280773. URL: https://doi.org/10.1145/280765.280773.

[55] W. Burgard, M. Moors and F. Schneider. 'Collaborative Exploration of Unknown Environments with Teams of Mobile Robots'. In: *Advances in Plan-Based Control of Robotic Agents*. Ed. by M. Beetz, J. Hertzberg, M. Ghallab and M. E. Pollack. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 52–70. ISBN: 978-3-540-37724-5.

[56] J. Kuckling, R. Luckey, V. Avrutin, A. Vardy, A. Reina and H. Hamann. 'Do We Run Large-scale Multi-Robot Systems on the Edge? More Evidence for Two-Phase Performance in System Size Scaling'. In: (Oct. 2023).

[57] E. Renshaw and R. Henderson. 'The correlated random walk'. In: *Journal of Applied Probability* 18.2 (June 1981), pp. 403–414. ISSN: 0021-9002. DOI: 10.2307/3213286.

[58] V. Zaburdaev, S. Denisov and J. Klafter. 'Lévy walks'. In: *Reviews of Modern Physics* 87.2 (June 2015), pp. 483–530. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.87.483.

[59] Z. Pasternak, F. Bartumeus and F. Grasso. ‘Lévy-taxis: A novel search strategy for finding odor plumes in turbulent flow-dominated environments’. In: *Journal of Physics A: Mathematical and Theoretical* 42 (Oct. 2009), p. 434010. DOI: 10.1088/1751-8113/42/43/434010.

[60] A. Bircher, M. S. Kamel, K. Alexis, H. Oleynikova and R. Siegwart. ‘Receding Horizon "Next-Best-View" Planner for 3D Exploration’. In: Oct. 2016, pp. 1462–1468. DOI: 10.1109/ICRA.2016.7487281.

[61] M. Selin, M. Tiger, D. Duberg, F. Heintz and P. Jensfelt. ‘Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments’. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 1699–1706. ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2897343.

[62] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart and J. Nieto. ‘An Efficient Sampling-Based Method for Online Informative Path Planning in Unknown Environments’. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1500–1507. DOI: 10.1109/LRA.2020.2969191.

[63] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006. ISBN: 978-0-521-86205-9. DOI: 10.1017/CBO9780511546877.

[64] S. M. LaValle. ‘Rapidly-exploring random trees : a new tool for path planning’. In: *The annual research report* (1998). URL: https://api.semanticscholar.org/CorpusID:14744621.

[65] L. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars. ‘Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces’. In: *Robotics and Automation, IEEE Transactions on* 12 (Oct. 1996), pp. 566–580. DOI: 10.1109/70.508439.

[66] S. Karaman and E. Frazzoli. *Sampling-based Algorithms for Optimal Motion Planning*. 2011. URL: https://arxiv.org/abs/1105.1186.

[67] D. K. Muhsen, F. A. Raheem and A. T. Sadiq. ‘A Systematic Review of Rapidly Exploring Random Tree RRT Algorithm for Single and Multiple Robots’. In: *Cybernetics and Information Technologies* 24.3 (2024), pp. 78–101. DOI: 10.2478/cait-2024-0026. URL: https://doi.org/10.2478/cait-2024-0026.

[68] I. Noreen, A. Khan and Z. Habib. ‘Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions’. In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016). ISSN: 21565570. DOI: 10.14569/IJACSA.2016.071114.

[69] G. Oriolo, M. Vendittelli, L. Freda and G. Troso. ‘The SRT method: randomized strategies for exploration’. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 5. 2004, pp. 4688–4694. DOI: 10.1109/ROBOT.2004.1302457.

[70] Z. Xu, D. Deng and K. Shimada. 'Autonomous UAV Exploration of Dynamic Environments Via Incremental Sampling and Probabilistic Roadmap'. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2729–2736. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3062008.

[71] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot. 'Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic'. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

[72] H. Liu, X. Zhang, J. Wen, R. Wang and X. Chen. 'Goal-biased Bidirectional RRT based on Curve-smoothing'. In: *IFAC-PapersOnLine* 52.24 (2019), pp. 255–260. ISSN: 24058963. DOI: 10.1016/j.ifacol.2019.12.417.

[73] J. J. Kuffner and S. M. LaValle. 'RRT-connect: An efficient approach to single-query path planning'. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, pp. 995–1001. DOI: 10.1109/ROBOT.2000.844730.

[74] L. Freda and G. Oriolo. 'Frontier-Based Probabilistic Strategies for Sensor-Based Exploration'. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 3881–3887. DOI: 10.1109/ROBOT.2005.1570713.

[75] S. Long, Y. Li, C. Wu, B. Xu and W. Fan. 'HPHS: Hierarchical Planning based on Hybrid Frontier Sampling for Unknown Environments Exploration'. In: (July 2024).

[76] Y. Ning, T. Li, C. Yao, W. Du and Y. Zhang. 'HMS-RRT: A novel hybrid multi-strategy rapidly-exploring random tree algorithm for multi-robot collaborative exploration in unknown environments'. In: *Expert Systems with Applications* 247 (2024), p. 123238. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.123238. URL: https://www.sciencedirect.com/science/article/pii/S0957417424001039.

[77] C. Wang, H. Ma, W. Chen, L. Liu and M. Q.-H. Meng. 'Efficient Autonomous Exploration With Incrementally Built Topological Map in 3-D Environments'. In: *IEEE Transactions on Instrumentation and Measurement* 69.12 (Dec. 2020), pp. 9853–9865. ISSN: 0018-9456. DOI: 10.1109/TIM.2020.3001816.

[78] K. Leong. 'Reinforcement Learning with Frontier-Based Exploration via Autonomous Environment'. In: (July 2023).

[79] G. Cai, L. Guo and X. Chang. 'An Enhanced Hierarchical Planning Framework for Multi-Robot Autonomous Exploration'. In: (Oct. 2024).

[80] B. Sun, H. Chen, S. Leutenegger, C. Cadena, M. Pollefeys and H. Blum. 'FrontierNet: Learning Visual Cues to Explore'. In: (Jan. 2025).

[81] H. Zhang, G. Lyu, Y. Xie, F. Dong and K. Chen. 'LCFP-RRT: A Robot Exploration Algorithm Based on Local Constrained Sampling and Frontier Prioritization Classification'. In: 2025, pp. 36–45. DOI: 10.1007/978-981-96-4756-9{_}4.

[82] S. A. Dumitru, L. Vladareanu, T. H. Yan and C. K. Qi. 'Mobile Robot Navigation Techniques Using Potential Field Method in Unknown Environments'. In: *Applied Mechanics and Materials* 656 (Oct. 2014), pp. 388–394. ISSN: 1662-7482. DOI: 10.4028/www.scientific.net/AMM.656.388.

[83] R. N. Darmanin and M. K. Bugeja. 'Autonomous Exploration and Mapping using a Mobile Robot Running ROS'. In: *International Conference on Informatics in Control, Automation and Robotics*. 2016. URL: https://api.semanticscholar.org/CorpusID:13646501.

[84] S. Macenski, F. Martín, R. White and J. Ginés Clavero. 'The Marathon 2: A Navigation System'. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020. URL: https://github.com/ros-planning/navigation2.

[85] H. Umari and S. Mukhopadhyay. 'Autonomous robotic exploration based on multiple rapidly-exploring randomized trees'. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017, pp. 1396–1402. ISBN: 978-1-5386-2682-5. DOI: 10.1109/IROS.2017.8202319.

[86] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard. 'OctoMap: an efficient probabilistic 3D mapping framework based on octrees'. In: *Autonomous Robots* 34.3 (2013), pp. 189–206. ISSN: 1573-7527. DOI: 10.1007/s10514-012-9321-0. URL: https://doi.org/10.1007/s10514-012-9321-0.

[87] R. Brooks. 'A robust layered control system for a mobile robot'. In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 0882-4967. DOI: 10.1109/JRA.1986.1087032.

[88] O. Ahmad, J. Cremer, J. Kearney, P. Willemsen and S. Hansen. 'Hierarchical, concurrent state machines for behavior modeling and scenario control'. In: *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*. 1994, pp. 36–42. DOI: 10.1109/AIHAS.1994.390503.

[89] V. S. Alagar and K. Periyasamy. 'Extended Finite State Machine'. In: 2011, pp. 105–128. DOI: 10.1007/978-0-85729-277-3{_}7.

[90] D. Brand and P. Zafiropulo. 'On Communicating Finite-State Machines'. In: *Journal of the ACM* 30.2 (Apr. 1983), pp. 323–342. ISSN: 0004-5411. DOI: 10.1145/322374.322380.

[91] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta and R. Carrasco. 'Probabilistic finite-state machines - part I'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.7 (July 2005), pp. 1013–1025. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.147.

[92] M. Steinbrink, P. Koch, S. May, B. Jung and M. Schmidpeter. 'State Machine for Arbitrary Robots for Exploration and Inspection Tasks'. In: *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*. ICVISP 2020. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 978-1-4503-8953-2. DOI: 10.1145/3448823.3448857. URL: https://doi.org/10.1145/3448823.3448857.

[93] J. Bohren and S. Cousins. 'The SMACH High-Level Executive [ROS News]'. In: *IEEE Robotics & Automation Magazine* 17.4 (Dec. 2010), pp. 18–20. ISSN: 1070-9932. DOI: 10.1109/MRA.2010.938836.

[94] P. Schillinger, S. Kohlbrecher and O. von Stryk. 'Human-robot collaborative high-level control with application to rescue robotics'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016, pp. 2796–2802. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487442.

[95] M. Á. González-Santamarta, F. J. Rodríguez-Lera, V. Matellán-Olivera and C. Fernández-Llamas. 'YASMIN: Yet Another State MachINe'. In: 2023, pp. 528–539. DOI: 10.1007/978-3-031-21062-4{_}43.

[96] S. G. Brunner, F. Steinmetz, R. Belder and A. Domel. 'RAFCON: A graphical tool for engineering complex, robotic tasks'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016, pp. 3283–3290. ISBN: 978-1-5090-3762-9. DOI: 10.1109/IROS.2016.7759506.

[97] M. J. Schuster, S. G. Brunner, K. Bussmann, S. Büttner, A. Dömel, M. Hellerer, H. Lehner, P. Lehner, O. Porges, J. Reill, S. Riedel, M. Vayugundla, B. Vodermayer, T. Bodenmüller, C. Brand, W. Friedl, I. Grixa, H. Hirschmüller, M. Kaßecker, Z.-C. Márton, C. Nissler, F. Ruess, M. Suppa and A. Wedler. 'Towards Autonomous Planetary Exploration'. In: *Journal of Intelligent & Robotic Systems* 93.3 (2019), pp. 461–494. ISSN: 1573-0409. DOI: 10.1007/s10846-017-0680-9. URL: https://doi.org/10.1007/s10846-017-0680-9.

[98] D. G. Lowe. 'Distinctive Image Features from Scale-Invariant Keypoints'. In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94.

[99] H. Bay, T. Tuytelaars and L. Van Gool. 'SURF: Speeded Up Robust Features'. In: 2006, pp. 404–417. DOI: 10.1007/11744023{_}32.

[100] R. Girshick, J. Donahue, T. Darrell and J. Malik. 'Rich feature hierarchies for accurate object detection and semantic segmentation'. In: (Oct. 2014).

[101] R. Girshick. 'Fast R-CNN'. In: (Sept. 2015).

[102] S. Ren, K. He, R. Girshick and J. Sun. 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks'. In: (Jan. 2016).

[103] J. Redmon, S. Divvala, R. Girshick and A. Farhadi. 'You Only Look Once: Unified, Real-Time Object Detection'. In: (May 2016).

[104] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg. 'SSD: Single Shot MultiBox Detector'. In: (Dec. 2016). DOI: 10.1007/978-3-319-46448-0{_}2.

[105] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár. 'Focal Loss for Dense Object Detection'. In: (Feb. 2018).

[106] R. Khanam and M. Hussain. 'YOLOv11: An Overview of the Key Architectural Enhancements'. In: (Oct. 2024).

[107] D. H. Dos Reis, D. Welfer, M. A. De Souza Leite Cuadros and D. F. T. Gamarra. 'Mobile Robot Navigation Using an Object Recognition Software with RGBD Images and the YOLO Algorithm'. In: *Applied Artificial Intelligence* 33.14 (Dec. 2019), pp. 1290–1305. ISSN: 0883-9514. DOI: 10.1080/08839514.2019.1684778.

[108] H. Zhao, Z. Tang, Z. Li, Y. Dong, Y. Si, M. Lu and G. Panoutsos. 'Real-Time Object Detection and Robotic Manipulation for Agriculture Using a YOLO-Based Learning Approach'. In: *2024 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, Mar. 2024, pp. 1–6. ISBN: 979-8-3503-4026-6. DOI: 10.1109/ICIT58233.2024.10540740.

[109] Q. Ul Islam, F. Khozaei, E. M. Salah Al Barhoumi, I. Baig and D. Ignatyev. 'Advancing autonomous SLAM systems: Integrating YOLO object detection and enhanced loop closure techniques for robust environment mapping'. In: *Robotics and Autonomous Systems* 185 (Mar. 2025), p. 104871. ISSN: 09218890. DOI: 10.1016/j.robot.2024.104871.

[110] T. Dang, C. Papachristos and K. Alexis. 'Autonomous exploration and simultaneous object search using aerial robots'. In: *2018 IEEE Aerospace Conference*. IEEE, Mar. 2018, pp. 1–7. ISBN: 978-1-5386-2014-4. DOI: 10.1109/AERO.2018.8396632.

[111] H. Kim, H. Kim, S. Lee and H. Lee. 'Autonomous Exploration in a Cluttered Environment for a Mobile Robot With 2D-Map Segmentation and Object Detection'. In: *IEEE Robotics and Automation Letters* 7.3 (July 2022), pp. 6343–6350. ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3171069.

[112] A. Srivastava. *Sense-Plan-Act in Robotic Applications*. Oct. 2019. DOI: 10.13140/RG.2.2.21308.36481.

[113] R. Murphy. *Introduction to AI robotics*. MIT Press, 2000, p. 466. ISBN: 978-0-262-13383-8.

[114] Â. Morgado, C. Gonçalves and N. Pombo. 'Enhancing Autonomous Vehicles: An Experimental Analysis of Path Planning and Decision-Making Processes through Simulink-Based Architecture'. In: Oct. 2023, pp. 1–9. DOI: 10.1109/AICT59525.2023.10313190.

[115] D. Scaramuzza and F. Fraundorfer. 'Visual Odometry [Tutorial]'. In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[116] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Mar. 2004. ISBN: 978-0-521-54051-3. DOI: 10.1017/CBO9780511811685.

[117] B. Triggs, P. F. McLauchlan, R. I. Hartley and A. W. Fitzgibbon. 'Bundle Adjustment — A Modern Synthesis'. In: 2000, pp. 298–372. DOI: 10.1007/3-540-44480-7{_}21.

[118] H. Durrant-Whyte and T. Bailey. 'Simultaneous localization and mapping: part I'. In: *IEEE Robotics & Automation Magazine* 13.2 (June 2006), pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[119] G. Grisetti, R. Kümmerle, C. Stachniss and W. Burgard. 'A Tutorial on Graph-Based SLAM'. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[120] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard. 'G2o: A general framework for graph optimization'. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011, pp. 3607–3613. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5979949.

[121] S. Thrun, W. Burgard and D. Fox. 'Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)'. In: Oct. 2005. ISBN: 0-262-20162-3.

[122] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV]. URL: https://arxiv.org/abs/2207.02696.

[123] C. Stachniss, G. Grisetti and W. Burgard. 'Information Gain-based Exploration Using Rao-Blackwellized Particle Filters'. In: *Robotics: Science and Systems I*. Robotics: Science and Systems Foundation, June 2005. ISBN: 978-0-262-70114-3. DOI: 10.15607/RSS.2005.I.009.

[124] G. Metta, P. Fitzpatrick and L. Natale. 'YARP: Yet Another Robot Platform'. In: *International Journal of Advanced Robotic Systems* 3.1 (Mar. 2006). ISSN: 1729-8806. DOI: 10.5772/5761.

[125] A. Grati. 'MOOS-IvP: Cross Platform Software for Robotics Research'. In: *NATO's Science and Technology Organization (STO)* (2015), pp. 1–14.

[126] D. J. Withey and B. T. Matebese. 'An OctoMap-based 3D CostMap'. In: *2021 Rapid Product Development Association of South Africa - Robotics and Mechatronics - Pattern Recognition Association of South Africa (RAPDASA-RobMech-PRASA)*. IEEE, Nov. 2021, pp. 01–06. ISBN: 978-1-6654-0803-5. DOI: 10.1109/RAPDASA-RobMech-PRAS53819.2021.9829092.

[127] A. Asgharivaskasi and N. Atanasov. 'Semantic OcTree Mapping and Shannon Mutual Information Computation for Robot Exploration'. In: *IEEE Transactions on Robotics* 39.3 (June 2023), pp. 1910–1928. ISSN: 1552-3098. DOI: 10.1109/TRO.2023.3245986.

[128] V. S. Engelschiøn, S. R. Eriksson, A. Cowley, M. Fateri, A. Meurisse, U. Kueppers and M. Sperl. 'EAC-1A: A novel large-volume lunar regolith simulant'. In: *Scientific Reports* 10.1 (Mar. 2020), p. 5473. ISSN: 2045-2322. DOI: 10.1038/s41598-020-62312-4.

[129] L. Meyer, M. Vayugundla, P. Kenny, M. Smíšek, J. Biele, A. Maturilli, M. G. Müller, W. Stürzl, M. J. Schuster, T. Bodenmüller, A. Wedler and R. Triebel. 'Testing for the MMX Rover Autonomous Navigation Experiment on Phobos'. In: *2023 IEEE Aerospace Conference*. IEEE, Mar. 2023, pp. 1–19. ISBN: 978-1-6654-9032-0. DOI: 10.1109/AERO55745.2023.10115919.

A

Comparison of Low-Cost COTS Built Open-Source Rovers

Rover	Dimensions	Mass	Embedded Capabilities	Max. Velocity	Autonomoy	Price	Software
ASURO	45x122x117 mm	0.165 kg	Basic object detection and avoidance	-	-	50 €	C
OSR	610x305x356 mm	12.7 kg	LED head information display	1.75 m/s	5 h	2160 €	Linux, Python
Sawppy	685x320x306 mm	-	Simple camera support	-	-	430 €	ROS, C, C++
ExoMy	300x390x420 mm	2.5 kg	2D RGB visual imaging	0.23 m/s	3 h	360 €	ROS, Python
LRM	360x260x390 mm	3.7 kg	RGB-D vision, autonomous nav. & exploration, 6-DOF robotic arm, object detection	0.13 m/s	3 h	2000 €	ROS, Python, C++, Linux

Table A.1.: Comparison of low-cost COTS built open-source rovers.

B

ROS Message and Topics Definitions

This appendix lists the main ROS message types referenced throughout this work, along with links to their official documentation for complete definitions, as well as the description of the ROS topics used throughout this report.

B.1. ROS Messages

`sensor_msgs/PointCloud2`

`octomap_msgs/Octomap`

`std_msgs/Bool`

`nav_msgs/OccupancyGrid`

`visualization_msgs/Marker`

`visualization_msgs/MarkerArray`

`sensor_msgs/Image`

B.2. ROS Topics

`/trigger_saver`: Subscribing to a Bool message and triggers the custom node `octomap_saver_node.cpp` to save the current OctoMap.

`/system_stats`: Provides real time information about the computational load and system performance, including CPU, RAM, and disk, to monitor whether the rover operates within the processing capabilities of the Intel NUC.

`/grid_map`: Publishes a 2D occupancy grid of the environment generated from the point cloud, used for navigation and visualization.

`/octomap_full`: Contains the full 3D OctoMap representation of the environment, indicating occupied, free, and unknown space for the exploration strategy to use.

`/odom`: Provides the rover's odometry data, including position and orientation estimates, used for trajectory tracking and SLAM evaluation.

`/candidate_frontiers`: Publishes the set of candidate frontier points identified by the exploration algorithm, which represent potential areas to explore.

`/segmented_frontiers`: Contains clusters of frontier points segmented from the raw candidate frontiers.

`/segmented_frontiers_centroids`: Publishes the centroid positions of the segmented frontier clusters, which will be used by the utility-based exploration algorithm.

`/centroids_sphere_view`: Publishes the ray casting sphere around each centroid for entropy to be computed.

`/utility_function`: Contains the computed utility values for each candidate centroid.

`/normalized_entropy`: Publishes the normalized entropy associated with each centroid.

`/cloud_map`: Contains the accumulated 3D point cloud generated from stereo vision camera data, representing the explored environment in detail.

`/map_stats`: Provides the total area that has been covered so far by the rover.

`/detected_objects`: Publishes the class labels and 3D coordinates of objects detected by the YOLOv7-based custom model, used for object localization and visualization in RViz.

C

First Experiment Mission Scenarios

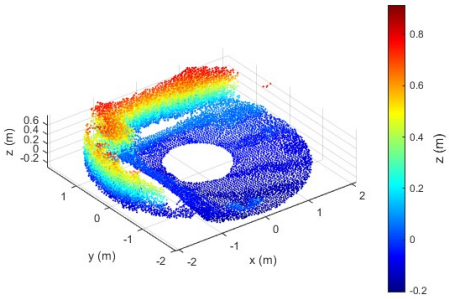


Figure C.1.: Mission scenario 1.

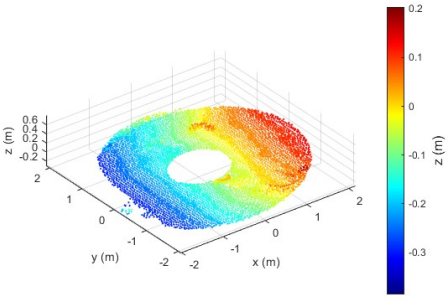


Figure C.3.: Mission scenario 3.

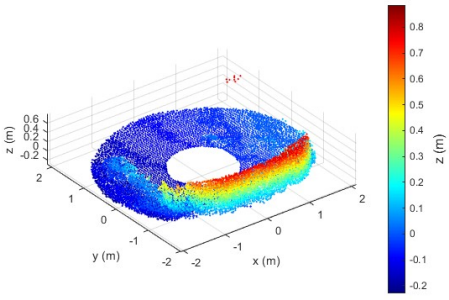


Figure C.2.: Mission scenario 2.

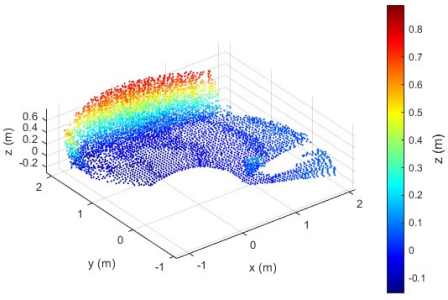


Figure C.4.: Mission scenario 4.

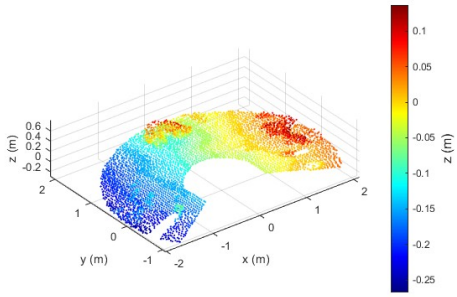


Figure C.5.: Mission scenario 5.

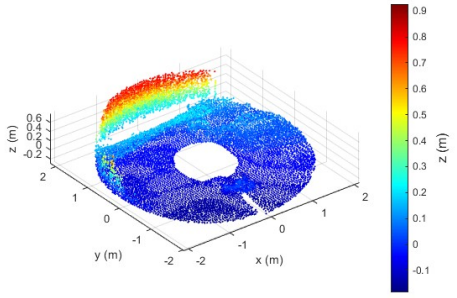


Figure C.6.: Mission scenario 6.

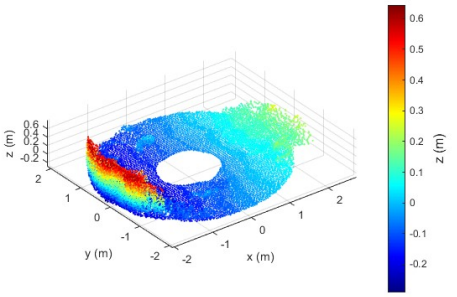


Figure C.7.: Mission scenario 7.

D

3D Point Cloud and Rover Odometry Overlay

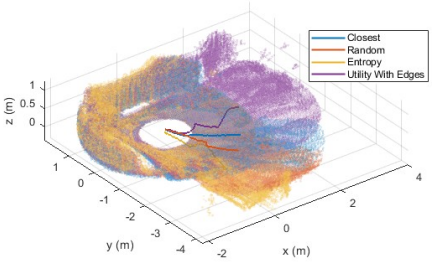


Figure D.1.: Mission scenario 1.

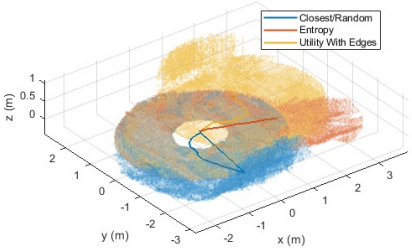


Figure D.3.: Mission scenario 3.

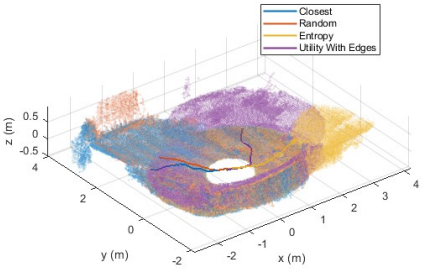


Figure D.2.: Mission scenario 2.

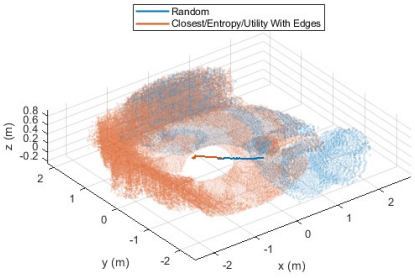


Figure D.4.: Mission scenario 4.

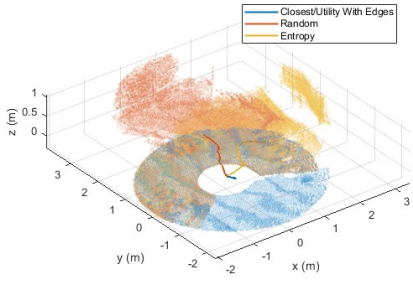


Figure D.5.: Mission scenario 5.

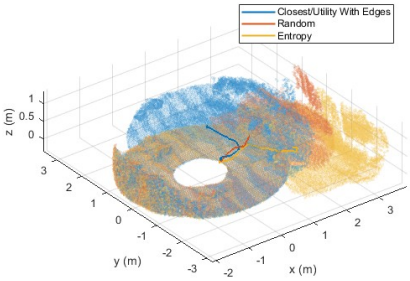


Figure D.7.: Mission scenario 7.

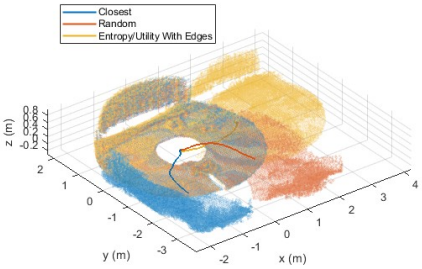


Figure D.6.: Mission scenario 6.

E

iSpaRo 2025 Conference Paper

During the writing of my master thesis I was the main author of a paper submitted and accepted to the proceedings of International Conference on Space Robotics 2025.

A Modular Open-Source Rover for Space Robotics Research

Tomás Plácido de Castro[1,2], Jui-Wen Yeh[1], Ivan Gilberto Martin Enciso[1], Hugo Allard[1], Armin Wedler[1], Josef Reill and Riccardo Giubilato[1]

***Abstract*— This paper introduces the Lunar Rover Mini (LRM), an open-source, low cost mobile robotic platform developed for space robotics research. Started in 2015 at DLR's Institute of Robotics and Mechatronics, the LRM is built using off-the-shelf hardware and 3D-printed components, with custom body and bogie PCBs that can be released and made openly available for reproduction. It features a modular software framework and robust perception capabilities that support autonomous functionality. The rover's architecture includes a 6-wheel triple-rocker-bogie suspension system, a stereo RGB-D vision camera with an integrated IMU, and a 6-DOF robotic arm with a gripper. Middleware frameworks such as ROS, Simulink, and RAFCON are used to enable teleoperation, autonomous navigation, arm manipulation, and frontier-based exploration. Moreover, the compact rover measures 36x26x39 cm and has a total mass of 3.7 kg, while achieving a maximum speed of 0.13 m/s. The LRM has been successfully field-tested at different summer schools, demonstrating autonomous 3D mapping, frontier-based exploration, and robotic arm grasping while monitoring electric current.**

Ongoing work includes the integration of a YOLOv7-based vision pipeline for object detection in mapped environment, integration of IMU data into the transformation tree for improved SLAM accuracy, and further mechanical enhancements. Also, increasing the complexity of autonomous exploration strategies through an utility function. The LRM seeks to bridge the accessibility gap in space robotics by offering a cost-effective, modular, easy-to-assemble rover that features capabilities comparable to those of state-of-the-art planetary rovers.

***Index Terms*— Mobile space robotics, robotic arm manipulator, space rover, autonomous navigation, frontier-based exploration, object detection**

I. Introduction

The development of mobile robotic platforms revolutionized space exploration through their ability to collect data on the surface of planetary bodies. While the Apollo Program achieved 11 crewed missions, including 6 successful lunar landings between 1969 and 1972, carrying out a program of scientific exploration of the Moon [1], such missions are resource-intensive, limited to nearby celestial bodies, and pose significant risks to human life. Furthermore, even though advancements have been made in rover teleoperations since Lunokhod 1 and Lunokhod 2 successfully landed on the Moon in the 70's [2], communication delays increase on more distant celestial bodies, rendering teleoperation from Earth impractical. For instance, Mars exploration missions, starting with the Mars Pathfinder, which deployed the Sojourner rover on Mars in 1997 [3], are either controlled on an asynchronous send commands and execute tasks basis, or rely on their ever-increasing onboard autonomy. Thus, it is clear that space rovers capable of exploring terrain, conducting experiments, and retrieving samples autonomously, stand at the core of next-generation deep-space missions. Recent missions, such as NASA's Perseverance rover [4], which landed on Mars in 2021, and ESA's ExoMars rover [5], targeted to launch in 2028, both feature substantial autonomous capabilities. In addition, the MMX rover [6], [7], developed by DLR together with CNES, scheduled to launch in 2026, will set down on Phobos as part of JAXA's MMX mission [8].

Fig. 1. Picture of the Lunar Rover Mini 1, on the right with the robotic arm, and the Lunar Rover Mini 2, on the left.

However, these advanced capabilities come with significant drawbacks for STEM researchers and students, such as costly hardware, closed-source non-modular software, and complexity of manufacturing and testing, making them unsuitable as testing platforms for experimental purposes. This being the case, building on top of this knowledge, there has been a growing interest in developing more accessible, open-source alternatives using off-the-shelf components that incorporate the same capabilities found in such systems. Taking inspiration from the ExoMars rover locomotion system, and using some of the same software packages as DLR's Lightweight Rover Unit (LRU), designed for search and exploration activities with a high-level of autonomy [9], the Lunar Rover Mini (LRM), Fig. 1, bridges this gap by presenting itself as a cost-effective, modular, easy-to-

[1]Tomás P. (tomas.placidodecastro@dlr.de), Jui-Wen Y. (jui-wen.yeh@dlr.de), Ivan M. (ivan.martinenciso@dlr.de), Hugo A. (hugo.allard@dlr.de), Armin W. (Armin.Wedler@dlr.de), Josef R. (Josef.Reill@dlr.de) and Riccardo G. (riccardo.giubilato@dlr.de) are with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Oberpfaffenhofen, Germany.

[2]Tomás P. (tomas.placido.castro@gmail.com) is also with Delft University of Technology (TU Delft), Delft, Netherlands.

TABLE I
COMPARISON OF LOW-COST COTS BUILT OPEN-SOURCE ROVERS

Rover	Dimensions	Mass	Embedded Capabilities	Max. Velocity	Autonomy*	Price**	Software Stack
ASURO	45x122x117 mm	0.165 kg	Basic object detection and avoidance	-	-	50 €	C
OSR	610x305x356 mm	12.7 kg	LED head information display	1.75 m/s	5 h	2160 €	Linux, Python
Sawppy	685x320x306 mm	-	Simple camera support	-	-	430 €	ROS, C, C++
ExoMy	300x390x420 mm	2.5 kg	2D RGB visual imaging	0.23 m/s	3 h	360 €	ROS, Python
LRM	360x260x390 mm	3.7 kg	RGB-D vision, autonomous nav. & exploration, 6-DOF robotic arm, object detection	0.13 m/s	3 h	2000 €	ROS, Python, C++, Linux

* Autonomy may vary depending on system configuration and use case.
** Prices are approximate and may vary due to market fluctuations.

assemble rover, which shares the same software components with other DLR robotic systems. Moreover, the LRM benefits from an open-source middleware framework.

The LRM, with dimensions of 36x26x39 cm, a mass of 3.7 kg, and a passive 6-wheel, triple-rocker-bogie suspension system, features 3D-printed body parts, a custom body and bogie PCBs, and a stereo vision RGB-D camera with an integrated IMU, mounted on a servo-controlled pan-tilt unit. It also features a 6-DOF robotic arm using off-the-shelf radio-control servos. Previous contributions to the project focused on improving the autonomous capabilities of the rover, by integrating the LRM with an autonomous navigation algorithm and corresponding data pipeline [10]. Also, in the development of a visual-inertial SLAM algorithm that combines visual and inertial measurements, from the stereo vision camera and the onboard IMU, respectively, to solve the SLAM problem through performing tightly coupled sensor fusion [11]. More recently, the accuracy of the 6-DOF robotic arm was improved through an elasto-kinematic calibration [12]. Ongoing work concerns the implementation of autonomous exploration strategies with object detection to map the environment and locate objects of interest, and a vision-based pipeline for autonomous grasping.

Moreover, several capabilities of the rover were field-tested during the PETRAS Summer School 2024 on the Italian island of Vulcano, north of Sicily, and at the iFOODis Summer School 2025 at the Kristineberg Center in Gothenburg, Sweden[1]. The PETRAS Summer School 2024 focused on validating the kinematic performance of the robotic arm, the behavior of the autonomous navigation stack, and investigating the thermal control of components. The integration of RAFCON [13] within the LRM for mission control, alongside the state machines for autonomous exploration, as well as monitoring of electric current for both autonomous grasping and temperature overheat estimation of the robotic arm, were addressed during the iFOODis Summer School 2025.

The paper is structured as follows. Section II provides a survey of related robotic platforms. In Section III, the system design overview is presented, including the system architecture, onboard electronics and communication, and actuator behavior for different driving modes. Section IV discusses the software architecture, including the main software development pipeline and middleware process management, perception capabilities, autonomous exploration, and image detection. Section V and Section VI describes different rover operations scenarios, including a high and low-level GUI and RAFCON, and field test work carried out on the rover, respectively. At last, Section VII concludes the paper by presenting a brief summary of the LRM, highlighting the major contributions to the project alongside future research directions and work.

[1]Details for the PETRAS Summer School 2024 and the iFOODis Summer School 2025 are provided, respectively.

II. RELATED WORK

Modular, built with off-the-shelf components, robotic platforms tailored to research and educational purposes are increasingly being adopted due to their ease of prototyping, assembly, affordability, and compatibility with widely used software, when compared to others, built specifically for the requirements of a particular mission. Hence, it comes with no surprise that organizations have dedicate resources to their development.

DLR took the first steps in this direction by launching ASURO[2] as early as 2003. This 2-wheel differential drive small mobile robot, sold over 30,000 units and at just a price of 50 €, is equipped with six collision probes and an optical unit, including two light sensors for line tracking and 2 light barriers for odometry, making it possible to recognize and avoid obstacles. A more recent example is NASA's JPL Open Source Rover[3] (OSR), launched in 2019. With a total cost of 2160 €, this remote-controlled rover features the flagship 6-wheel rocker-bogie suspension system used in NASA's space rovers. It achieves a top speed of 1.75 m/s and is designed to accommodate the integration of a LED head display, with at least 5 hours of autonomy. The OSR is powered by a Raspberry Pi and runs a software stack based on Linux, ROS, and Python, and benefits from a modular design framework that easily allow the introduction of hardware upgrades, such as a vision camera for collision avoidance, an IMU for pose estimation and navigation, or an arm manipulator for sample retrieval. Following this trend, the Sawppy rover kept the same design approach, mimicking the proportions and overall design of the Curiosity and Perseverance rovers. However, it reduces overall costs to 430 € by using a smaller frame, a mix of 3D-printed and aluminum components, and hobby servo motors instead of dedicated DC motors. Initially designed with a proprietary control stack, recent updates have made it ROS-compatible.

[2]Details about DLR's ASURO.
[3]Details about NASA's JPL OSR.

TABLE II
HARDWARE COMPONENTS USED ON THE LRM AND THEIR OPEN-SOURCE AVAILABILITY [14]

Subsystem	Hardware Component	Availability
On-board processing	Intel NUC (V7) [Info]	✓
Perception	Intel RealSense Depth Camera D435i [Info]	✓
Ground station communication	airMAX Bullet AC [Info]	✓
I/O and power management	DLR custom body PCB	✗*
Steer and drive control	DLR custom bogie PCB	✗*
Wheel actuators	Faulhaber 2619 006 SR [Info]	✓
Pan-tilt mechanism	AGFRC A50BHL Ultra Torque Programmable Brushless Servo [Info]	✓
Arm manipulator	AGFRC A80BHMF[†]/A35BHL[‡]/A20CLS[§] Programmable Brushless Servo [[†]Info] [[‡]Info] [[§]Info]	✓
Teleoperation controller	Logitech F710 Wireless Gamepad [Info]	✓

* Not available yet, but PCB design can be open-sourced for manufacturing.

Later, in November 2020, ESA developed the ExoMy rover [15]. The ExoMy rover features a 6-wheel triple-bogie steering system with a top speed of 0.23 m/s. It is also powered by a Raspberry Pi, but it is further equipped with a Raspberry Pi Camera. Its fully 3D-printed frame design brings costs down to 360 €. Like the OSR, it uses Python within a ROS framework. DLR's LRM contributes to this ecosystem of rovers by integrating a more powerful Intel NUC in place of a Raspberry Pi. It is also equipped with an Intel RealSense Depth Camera D435i and a 6-DOF robotic arm with a gripper. These features enable advanced autonomous navigation and arm manipulation capabilities out of the box, unlike the others rovers, which require additional, costly upgrade packages to achieve similar functionality, while maintaining a relative affordable price point of 2000 €, under an open-source software development stack. Also, note that the bulk of this price point could be substantially reduced if the custom PCBs were mass-produced.

Table I summarizes the available key features of the aforementioned rovers.

III. SYSTEM DESIGN

A. System Architecture Overview

The LRM consists of a main body frame that houses an on-board computer, middleboard PCB, and a wireless communication antenna. The head section features a pan-tilt unit actuated by two servos through PWM signals, which control the stereo vision camera. The locomotion system is a 6-wheel, 3-bogie suspension which guarantees continuous ground contact for all six wheels while traversing uneven terrain. Each bogie integrates two custom bogie PCBs and two servos for independent wheel steering. Each wheel is individually driven by an internal DC motor. Thus, in total, the system includes 12 DC motors for wheel movement. Mounted on the body is a 6-DOF robotic arm comprising six actuated joints, along with a gripper driven by an additional servo as its end-effector. The battery fitting is positioned underneath the body frame to power all onboard components. An overview of the main components of the rover is shown in Figure 2, and the key hardware modules used in the system are summarized in Table II.

The entire rover casing is fully 3D-printed using ASA (Acrylonitrile Styrene Acrylate) material, which contributes to its lightweight structure and can be easily reproduced at home using standard hobby-style 3D printers. With compact dimensions of 360x260x390 mm, the LRM is easy to transport and well-suited for field testing. An offline computer is used as ground station by remotely connecting to the on-board computer for controlling and monitoring processes.

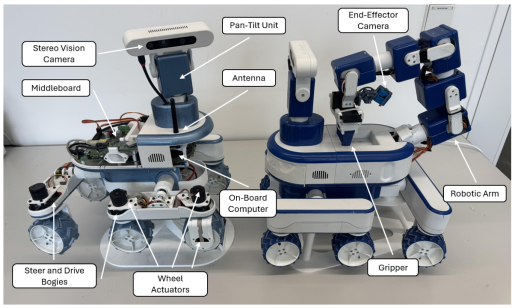


Fig. 2. Overview of main hardware components of the LRM, highlighting the structural design and the mechanical and electronic subsystems.

The wheels, specifically designed for sandy environments with small obstacles, have a diameter of 6.5 cm with periodic 3 mm protuberances. This design improves traction and grip during locomotion. The small wheel diameter keeps the rover compact and easy to transport. However, the reduced wheel size limits the rover's ability to traverse larger obstacles, thereby decreasing ground clearance compared to high-end planetary rovers. Therefore, future work will be dedicated to explore alternative wheel concepts to improve terrain adaptability.

The robotic arm is positioned opposite the pan-tilt unit to ensure better weight distribution across the rover's casing and to make sure the robotic arm's home position does not interfere with the stereo vision camera FOV. Since the pan-tilt unit can rotate, the camera can still be used to monitor object grasping by the robotic arm in this configuration. The size of the robotic arm is constrained by the dimensions of the joint RC servos, leaving minimal clearance for the joints to rotate.

B. Onboard Electronics and Communication

Figure 3 illustrates the system's onboard electronics architecture, showing how the on-board computer connects with other electronics. The Intel NUC is running an Open-Suse

Leap 15.4 OS. On top of that, USB connections connect the Intel NUC to both the pan-tilt unit (via USB-C) and the body PCB (via USB Mini-B). Additionally, the middleboard communicates with the six bogie PCBs using RS-485 as the differential serial communication standard, while the data itself is formatted and handled using the UART protocol.

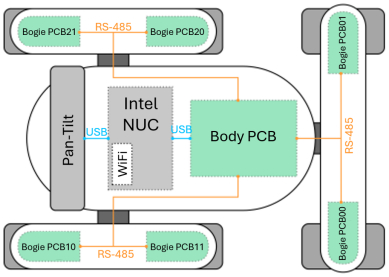


Fig. 3. Schematic of the onboard electronics and communication between modules.

Moreover, the body PCB manages the communication between all actuators and the on-board computer. The entire system is powered by a 14.9 V LiPo battery.

The on-board computer is responsible for processing and managing all data, including autonomous navigation tasks, execution of state machines, input controls, joint states for the robotic arm and communication with the microcontroller that interfaces with the hardware components. The on-board computer acquires IMU sensor data from both the Intel RealSense Depth Camera D435i and the microcontroller PCB. The communication frequency between the on-board system and the microcontroller has been increased from 10 Hz to 100 Hz to improve data acquisition and sensor sampling rates, thereby enhancing overall rover performance.

C. Driving and Steering Mechanism

There are three different driving modes defined by a combination of zero and non-zero linear velocities in the rover's plane, x and y direction, and angular velocity, along the z-axis, which corresponds to a rotation about the rover's vertical axis, Table III. These modes are *Ackermann Mode* (car-like turning with sharper angles on the inside), *Rotation Mode* (in place rotation), and *Crabwalk Mode* (sideways motion with all wheels aligned in the same direction) [10].

TABLE III
ROVER DRIVING MODES BASED ON LINEAR AND ANGULAR VELOCITY INPUTS

Driving Mode	Linear x	Linear y	Angular z
Ackermann	0	0	0
Rotation	0	0	$\neq 0$
Crabwalk	0	$\neq 0$	0
Ackermann	$\neq 0$	0	0
Ackermann	$\neq 0$	0	$\neq 0$
Crabwalk	$\neq 0$	$\neq 0$	0

* Undefined combinations are not showed as they result in no movement.

The maximum linear velocity achieved by the rover is 0.13 m/s, while the maximum angular velocity in rotation mode is 12.8 °/s. Nevertheless, it is important to note that, regardless of the control mode, them being the *high-level GUI*, the *gamepad*, or the *autonomous navigation stack*, a conversion always takes place to translate the control inputs into a universal steering angle and driving velocity command output to the wheels.

IV. Software Architecture

A. Software Pipeline and Middleware Process Management

As previously mentioned, the high-level software infrastructure of DLR's LRU is also employed in the LRM. This allows to reuse software packages from different robotic platforms from the Robotics and Mechatronics (RM) institute of DLR. The open-source middleware framework comprises the real-time middleware process manager tool Links-and-Nodes [16], ROS for all autonomous perception and navigation-related tasks, Simulink to determine the commands sent to the rover's steering, driving and pan-tilt motors from control inputs, and RAFCON to program and manage the execution of autonomous tasks through hierarchical state machines [13], all running at the same time on the on-board computer. The software packages are deployed through the Links-and-Nodes Manager.

Besides, a CI/CD pipeline supports the software development, by simplifying the build, test, and deployment processes. This pipeline leverages open-source tools, such as Conan for package management, Jenkins for automated builds, and Artifactory for storing and distributing software dependencies[4]. Similarly to Table II, Table IV provides an overview of the software and middleware modules as well as the CI/CD pipeline used on the system.

B. Perception

Visual odometry (VO), a *Visual SLAM pipeline* (including *local mapping*), and a *navigation stack* are widely recognized as the three foundational capabilities for the implementation of an autonomous exploration system.

1) Visual Odometry: In the LRM case, a feature-based VO approach is used with the stereo vision camera. It detects key points using the Good Features to Track (GFTT) algorithm and describes them with BRIEF descriptors to match features across frames within a Frame-to-Frame strategy. The VO pipeline runs within the ROS `rtabmap_odom`[5] node.

2) Visual SLAM Pipeline: Furthermore, a visual SLAM pipeline is used to map the environment, employing the open-source RTAB-Map library [17] with the C++ GTSAM library [18] as the low-level optimizer. The RTAB-Map library is responsible for sensor data integration, loop closure detection, and overall maintaining a globally consistent map, while GTSAM performs efficient nonlinear optimization on the factor graph representing the robot poses and landmark constraints. This allows to generate a real-time 3D point

[4]More information about these open-source tools is available at: Conan, Jenkins, and Artifactory.

[5]Available: `https://wiki.ros.org/rtabmap_odom`

TABLE IV

SOFTWARE, MIDDLEWARE AND CI/CD MODULES USED ON THE LRM AND THEIR OPEN-SOURCE AVAILABILITY [14]

Module	Specifications	Availability
Software		
Visual odometry	Frame-to-Frame Strategy from RTAB-Map VO [Info]	✓
Visual SLAM	RTAB-Map [Info]	✓
Navigation stack	DLR custom navigation stack	✗*
I/O rover actuator controller	Simulink[Info]	✓
Middleware		
Process manager	Links-and-Nodes[Info]	✓
Perception and navigation tasks	ROS [Info]	✓
Manage and execution of autonomous tasks	RAFCON [Info]	✓
CI/CD Pipeline		
Package management	Conan [Info]	✓
Software build automation	Jenkins [Info]	✓
Store and distribute software artifacts	Artifactory [Info]	✓

* Not open-sourced but can be replaced by another navigation package compatible with ROS.

cloud map of the environment, and respective occupancy grid, which is shared with the navigation stack. The SLAM pipeline runs within the ROS `rtabmap_slam`[6] node.

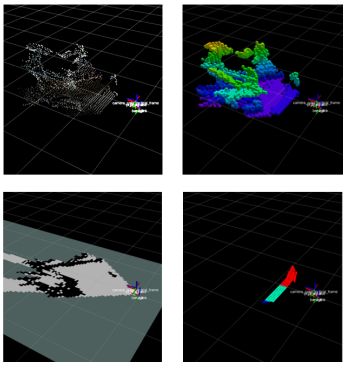


Fig. 4. Example of the generated 3D point cloud, on the top left, and respective 3D OctoMap, on the top right, occupancy grid, on the bottom left, and local mapping with obstacle detection, on the bottom right.

3) Navigation Stack: The navigation stack developed in-house by DLR includes a path planning algorithm with obstacle avoidance. More specifically, the path planning algorithm of the local planner uses the classic A* search method to compute a short, collision-free path in a local costmap derived from elevation maps and computed by a local mapping node. A motion client, part of the same navigation stack, is responsible for publishing the velocity commands and deciding which driving mode to use accordingly.

Figure 4 illustrates the result of the perception capabilities visualized in RViz. With respect to the occupancy grid, black grid cells (■) represent obstacles, while grey (■) and green (■) represent free and unknown space, respectively. The gradient of the local mapping depicts sloped areas, which can be classified as obstacles.

[6]Available: `https://wiki.ros.org/rtabmap_slam`

C. Autonomous Exploration

Autonomous frontier-based exploration strategies have been implemented in the LRM within a RAFCON-based state machine framework to generate 3D maps of unknown environments, using a modular and scalable approach that enables future expandability. The core of using a frontier-based approach for autonomous exploration relies on knowing how to use the stereo vision camera image, Fig 5a, to classify candidate frontiers from the generated 3D point cloud, Fig 5b, and respective 2D occupancy grid Fig 5c, from the visual SLAM pipeline. Figure 4 depicts such occupancy grid on the bottom left. Candidate frontiers, Fig 5d, are defined as free grid cells adjacent to unknown ones. Candidate frontiers are then segmented into clusters, Fig 5e, constrained by minimum and maximum sizes. For instance, a frontier cluster consisting of only a 5x5 cm grid cell may be considered too small to explore. Next, the centroid coordinates of each cluster are calculated, Fig 5f, along with the corresponding edges, Fig 5g.

Given the shape of the FOV of the stereo vision camera, the 2D occupancy grid outer boundary will be convex from the point of view of the camera, placing the centroid between the frontier and the rover, instead of behind the frontier. There is also a position and orientation watcher in place, responsible to stop the forward movement of the rover towards the frontier when a certain threshold distance is reached. Additionally, two criteria are evaluated to classify each centroid as a valid one to explore. First, if a small cluster of unknown cells is completely surrounded by free cells, it is reclassified as free space and thus not marked for exploration. Second, if there is no clear free path between the centroid and the rover, the centroid is excluded from exploration.

Based on the selected exploration strategy, a path through waypoints is then generated to guide the rover toward one or more of these centroids, Fig 5h, fully autonomously. The number of waypoints to explore determines if this path generation is done by brute force or by using a nearest-neighbor heuristic for faster planning. The state machines responsible for navigation and monitoring direct the rover to move toward the selected centroids relying on the capabilities of the local mapping and navigation stack mentioned in

Subsection IV-B. Once the rover reaches a position and orientation within defined thresholds, it stops its linear motion and performs a rotation within the range of edges of the frontier for maximum frontier coverage, fully autonomously. This process can be repeated until a certain area is mapped or there are no more frontiers to explore.

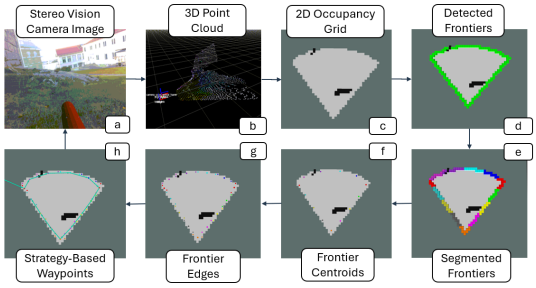


Fig. 5. Breadth-First Search exploration strategy, from (d) through (h), including the stereo vision camera images (a), and generated 3D point cloud (b), used to build the 2D occupancy grid (c).

So far, two different high-level exploration strategies have been implemented: *Breadth-First Search* and *Greedy Frontier Selection*. In the first one, the rover explores all waypoints of one node of the occupancy grid, before looking at the newly updated occupancy grid for the next node of waypoints, while the second one moves the rover to one frontier selected from all detected frontiers based on certain criteria, such as the frontier's size or distance from the rover. Figure 5 showcases the entire loop of the previously explained autonomous exploration pipeline algorithm for the Breadth-First Search strategy, as an example.

In addition, a probabilistic OctoMap, a 3D map representation based on an octree data structure which divides 3D space into voxels (smaller cubes), each storing a probability of occupancy [19], Figure 4, has also been implemented ahead of the current work to expand the exploration strategies towards a *Utility-Based Goal Selection* using an utility function that balances traveling cost and expected information gain.

D. Object Detection

To further advance the autonomous exploration capabilities of the rover, a real-time YOLOv7 object detector [20] has been trained on synthetic data for the detection of white cubes. The trained YOLOv7 model was then deployed on the rover's Intel NUC using the OpenVINO [7] (Open Visual Inference and Neural Network Optimization) toolkit, which first takes the trained model from a TensorFlow framework, and converted it to an intermediate format for accelerated hardware-specific inference by leveraging the NUC's CPU and integrated GPU, allowing for real-time on-board object detection at a given rate, during exploration. Figure 6 depicts an example of the trained YOLOv7 model output. Ongoing work focuses on integrating this object detector so that the rover is capable of determining the coordinates of such objects in the global coordinate frame, while mapping the 3D environment. This capability will be integrated with the ongoing work on the arm manipulator, Subsection VI-B, to enable fully autonomous exploration and sample retrieval.

[7]Available: https://docs.openvino.ai/2023.3/notebooks/226-yolov7-optimization-with-output.html

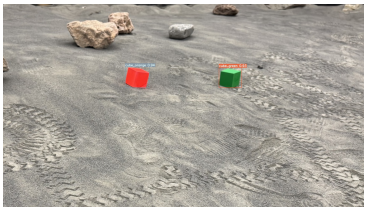


Fig. 6. Trained YOLOv7 object detector output, running on the rover's on-board computer, in the PEL laboratory, on colored cubes near rocks.

Using RAFCON state machines, the rover will perform 3D environmental mapping, accurately identify objects of interest along with their coordinates in the global map frame, and relay this information to the arm manipulator, which will then autonomously execute object grasping using a visual pipeline based on a low-cost camera mounted on the end-effector.

V. Rover Operations

The LRM features a *low-level GUI* and a *high-level GUI*. The high-level GUI is deployed through the Links-and-Nodes manager, just like all other rover processes, while the low-level GUI runs as a standalone application.

A. High and Low-Level GUI

The high-level GUI allows to steer and drive the LRM, rotate the pan-tilt unit, move the robotic arm to pre-defined positions, or specify the target position for the end-effector. Also, it enables switching between different driving modes, including the high-level GUI, a gamepad controller, and the autonomous navigation stack. The target position goals from the autonomous navigation can come either from a 2D position goal from RViz, or directly by setting a local goal from RAFCON state machines. Likewise, the low-level GUI is used for calibrating the wheels and tuning the PID control values for both steering and driving, including both the inner and outer control loops. It also includes live data from the IMU, electric current of the whole robotic arm and temperature of the body PCB. Finally, support for single-joint control and functionalities for saving, deleting, and managing predefined positions are also implemented.

B. RAFCON

What is more, RAFCON integrates the capabilities of the high-level GUI into an autonomous task framework. For example, while the high-level GUI can manually rotate the pan-tilt unit, a dedicated state machine was created for the same capability, taking a specific rotation value as input. This state machine can then be embedded in a hierarchical state

machine to perform automatic rotations as part of a sequence of autonomous tasks. In this way, the same functionalities offered by the high-level GUI are now within an autonomous execution framework. The designed state machines can also switch between driving modes with different linear and rotation velocities.

Figure 7 depicts the on-board computer interface during normal rover operations running the Links-and-Nodes manager, RViz for the 3D environment map and rover pose, and RAFCON for managing the execution of state machines.

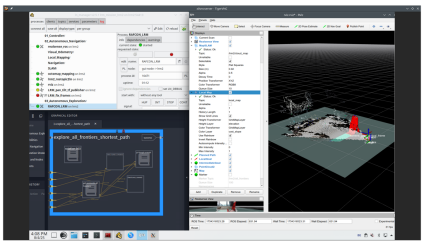


Fig. 7. On-board computer interface during typical rover operation, showing the Links-and-Nodes manager, on the top left, RAFCON for autonomous task execution, on the bottom left, and RViz for 3D environment mapping and navigation, on the right.

VI. Field Test Work

A. Breadth-First Search Autonomous Exploration

The implementation of the RAFCON state machines necessary for Breadth-First Search autonomous exploration was tested at the iFOODis Summer School 2025 to map a sand pit, Figure 8. The rover started by performing a 360-degree rotation so it could detect the most frontiers from the beginning. A total of 52 state machines were tested, comprising 13 hierarchical states, 12 for computations, 9 for triggering actions, 6 for reading data, 7 for action monitoring, and 5 responsible for interfacing with other software on the rover. The resulting map covered an area of approximately 23 m^2.

The results validated both the implemented state machines and the accuracy of the rover's perception capabilities to generate a detailed 3D map of the environment. Notably, the generated map captures the presence of the other LRM positioned at the sand pit. It is important to note that the maximum range of the stereo vision camera was purposefully limited to 1 meter in the VO and SLAM pipeline, requiring the rover to explore more frontiers to map the same area. This was introduced to bolster the validation of the implemented exploration strategy.

Further, this experiment allow to conclude about future improvements on the implemented exploration strategy. First, more efficient usage of the pan-tilt mechanism to replace frequent body commands. In addition, implementing the rover's URDF in RViz would improve visualization by being able to better judge the rover's dimensions in the 3D environment, during mapping.

Fig. 8. 3D map of a sand pit of approximately 23 m^2, generated through Breadth-First Search autonomous exploration, validating the correct implementation of RAFCON state machines, perception capabilities, and autonomous task execution.

B. Arm Manipulator

As previously mentioned, the robotic arm is already fully implemented and has been extensively tested during both summer schools. The 6-DOF robotic arm of the LRM is equipped with consumer-grade servo motors that lack position feedback, which limits the arm's control capabilities and prevents the system from detecting whether an object has been successfully grasped by the gripper.

Experimental research was conducted to address these limitations. The LRM middleboard is equipped with an electrical current sensor, which was used to control the grasping process and detect the presence of grasped objects. The underlying hypothesis was that when all nine servos are in a static position, the electric current draw is constant, allowing the separation of current specific to the gripper servo during object grasping. Results demonstrated that grasping operations could be detected using a single current sensor monitoring all nine servos simultaneously, with grasping operations increasing current by 0.7 A, above the measured static arm position. This finding enabled successful automatic grasping of rocks weighing up to 80 grams without requiring additional feedback from the servos.

Future work will focus on developing a complete autonomous grasping pipeline, using a low-cost camera mounted on the end-effector to provide visual feedback for a YOLO-based object detector.

C. Mechanical Improvement

Although the rover's mechanical design is already robust, several issues were identified during field testing concerning the gripper. The current gripper design uses a servo motor to control the gear rack system to close the gripper. However, the current design has two main issues: the rack is not long enough, and the rack was not properly restrained, which leads to instability when the gripper is moving. Temporary fixes included using straps to restrain the rack and adding a 3D-printed extension.

Future improvements will maintain the current gripper concept but incorporate updated components to simplify manufacturing and testing. A modular camera mount for the robotic arm is also planned to be designed. Moreover, some components showed fragility under harsh conditions during the field testing and shall be redesigned.

VII. Conclusion

This paper presented the Lunar Rover Mini, a project created in 2015 by DLR to develop and open-source, low-cost robotic platform for research and educational purposes, which has since involved contributions from over 20 students. The rover is able to autonomously navigate the environment while avoiding obstacles thanks to its stereo vision camera. Furthermore, its perception capabilities allowed the implementation of autonomous exploration strategies, validated during the iFOODis Summer School 2025. The integration of the already trained YOLOv7 object detector will follow to determine the coordinates of objects of interest in the global 3D map frame. The software development of rover is done through an internal CI/CD pipeline, and all real-time middleware process tools are open-source. The robotic arm, has fully functional kinematics (both forward and inverse). Autonomous grasping with the aid of electric current sensors has been tested and shows promising results for further tuning, as well as temperature prediction of the robotic arm servo motors using the same current sensor for safe operation.

Future work includes the deployement of the trained YOLOv7 object detector within the RAFCON autonomous exploration pipeline, and the implementation of an autonomous vision-based grasping pipeline to enable end-to-end mission cycle autonomy from exploration to sample retrieval. All things considered, the LRM poses as a versatile and accessible robotic platform that incorporates advanced functionalities of high-end rovers, while being build primarily on off-the-shelf hardware components.

ACKNOWLEDGMENTS

We thank the colleagues from the Institute of Robotics and Mechatronics, that have taken their time to support this project. This work was made possible through the Helmholtz Association, project alliance ROBEX (contract no. HA-304), project ARCHES (contract no. ZT-0033), and project iFOODis (contract no. KA-HSC-06 iFOODis).

References

[1] NASA. The apollo program. Accessed: 16.07.2025. [Online]. Available: https://www.nasa.gov/the-apollo-program/

[2] L. A. Vedeshin, "The first soviet experiments on remote sensing and contact study of the moon (on the 50th anniversary of the moon landing of lunokhod 1 self-propelled vehicle)," *Izvestiya, Atmospheric and Oceanic Physics*, vol. 57, no. 12, p. 1800–1802, 2021.

[3] R. A. Cook and A. J. Spear, "Back to mars: The mars pathfinder mission," *Acta Astronautica*, vol. 41, no. 4, p. 599–608, 1997, developing Business. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576598000691

[4] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. K. Herd, R. Hueso, Y. Liu, J. N. Maki, G. Martinez, R. C. Moeller, A. Nelessen, C. E. Newman, D. Nunes, A. Ponce, N. Spanovich, P. A. Willis, L. W. Beegle, J. F. Bell, A. J. Brown, S.-E. Hamran, J. A. Hurowitz, S. Maurice, D. A. Paige, J. A. Rodriguez-Manfredi, M. Schulte, and R. C. Wiens, "Mars 2020 mission overview," *Space Science Reviews*, vol. 216, no. 8, p. 142, 2020.

[5] M. Winnendael, P. Baglioni, and J. Vago, "Development of the esa exomars rover," 08 2005.

[6] S. Ulamec, P. Michel, M. Grott, U. Böttger, S. Schröder, H.-W. Hübers, Y. Cho, F. Rull, N. Murdoch, P. Vernazza, O. Prieto-Ballesteros, J. Biele, S. Tardivel, D. Arrat, T. Hagelschuer, J. Knollenberg, D. Vivet, C. Sunday, L. Jorda, O. Groussin, C. Robin, and H. Miyamoto, "Science objectives of the mmx rover," *Acta Astronautica*, vol. 210, p. 95–101, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576523002448

[7] M. Vayugundla, T. Bodenmüller, L. Burkhard, M. J. Schuster, B.-M. Steinmetz, M. Sewtz, N. Borgsmüller, F. Buse, W. Stürzl, R. Giubilato *et al.*, "The dlr autonomous navigation experiment with the idefix rover: Software architecture, autonomous navigation features and preliminary operations concept," in *2025 IEEE Aerospace Conference*. IEEE, 2025, pp. 1–20.

[8] Y. Kawakatsu, K. Kuramoto, T. Usui, H. Sugahara, H. Ootake, R. Yasumitsu, K. Yoshikawa, S. Mary, M. Grebenstein, H. Sawada, T. Imada, T. Shimada, K. Ogawa, M. Otsuki, M. Baba, K. Fujita, K. Zacny, D. van Dyne, Y. Satoh, and A. Tokaji, "Preliminary design of martian moons exploration (mmx)," *Acta Astronautica*, vol. 202, p. 715–728, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009457652200474X

[9] M. J. Schuster, S. G. Brunner, K. Bussmann, S. Büttner, A. Dömel, M. Hellerer, H. Lehner, P. Lehner, O. Porges, J. Reill, S. Riedel, M. Vayugundla, B. Vodermayer, T. Bodenmüller, C. Brand, W. Friedl, I. Grixa, H. Hirschmüller, M. Kaßecker, Z.-C. Márton, C. Nissler, F. Ruess, M. Suppa, and A. Wedler, "Towards autonomous planetary exploration," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 3, p. 461–494, 2019.

[10] A. Ricardez Ortigosa, "Systems integration with autonomous navigation of the lunar rover mini for a space demo mission," Master's thesis, TU Berlin, 2023. [Online]. Available: https://elib.dlr.de/200804/

[11] Sam Bekkers, "Sensor Fusion for Visual-Inertial Simultaneous Localisation and Mapping," 2024. [Online]. Available: https://resolver.tudelft.nl/uuid:06bf2239-db11-41ee-bab2-a90e95b3ceb8

[12] M. Conenna, "Elasto-Kinematic Calibration of the Lunar Rover Mini 6DOF Robotic Arm," 2024. [Online]. Available: https://elib.dlr.de/212024/

[13] S. G. Brunner, F. Steinmetz, R. Belder, and A. Dömel, "Rafcon: A graphical tool for engineering complex, robotic tasks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3283–3290.

[14] S. Bekkers, M. T. Bettendorf, J. Reill, R. Giubilato, and A. Wedler, "The lunar rover mini: towards a versatile, open-source mobile robotic platform for educational and experimental purposes," in *17th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, October 2023. [Online]. Available: https://elib.dlr.de/202313/

[15] M. Voellmy and M. Ehrhardt, "Exomy: A low cost 3d printed rover," 10 2020.

[16] Florian Schmidt, "Links and Nodes Manager," 2020, accessed: 17.07.2025. [Online]. Available: https://gitlab.com/links_and_nodes/links_and_nodes

[17] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, p. 416–446, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1002/rob.21831

[18] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," *Georgia Institute of Technology, Tech. Rep*, vol. 2, no. 4, 2012.

[19] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013. [Online]. Available: https://doi.org/10.1007/s10514-012-9321-0

[20] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022. [Online]. Available: https://arxiv.org/abs/2207.02696