Final Year Project Report

**********

# Multi Agent Scene Exploration and Mapping for Continuous Digital Twin Creation

**********

BELPOIS Vincent
Under the supervision of Dr. Ivan MUTIS

2024



# Acknowledgments

Acknowledgments.

# Contents

# 1 Introduction

EMPTY

# 2 Setting up the physical agents

Three platforms were used in this project: a six-wheeled platform, a quadcopter, and a quadruped platform. Each platform was chosen for its specific characteristics, with the goal of having a multi-agent system that could explore and map a scene. The wheeled platform was chosen for its stability and ability to carry heavy loads. The quadcopter was chosen for its ability to fly and provide a bird's-eye view of the scene. The quadruped platform was chosen for its ability to climb stairs and navigate bumpy terrain.

## 2.1 Wheeled platform setup

The wheeled platform was chosen for its stability on flat terrain and with other projects in mind, its ability to carry heavy loads. The intent was to have a robotic arm mounted for other projects. Some previous work had already been done on the platform, but the person was apparently unsuccessful. These earlier attempts had left the platform in a partially modified state, requiring a comprehensive reassessment and redesign of both the mechanical and electrical systems. Despite these setbacks, the robust chassis of the six-wheeled platform still presented an excellent foundation for our project, offering the potential for further development.

### 2.1.1 Mechanical modifications

As it arrived, the six-wheeled platform only consisted of a stainless steel chassis, six DC motors, and wheels. The platform required several mechanical modifications to accommodate the necessary components for autonomous operation. Specifically, it needed a mount for a LIDAR sensor and an embedded computer on its top surface (see Figure **??**). The inside was reserved for the motors, battery, and power circuitry. Additionally, a mounting solution for the motor drivers inside the chassis was essential (see Figure 1), as the previous version didn't secure them in place and had several drivers stop functioning. These modifications were designed and implemented to ensure proper integration of all components while maintaining the structural integrity of the platform, the integration of the LIDAR and the embedded computer can be seen in Figure 2.

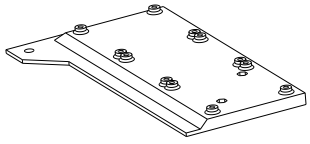


Figure 1: Drawing of CAD model of the motor driver mounting solution

To further accelerate development of the construction of the robots, a 3D printer was purchased and set up by myself. We chose the Bambu Lab P1P 3D printer for its large enclosed build volume, which would allow us to print engineering materials such as ABS, Nylon, or carbon fiber filed materials. It required a nozzle to be changed to hardened steel to be able to handle fiber filled filaments, which I performed and tested by printing protection cages for the LIDARs. The printer was set up in the lab and was used to print the parts needed for the mechanical modifications of the

platforms. It was also used to print parts for other projects in the lab, such as storage for VR headsets, and all kinds of brackets.

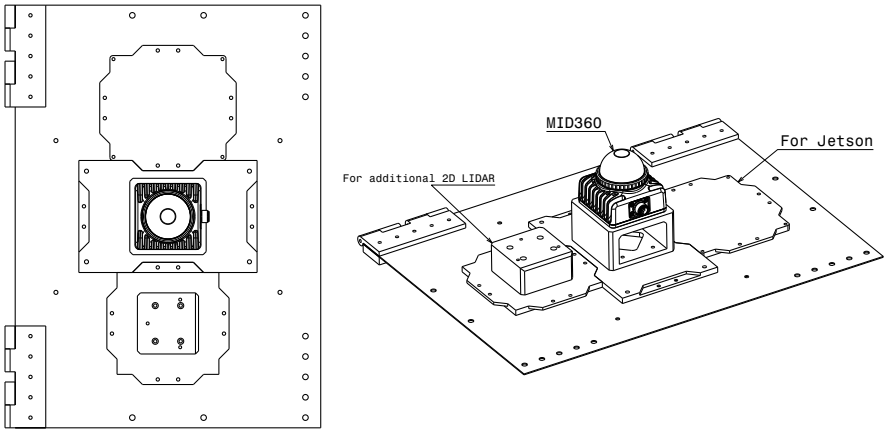



Figure 2: Drawing of CAD model of the modified six-wheeled platform's top

### 2.1.2 Electronics architecture

The essential electronic components needed to get the platform running were mainly DC motor drivers to drive the motors, a LIDAR sensor and an embedded computer.

Difficulties were encountered when trying to use the drivers someone else tried beforehand as they were under powered : at stall, the motors required around 5 amps of current, as measured with a bench top power supply, and the drivers I was trying to use were only capable of delivering 2 amps per channel or a total of 4 amps when combining outputs. The drivers in question were the based on the LN298N which were in terms replaced by the 7A dual motor drivers from DFRobot. A physical comparison can be seen in Figure 3.

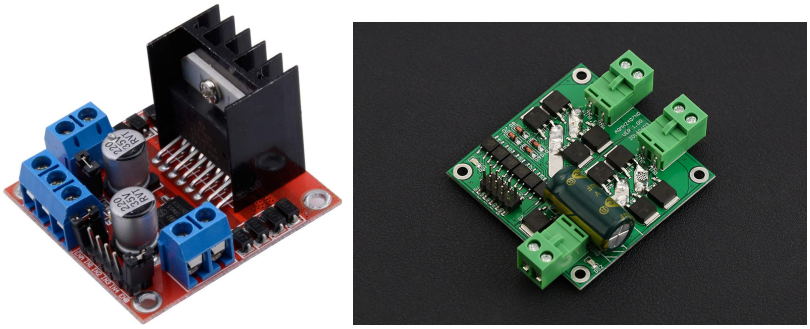


Figure 3: L298N (left) and DFRobot 7A dual DC Motor Driver (right)

The three motor drivers were connected to a microcontroller through custom made cables, as the cables previously used were undersized for the current of the motors. The connection can be seen in Figure 5, as well as the connection from the microcontroller to the radio receiver. I chose to use a Raspberry Pi Pico microcontroller for its many outputs, totaling 22 general purpose input outputs (GPIO).

Each driver required 6 control signals or 3 per motor: two signals are used to control the direction of the motor according to Table 1, while the third signal's duty cycle determines the speed.

A radio control (RC) receiver was also connected to interrupt capable GPIOs of the microcontroller to be able to control the platform manually. We use interrupts for those signals as to measure the pulse duration accurately and thus infer the control signal sent via radio. Three channels of the RC receiver were used to control the speed, the direction and the mode of the platform. The mode refers to whether or not the platform is in manual control or in autonomous mode and is connected to channel 5 of the radio which has a two-way switch.

Finally, the Pico is connected to an Nvidia Jetson Orin single board computer (SBC) via USB. This connection is used both to reprogram the Pico, and to send speed and direction commands to each motor via a serial communication.

We chose a Jetson Orin as it has a powerful GPU, which is not the case for many other SBCs like Raspberry Pis, and is designed for robotics applications in mind. Even though the algorithms we are using don't make use of the GPU and the CUDA cores, other applications like machine vision, or certain machine learning based terrain traversability methods could benefit from it [1]. Small form factor computers featuring x86 based processors were also considered and would have offered more CPU performance, but powering them and interfacing with them would have been more difficult as they are not designed for robotics.

To summarize the different connections of all the components, Figure 4 shows the electrical connections of these components on the modified six-wheeled platform, excluding power distribution and regulation.

| IN1 | IN2 | ENA/ENB | Motor1/2 Behavior |
|---|---|---|---|
| 0 | 0 | x | Stop (brake) |
| 1 | 1 | x | Vacant |
| 1 | 0 | 1 | Forward 100% |
| 0 | 1 | 1 | Reverse 100% |
| 1 | 0 | PWM | Forward at PWM speed |
| 0 | 1 | PWM | Reverse at PWM speed |

Table 1: Motor control signal table, where 1 is a high signal and 0 is a low signal

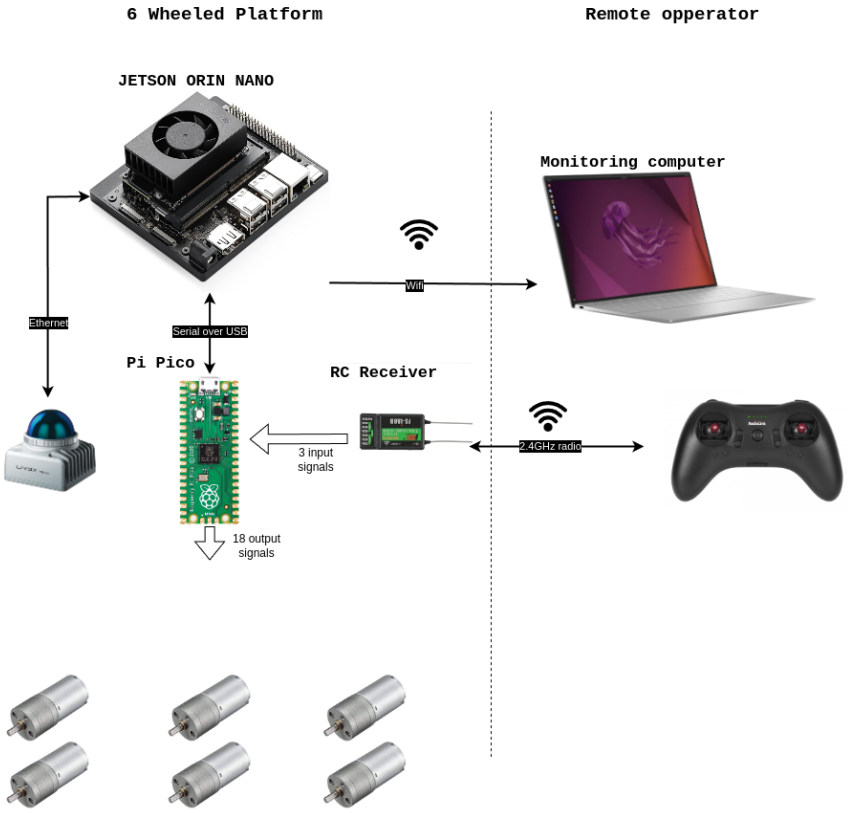



Figure 4: Overall electrical system, excluding power distribution and regulation

### 2.1.3 Software architecture

The robot operating system (ROS) was chosen as the software framework for the platform running on the Jetson Orin embedded computer. Specifically, ROS2 Humble Hawksbill was selected due to its extensive package availability and compatibility with the Jetson Orin's hardware. Indeed, I was made aware of the struggles of another researcher running a ROS based robot on an Nvidia Jetson Nano and how Ubuntu, and ROS version mismatch may bring problems.

This version of ROS2 provides a robust and flexible framework for developing and integrating various components of the platform, including sensor processing, navigation, and control. For real-time critical tasks such as motor control and RC radio interrupts, the Raspberry Pi Pico microcontroller was utilized, leveraging its ability to handle low-level, time-sensitive operations. The microcontroller's firmware was developed from the ground up by myself, guaranteeing reliable execution of motor control and interrupt handling tasks. By combining the strengths of ROS2 on the Jetson Orin with the real-time capabilities of the Raspberry Pi Pico, the platform achieves a robust and efficient software architecture that enables seamless integration of autonomous navigation, sensor processing, and manual control. This software architecture is illustrated in Figure 6.

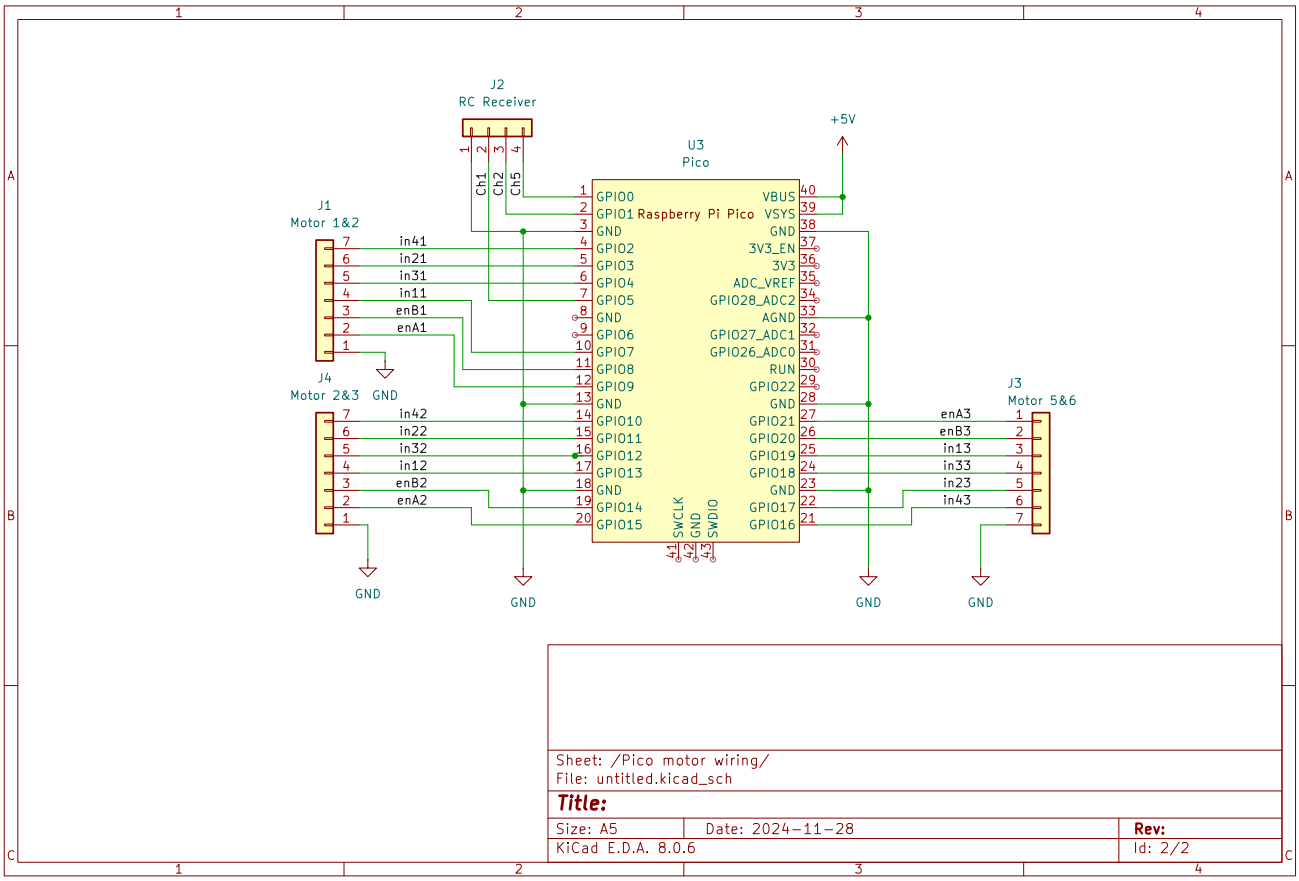



Figure 5: Wiring of the Raspberry Pi Pico to the motor drivers and to the RC receiver

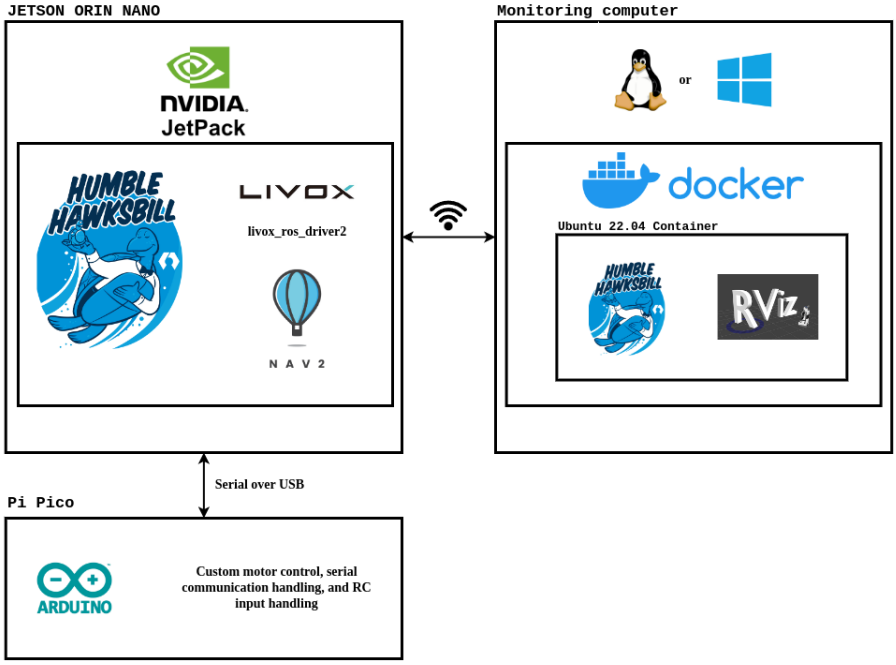



Figure 6: Software architecture

### 2.1.4 ROS 2 setup

As the distribution of Linux that we use on the Jetson is derived from Ubuntu 22.04, ROS2 can be installed using the Debian packages provided by the ROS2 foundation. The installation process is straightforward [22] and is well documented on the ROS2 website. ROS2 packages are often built for multiple architecture include x86, and ARM32. The Jetson Orin being an ARM64 architecture, the packages are directly compatible with it.

The specific distribution of Linux that we decided to install is Jetson Linux 36.4, also called Jetpack 6.1. This distribution is based on Ubuntu 22.04 and the Linux kernel 5.15. The main advantage of using this Linux distribution over any other is the included Nvidia drivers, and the included libraries like CUDA[21], TensorRT [20], cuDNN[7], VPI and many others. These libraries are essential for the use of the Jetson Orin as a computer vision and machine learning platform. The inference capabilities were not explored during this internship, but the embedded computer could be used for future sim to real [23] experiments.

In ROS2 a controller was created for handling the differential control of the robot. This packaged followed the interfaces provided by ros2_control, allowing the seamless control of the robot both in simulation and in real life. The package I created for this is called `skid_drive_controller` and is largely inspired by the differential controller provided by the ros2_control package.

As can be seen in Figure 7, taken from the ros2_control documentation, controllers request interfaces from the controller manager. The controller manager has previously received information from sensors, actuators, or systems that want to provide command or state interface. In our case, a modified version of the `diff_drive_arduino` takes care of providing one control interface and one state interface per wheel. Our `skid_drive_controller` then takes care of receiving data like wheel velocity from the state interfaces, and sends commands to the control interfaces. In the case of the Gazebo simulation, the simulator provides the state and command interfaces to the resource manager, which make our controller unaware of the fact that it is running in simulation.

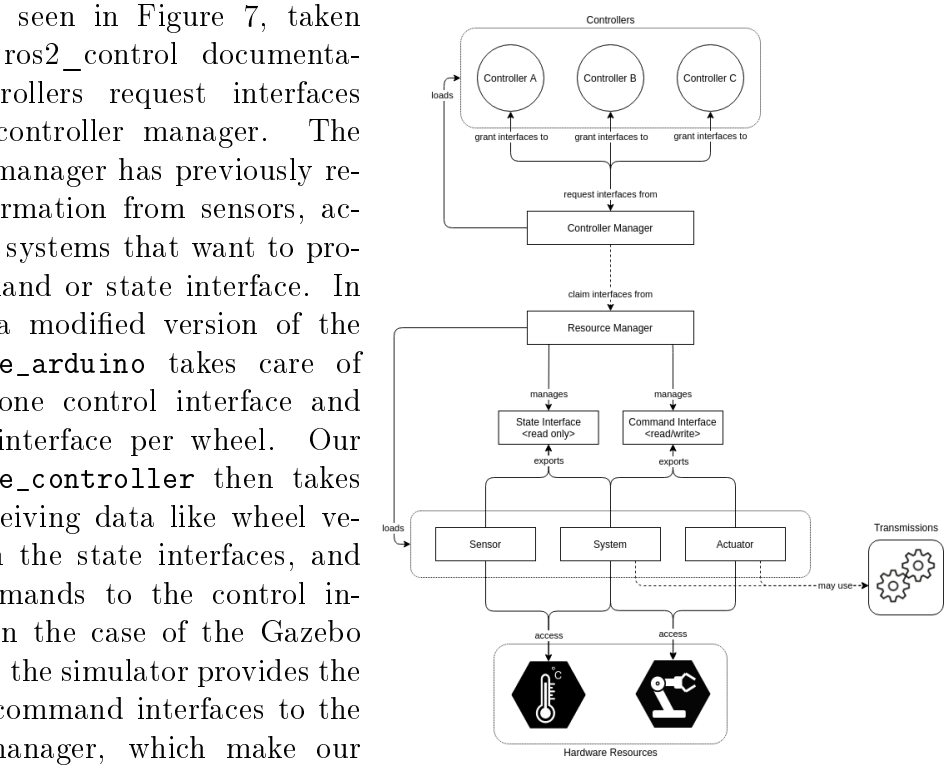



Figure 7: ROS2_control architecture

### 2.1.5 Issues encountered

The 6-wheeled rover, with its stainless steel frame and weak DC motors lacking encoders, was not the most suitable platform for the task of autonomous navigation. The lack of encoders on the motors made odometry calculations based on the LIDAR and inertial measurement unit unreliable. An attempt was made to do closed-loop

control with the aforementioned odometry, but the noise and the lack of per-wheel odometry made it impossible to have a stable and reliable control.

After encountering too many issues with the existing platform, we decided to create a new platform using closed-loop stepper motors, also called servos. Their closed-loop control and high torque at low speed would make them ideal for slow movements, thereby making the task of autonomous navigation feasible. A comparison of both platforms can be seen in Figure 8.

(a) Version 1 of the rover (b) Version 2 of the rover

Figure 8: Rover version 1 and 2

The new chassis is entirely 3D printed, making it fit exactly to our needs as can be seen by the CAD model in Figure 9. This make the overall platform more compact and lighter. The LIDAR was also placed in the front at an angle to more easily cover the ground, which is needed for the terrain analysis that will be described in section 3.1.4. The new platform is now four wheeled, but was designed with modularity in mind to be able to add two additional wheels and motors in the center of the chassis.

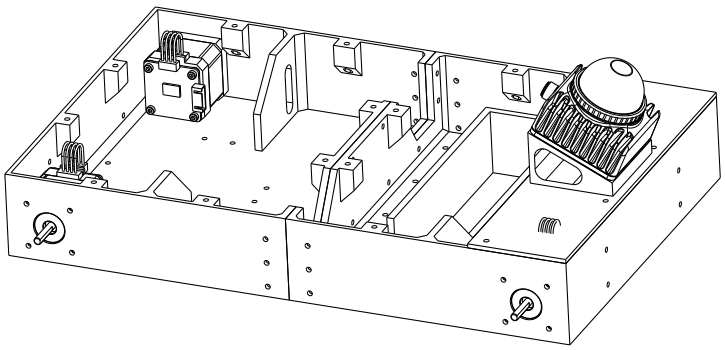


Figure 9: Drawing of CAD model of the new rover with the top removed

## 2.2 Quadcopter setup

One of the agents in our multi-agent system is a quadcopter, chosen for its ability to provide aerial perspective and navigate in three-dimensional space. After evaluating several drone options available to us, we selected a model that struck an optimal balance between payload capacity and size. This drone was capable of easily carrying the Livox Mid-360 LIDAR sensor and the Nvidia Jetson Orin embedded computer, while still maintaining a compact form factor suitable for indoor and outdoor operations.

The quadcopter platform required several modifications to integrate our specific sensor suite and computational hardware.

### 2.2.1 Mechanical modifications

I designed and 3D printed new landing legs that fit on the arms of the quadcopter. Those landing legs were designed to increase the landing stability, which I noticed was a problem in the manual flight I performed, and to reduce the blind spots of the LIDAR which was to be placed in the center of the underside of the drone.

I also took the opportunity to redesign the battery mounting mechanism which was bulky, heavy, and suitable to only one size of battery, to be much simpler and use Velcro straps as can be seen in Figure 10. This resulted in a saving of around 150 grams and thus increased the flight time of the drone. The Mid-360 LIDAR was mounted on the underside of the drone, in the center, to provide a full 360-degree view of the surroundings.

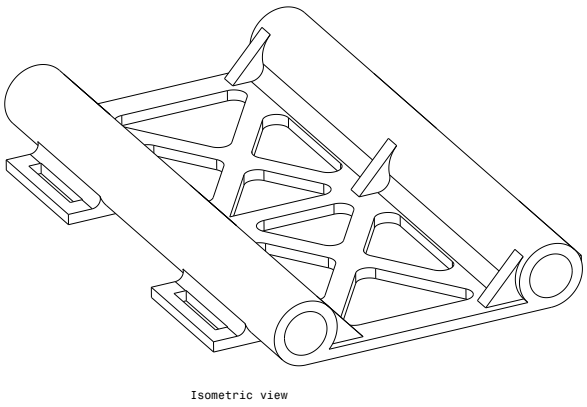



Figure 10: Drawing of CAD model of new battery tray

### 2.2.2 Electronics architecture

To keep the architecture similar to the one used on the wheeled robot, the quadcopter was also equipped with an Nvidia Jetson Orin embedded computer and a Livox Mid-360 LIDAR. The computer and the LIDAR get their power from the drone's battery, which is regulated to 19V and 12V respectively.

### 2.2.3 Software architecture and setup

Time didn't allow for the implementation of a full navigation stack on the drone. As such, only a theoretical architecture, similar to the previous platform, was planned.

The goal was to use a modified version of TARE planner for aerial navigation that would consider paths in 3D for the local planner, as demonstrated by the Carnegie Mellon University [5]. The embedded computer would then send the command orders to the flight computer, a Cube Orange+, via MavLINK one one of the flight computer's USB port.

For safety reasons, an operator with a radio controller would have been required for testing, as it is required by law to have the capability to have manual control at all time and to have the drone in view.

As time was running out, the drone was not fully operational and only manual testing was performed.

## 2.3 Quadruped platform setup

A quadruped was chosen as the third platform as it can navigate rough terrain and climb stairs. The platform chosen was the Unitree GO2, a quadruped robot with a 3D LIDAR and an embedded computer.

### 2.3.1 Compute backpack

Even though the Unitree GO2 has a LIDAR of its own and an embedded computer based on the Rockchip RK3588S inside, we decided to add an additional LIDAR on top of the platform and as such, we needed an additional embedded computer.

As is the case with the wheeled platform and the drone, we chose to use a Livox Mid-360 LIDAR and a Nvidia Jetson Orin embedded computer. Even though the quadruped already carries a 3D LIDAR, we chose to use a Mid-360 as the ladder produces around ten times more points per second (21600 points per second for the Unitree L1 LIDAR against the 200000 points per second of the Mid-360).

The compute backpack also provides access to a power port through an Amass XT-30connector and a gigabit Ethernet port on the robot that are usually covered by a plastic cover. Those ports are covered by the handle of the GO2, but a replacement plate with holes was designed as can be seen in Figure 11.

Figure 11: Handle and covered connectors

To fit the Nvidia Jetson, the LIDAR, and the power regulators needed for both of them, the top part of the robot was scanned, and a new top part was designed and 3D printed. The CAD model of top part and the scan can be seen in Figure 12. This holds the LIDAR on the back of the robot to be sure to cover the grounds during the scanning process. The angle was determined as to not have any of the robot itself in the field of view of the LIDAR. Additionally, the compute backpack was designed to accommodate other experiments like mounting a robotic arm on the robot. For that purpose, all four sides figure threaded mounting holes.

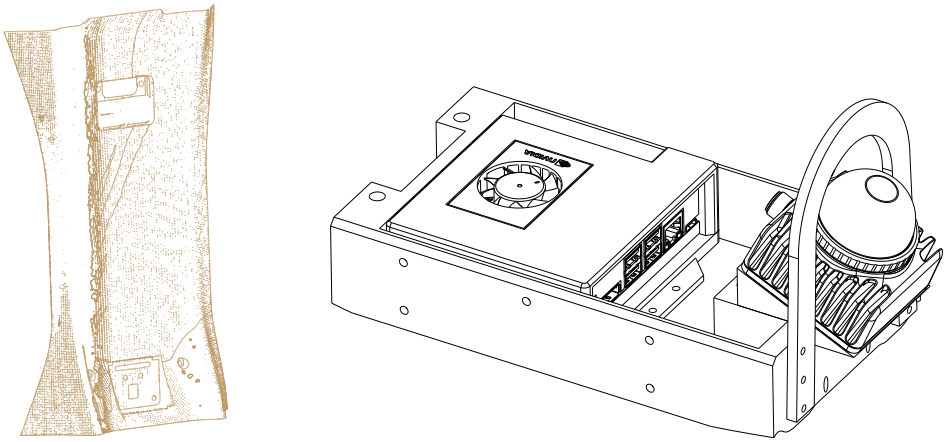


Figure 12: Scanned top of GO2 robot and drawing of CAD model of the compute backpack

The compute backpack's shell was 3D printed in PETG, as it is less brittle than many other filaments like PLA or ABS. This was done so that the shell be able to withstand potential falls of the robot. The compute backpack with the Jetson Orin, the Mid-360 LIDAR, the power regulators and the cabling can be seen in Figure 13. For development purposes, a video transmitter is also fitted to provide an easy way to see the Jetson's desktop without having to connect a monitor to it. As this is powered by USB, it doesn't require any additional power source.

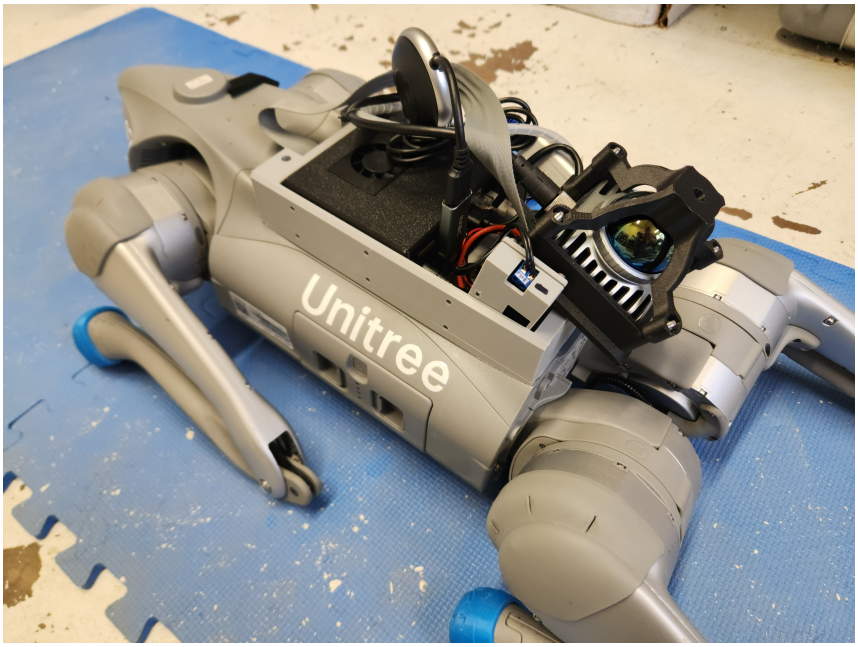



Figure 13: Compute backpack mounted on quadruped

### 2.3.2 Communication with the quadruped

To send command to the quadruped, Unitree offers a controller, a phone app, an SBUS port, a UWB remote, and an upgraded version of the robot that support CycloneDDS communication via Ethernet.

The controller uses both Bluetooth and a longer range 2.4GHz radio to communicate with the robot. This controller is only capable of sending movement commands,

predefined non-modifiable actions, and initiate calibration. The Bluetooth seems to be used for initialization but is not required at longer ranges and all control commands are sent via the 2.4GHz radio.

The phone app uses WebRTC, a real time communication protocol commonly used for web browser communication. In the case of the quadruped, the robot can either act as an access point (AP mode) and the phone then connects to the Wifi created, or both the phone and robot can connect to an existing Wifi network in station mode (STA mode). This communication is not encrypted and thus provides an easy communication to the robot that we can emulate [28]. Compared to the remote control, we also get information back from the robot, including but not limited to : joint angles, preprocessed LIDAR data, odometry estimation, camera feed. This communication method has the advantage that it is simple and gives us most of the data, that is why it was used for the first control tests of communication between a laptop and the robot. It however doesn't provide data at a high rate of speed at only 10 Hz for most of the data, and doesn't give access to the raw LIDAR measurements for example. It also doesn't allow us to control each joint individually if we wanted to implement additional gaits in the future.

The SBUS, or Serial Bus, port that is present on the top of the robot is meant for use with long range radio controllers, often used in radio controlled drones. This only gives us high level control of the movement and does not provide any data back.

The UWB, or Ultra Wide Band, remote is an optional feature of the robot that provides the capability of person following to the robot. This remote also features a small joystick for manual control and uses the same technologies as the main controller. The UWB uses multiple antennas in the robot to estimate the position of the tracker that a person carries, giving a rough estimation of the position of the remote which the robot can then follow. However, this is not something that will help us in the control of the robot.

Finally, the robot also features an Ethernet port, connected to the internal computer, as can be seen in Figure 11, and as will be discussed in section 2.4 this can provide a Data Distribution Service (DDS) using the CycloneDDS protocol. This protocol was developed by Eclipse and is an implementation of the Object Management Group (OMG) DDS. This protocol is used in many industrial applications and is known for its low latency and high reliability. CycloneDDS is one of the ROS MiddleWares (RMW) that can be used, and it is often praised for it's low latency and low CPU and memory usage. This way of communicating also offers the most control from very low level commands of each joint and raw LIDAR data, up to high level commands like predefined gaits and actions defined by Unitree. This is the communication method that we will use for the final implementation of the project.

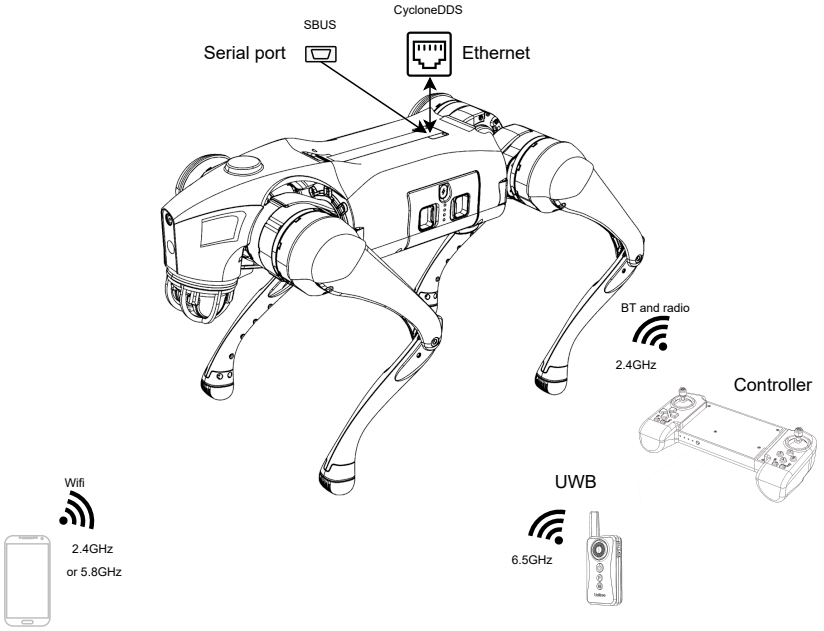
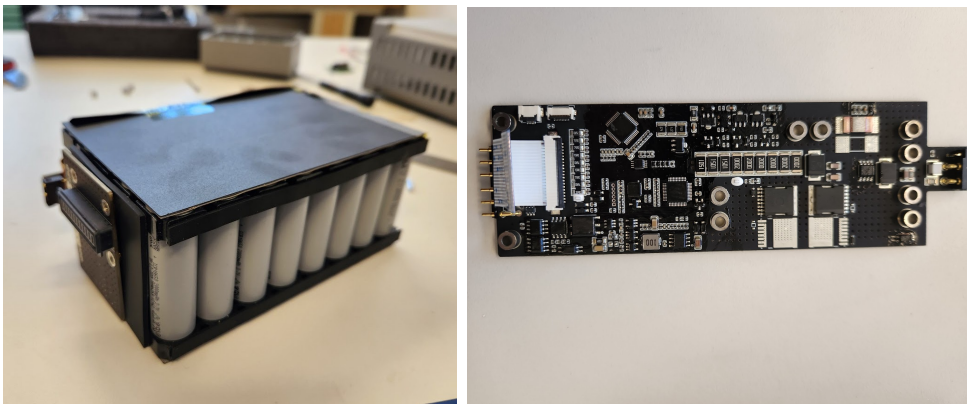



Figure 14: Communication methods with the GO2 robot

## 2.4 Issues encountered

As delivered, the robot was not functional as the battery was discharged beyond the point of no return. The battery was disassembled (see Figure 15) and partly reverse engineered to understand the problem. Even though the battery was not repairable, the reverse engineering allowed us to understand the battery's BMS and how it communicated with the robot. In Table 2.4 it can be seen that I was able to partly reverse engineer the battery communication protocol. This was shared with others from the open source community actively trying to design a replacement motherboard for the robot. The battery was then replaced by a new one and offered around two hours of walking with the LIDAR and the Jetson Orin powered on.



(a) Disassembled battery (b) Battery Management System (BMS)

Figure 15: Disassembled battery and Battery Management System (BMS)

One of the other issues encountered happened after a software update of the robot. We weren't able to connect via SSH to the Linux computer inside the robot.

| Byte | Data | Conversion | Comment |
|---|---|---|---|
| 0 | EF FE | -4098 | start of battery message |
| 2 | 3C 03 | 15363 | start of battery message |
| 4 | 0F 00 | 3840 | cell 8 voltage in mV |
| 6 | 0F 01 | 3841 | cell 7 voltage in mV |
| 8 | 0F 01 | 3841 | cell 6 voltage in mV |
| 10 | 0F 03 | 3843 | cell 5 voltage in mV |
| 12 | 0F 01 | 3841 | cell 4 voltage in mV |
| 14 | 0E FF | 3839 | cell 3 voltage in mV |
| 16 | 0F 01 | 3841 | cell 2 voltage in mV |
| 18 | 0F 01 | 3841 | cell 1 voltage in mV |
| 20 | 77 FB | 30715 | battery total voltage in mV |
| 22 | 01 29 | 297 | |
| 24 | FF F8 | -8 | |
| 26 | FF FF | -1 | |
| 28 | 64 00 | 100 | SoC in percent |
| 30 | 09 BF | 2495 | temperature 24.95 C |
| 32 | 09 C6 | 2502 | temperature 25.02 C |
| 34 | 09 A4 | 2468 | temperature 24.68 C |
| 36 | 09 A8 | 2472 | temperature 24.72 C |
| 38 | 00 00 | 0 | |
| 40 | 57 70 | 22384 | |
| 42 | 09 01 | 2305 | number of charge or discharge cycles |
| 44 | 0D 06 | 333 | |
| 46 | 00 00 | 0 | |
| 48 | 00 00 | 0 | |
| 50 | 00 00 | 0 | |
| 52 | 00 FF | 255 | |
| 54 | 00 00 | 0 | |
| 56 | 47 80 | 18304 | crc32 |
| 58 | 7E 9D | 32413 | crc32 |

Table 2: Partly decoded battery protocol

| Address | Original value | Original Instruction | Patched value | Patched instruction |
|---|---|---|---|---|
| 565BD80 | E0 7B BF A9 | stp x0, x30, [sp,#-0x10]! | 80 00 80 D2 | mov x0, #4 |
| | 63 4D 00 94 | bl sub_566F310 | E0 7B BF A9 | stp x0, x30, [sp,#-0x10]! |
| | 84 00 00 18 | ldr w4, #0x565BD98 | 63 4D 00 94 | bl sub_566F310 |
| | 84 7C 40 93 | sxtw x4, w4 | 64 00 00 98 | ldrsw X4, #0x565BD98 |

Table 3: Patched instructions for version check

This required many days of reverse engineering of the new firmware with the help of other people online.

The update had removed a vulnerable update mechanism that previously allowed us to gain a root shell to the robot. With that shell, we were then able to change the password of the robot and gain access to the robot via SSH. The new firmware had also prevented us from using the Ethernet port of the robot to communicate to the robot using the CycloneDDS which is one of the Data Distribution Services that ROS can use. This functionality of the robot is usually reserved for the educational version of the GO2, but the company doesn't offer any upgrade from the standard version to the educational version.

As such, I helped the open source community to design a new unlocking method. For reference, in the previous version, only a version number in a single file needed to be changed from a 2 to a 4 to unlock every functionality of the robot. The new update brought many new challenges as every binary was now heavily obfuscated and had increased in size by a factor of 10, every file's checksum was now verified by a yet unknown program, no debugger could be attached as every binary checked their parent process, and many other obfuscation techniques were used.

The anti debugger technique was quickly bypassed as only the name of the parent process was checked against a list of names, `gdb-server` not being part of this list made it easy to debug the binaries. Every binary also checked for the presence of a `tracepid` value, which was solved at first with kernel module that hides the presence of the `tracepid` value. Other threads were also created to continue monitoring those values in the binaries, but patching the `pthread_create` instructions to `MOV X0, #0` solved that problem.

Once the location of the checksum check was identified in a binary, the instruction responsible for the comparison was patched to always return true : the previous instruction `BL <compare>` was replaced by `MOV X0, #0`.

Finally, the version check was bypassed by consolidating two instruction in one, to in terms have room to set the value of the register `X0` to 4 as can be seen in Table 2.4.

# 3 Mapping, planning, and exploration

There are three main components to the software architecture of the multi-agent system: mapping, planning, and exploration. Mapping first involves knowing where the robot is and how it is moving, or it's odometry. Planning is often divided in two scales, a local one where we consider what movements the robot is capable of doing, a global one that aims to find the best path to a distant goal. Exploration uses the two former process to determine where the robot should go to maximize the information gathered.

## 3.1 Mapping

Mapping is the action of aligning the robot's acquired LIDAR scans in relation to the initial pose. For that, one or multiple methods of odometry are needed.

### 3.1.1 Introduction to SLAM

SLAM or Simultaneous Location And Mapping is the action of using a scanner, in our case a LIDAR, for both the construction of a map, and utilizing this map to infer our position. In 2D, algorithms like Monte Carlo based particle filters [10], or graph based approaches [18] are used and have not been lately improved. In 3D, the problem is much more complex and the algorithms are much more computationally expensive.

When using SLAM as an odometry method, problems like drift can occur. As can be seen in Figure 16, the drift of the odometry can be seen in map acquired in a large basement : a doubling of the corridors one the right of the image can be seen. This drift is due to inaccuracies in the odometry, often acquired in areas with few features to align the scans to. This can be remedied with methods that use loop closure [6] [14].

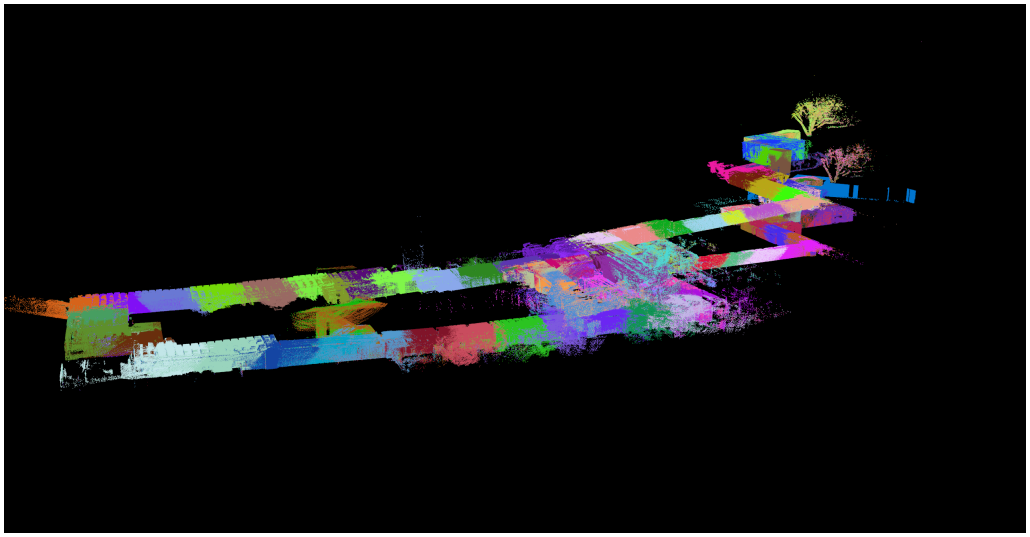


Figure 16: Scan of a large IIT basement (150m in length) showing the drift of the odometry. Each color represents a 10s scan.

### 3.1.2 LIDAR inertial odometry

LIDARs are often paired with Inertial Measurement Units (IMUs) as algorithms can be conceived to make use of both the LIDAR and the IMU for odometry. Registra-

tion of features of the last scan of the LIDAR with previous ones provide long term accuracy and minimize potential drift, while the IMU provide a high rate of movement data that can be used to both facilitate the alignment, as well as to undistort or unskew the scans that were captured during fast movements.

As the Livox Mid-360 also carries an IMU, this is the reason why I turned myself to those kinds of algorithms for the odometry.

One such algorithm is FAST-LIO2 [29], which I chose for being close to the state of the art and having a version available that was compatible with the Mid-360 LIDAR. A more detailed comparison of the different LIDAR inertial odometry algorithms will be presented in the section 3.1.3. The steps this algorithm takes are shown in Figure 17, taken from the paper. As an implementation of the algorithm was available, I did not have spend enough time to understand every step of it, but a state of the art of LIDAR inertial odometry algorithms will be presented in the next section to understand what were the improvements of FAST-LIO2 over previous works.

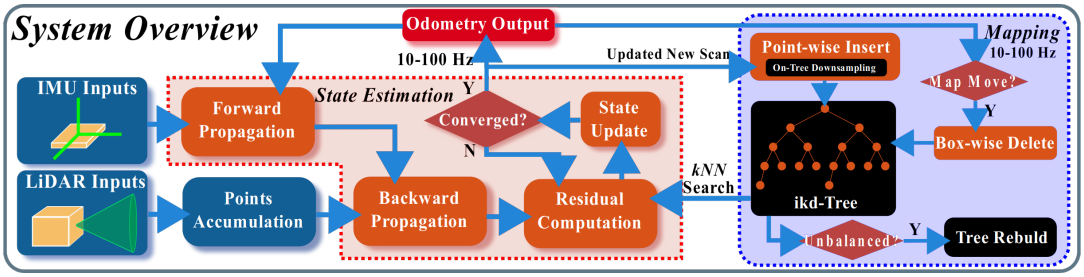



Figure 17: FAST-LIO2 algorithm description taken from [29]

### 3.1.3 Comparison of LIDAR inertial odometry algorithms

The comparison of LIDAR inertial odometry algorithms was chosen as it is the data that is available to us. LIDAR inertial odometry algorithms can be divided in two categories based on how the LIDAR and IMU data are fused : loosely coupled and tightly coupled.

Loosely coupled methods estimate the state of each sensor independently, combines the measurement with weights, and then determines the position of the robot. Those methods are useful when both sensors are independent and changes to them want to be tested. The weights can also be changed depending on the sensors used and on their accuracy and noise, or dynamically if a sensor fails or malfunctions. However, we have a LIDAR package that integrates an IMU, making the aforementioned benefits not relevant to us.

Tightly coupled methods, on the other hand, use the measurements of all sensors to estimate the robot's position. This can mean using the IMU readings to pre process the LIDAR measurements. Those methods however require a more complex algorithm and can be more computationally expensive. They can also be more prone to failure as the failure of one sensor entails the failure of the odometry system.

LOAM (LIDAR Odometry And Mapping) [13] was the first loosely coupled method to be proposed and offered low computational requirements. It used the IMU measurements to estimate velocity while the LIDAR measurement used the velocity estimation for undistortion to offer better position estimation. LeGo-LOAM [26] improved on the works of LOAM by improving the performance of the pose

estimation and by adding ground segmentation, making the method more robust in ground based applications.

As for tightly coupled methods to be proposed in the Zebedee paper [3] showed the great accuracy the tightly coupled methods can offer. It demonstrated stable mapping of a scene by placing a 2D LIDAR on a spring mobile base. LIPS [11] formalized the closest point plane representation and added IMU preintegration for more accurate pose estimation. Ye et al. [32] added joint optimization of the LIDAR and IMU measurements, improving the performance in high drift scenarios in long term mapping. This paper also added a rotation-contrained refinement which improves the consistency of the global map. FAST-LIO [30] greatly improved the performance of previous works by introducing the use of a tightly coupled iterated Kalman Filter. The high computing efficiency of the proposed gain calculation formula improves the performance, making this method interesting for real time applications on embedded systems. FAST-LIO2 [29] improved on FAST-LIO by getting rid of the feature extraction step and replacing it registration to the global map. It also improved performance by switching to an incrementally updated kd-tree data structure for map storage. POINT-LIO [12] employs a different method to the previously mentioned algorithms by using a point-wise position estimation providing a much higher frequency of odometry updates.

### 3.1.4 Terrain analysis

To determine the navigable and non-navigable areas, the aligned point cloud gathered by the LIDAR and the odometry are analyzed. The first analysis is one at a closer range, while the other has a longer range and a longer decay time. The topics that the two analysis subscribe and publish to are shown in Figure 18. The extended range analysis is used by some planners to determine the best path to take while the close range analysis is used to determine the immediate surroundings of the robot as is showcased in Figure 18.

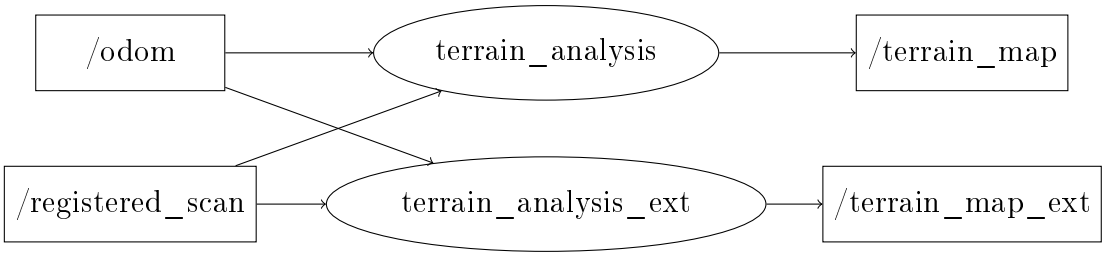
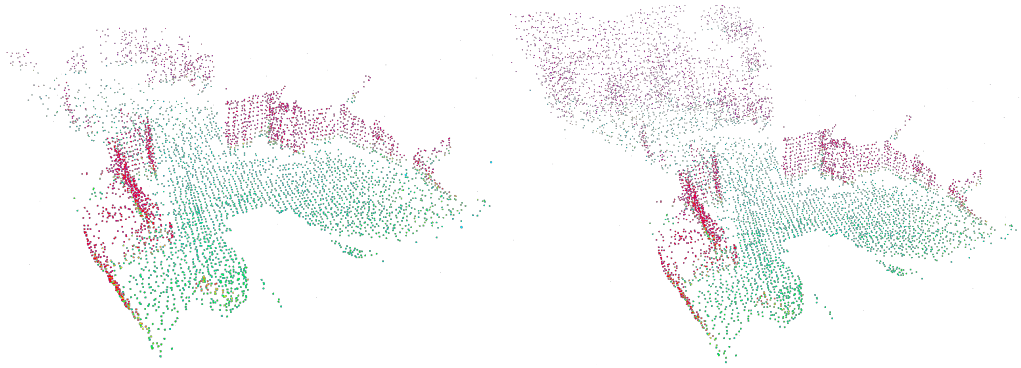



Figure 18: Subscriptions and publications of terrain_analysis and terrain_analysis_ext



(a) Close range terrain analysis (b) Extended terrain analysis

Figure 19: Comparison of close range and extended range terrain analysis

This terrain analysis paradigm is derived from the works of Ji Shang et al. on fast likelihood-based collision avoidance [34], as is the local path planning that will be described in section 3.2.2.

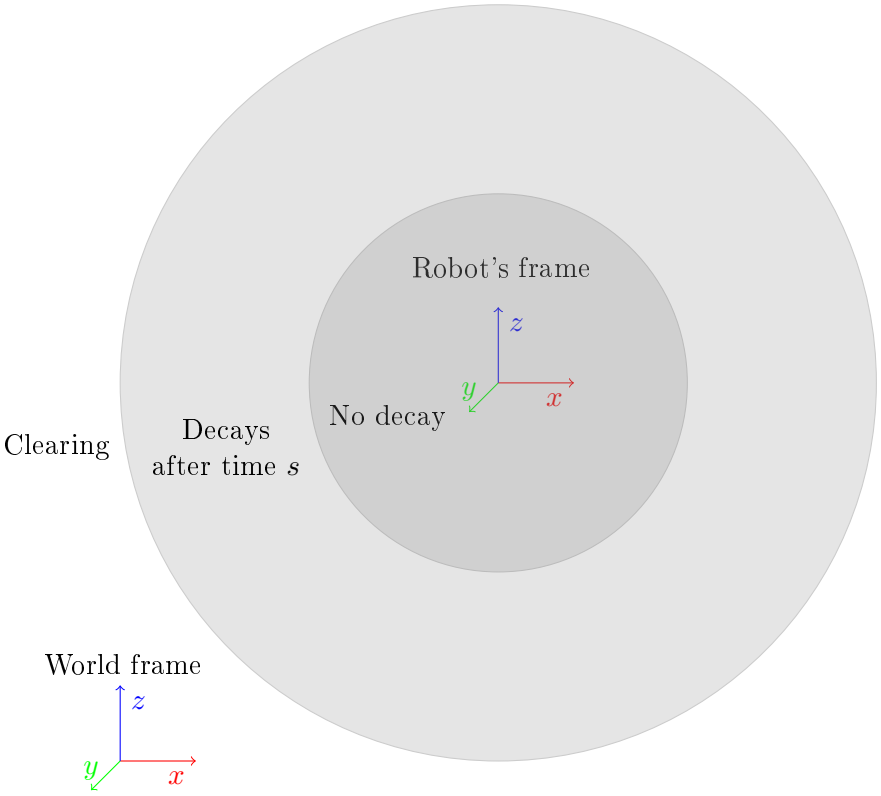



Figure 20: Decay of the terrain analysis in different zones

The point cloud is split into three zones. In the first zone, the one closest to the robot, the terrain analysis voxels don't get modified but can only get added. This is done to ensure that the robot doesn't forget obstacles when they get closer than the minimum distance the LIDAR can measure. This value was set to 0.1 m for the quadruped for example. The second zone is the zone within which a voxel that has not been hit by the LIDAR for a certain amount of time $s$ get removed. This ensures that the robot doesn't get by obstacles that have been removed from

the point cloud, like in the case of a human passing by. The third and final zone is the zone within which we clear any terrain voxel present. This is done to ensure

## 3.2 Planning

The problem of path planning in 2D is a well known problem in robotics and has been solved many times. However, the problem of path planning in 3D is much more complex, anly recently have efficient solutions been found.

### 3.2.1 2D path planning

As a first test of the wheeled and quadruped platform, a simple 2D navigation was put in place to showcase the navigation of the platforms on flat ground. Nav2 [19] was used with a simple Smac planner [17] to navigate the platforms. The 3D point cloud was converted to a 2D one by taking only a range of the points relative to the robot's height, the corresponding topic graph can be seen in Figure 22. This was done to showcase the navigation capabilities of the platforms and to test the odometry algorithms. Nav2 was chosen as it is a well known and well documented package that is compatible with every version of ROS2.

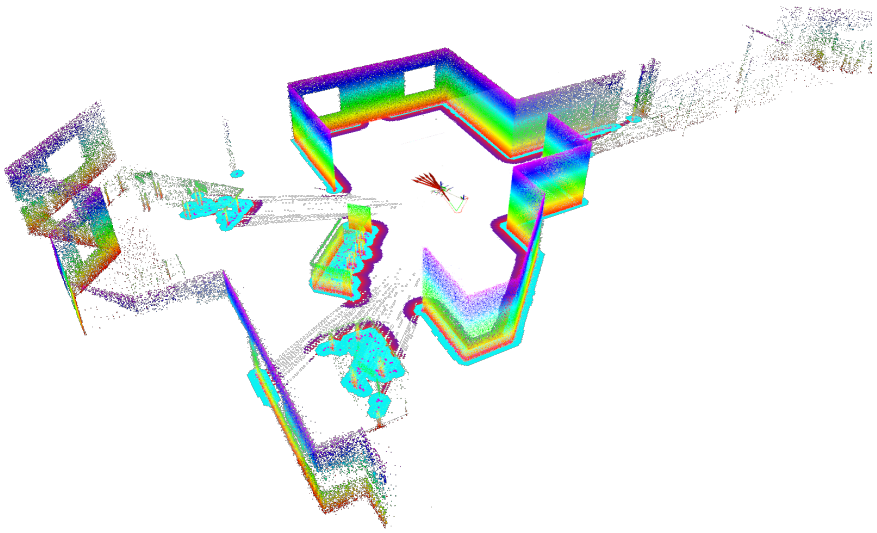


Figure 21: Nav2 creating multiple costmaps in a simulation, a gradient of colors can be seen around the walls

In Figure 3.2.1, the costmaps, used for navigation, generated by Nav2 can be seen. They represent areas that are free to navigate, areas that are occupied by obstacles, and areas that are unknown. In Figure 3.2.1 they can be seen as purple and blue colored zones around walls and obstacles.

### 3.2.2 3D local path planning

The local path planning in a 3D environment makes use of the Falco paper [34]. In this paper, a method is proposed where the environment is considered deterministically known within a certain range, usually the range of the LIDAR, and

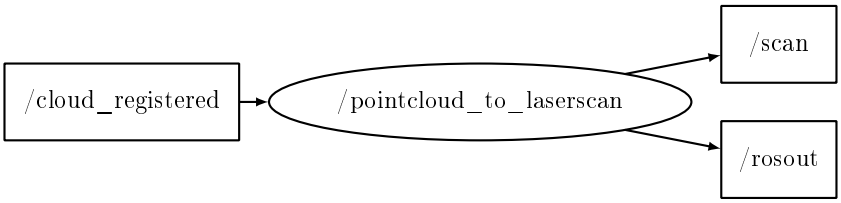



Figure 22: Conversion from a 3D pointcloud to a 2D scan

probabilistically known beyond the sensor's range. This contrasts with usual methods that require an online search of a graph that needs updating at every sensor reading. The proposed method by Ji Shang et al. eliminates the need for the online search, favoring a path that maximizes the likelihood to reach the goal, instead of selecting the shortest path like traditional methods.

ADD LOCAL PLANNER PATHS RVIZ2

Figure 23: Considered paths by the local planner

The local planner uses the terrain analysis mentioned in section 3.1.4 to determine the navigable areas. It also subscribes to the odometry, referred to as "/state_estimation" in the figure 24, and the registered scan. Its main goal is obtained from the topic "/way_point" that is generated by the global planner. The local planner then publishes the best path to the topic "/path" that is used by the controller to move the robot, as well as all free paths considered on the topic "/free_paths". As can be seen in figure 24, the local planner also makes sure to only consider paths that are within the exploration boundary if one is defined by subscribing to the topic "/exploration_boundary".

Setting the correct margin of inflation for the obstacles and the min/max obstacle height for the real robot was a challenge. The margin needed to be large enough to not run into walls and obstacles, but small enough so that the planner would go through tight doorways and passages.

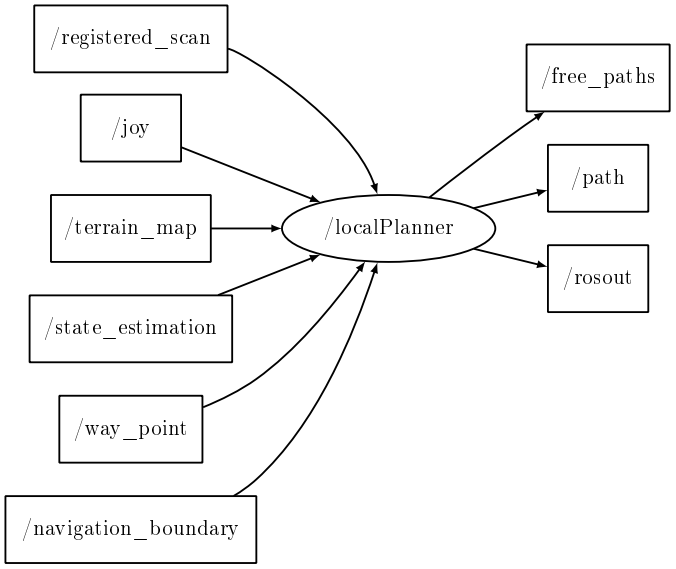



Figure 24: localPlanner subscriptions and publications

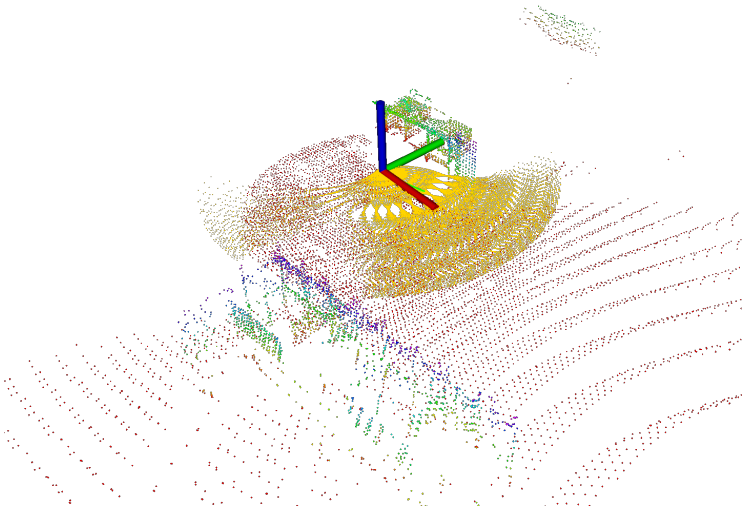


Figure 25: Local planner in simulation, the yellow points are the considered paths, the other points are from the terrain analysis. The robot is represented as a set of axis

### 3.2.3 Comparison of 3D planning algorithms

## 3.3 Exploration

### 3.3.1 Metrics for exploration

### 3.3.2 TARE planner

TARE [4], is a hierarchical framework for exploring unknown 3D environment. This planner was designed for Carnegie Mellon University's participation at the 2021 DARPA subterranean challenge [8]. In Figure 26, the exploration of one of DARPA's environments can be seen explored by tare planner. The planner doesn't use any prerecorded point cloud for exploration, but uses a viewpoint candidate method for exploration. The planner is able to explore large environments by building a reduced local path, while maintaining a coarse global path.

Viewpoint candidates are generated by the planner within the local planning horizon (see Figure 27), and on valid positions of the terrain map. Each candidate is then evaluated for the amount of information it would bring to the map. This is done by evaluating the uncovered surface of the map, and evaluating the amount of those areas the viewpoint candidate would cover from the LIDAR's perspective.

This method also implements a hierarchical exploration strategy where the entire exploration path is considered, rather than greedily maximizing the exploration rate. In contrasts to other methods such as Motion primitive-Based exploration with path-Planner (MBP) [9] or the Next Best View Planner (NBVP) [2], this method is able to explore large environments with a high exploration rate.

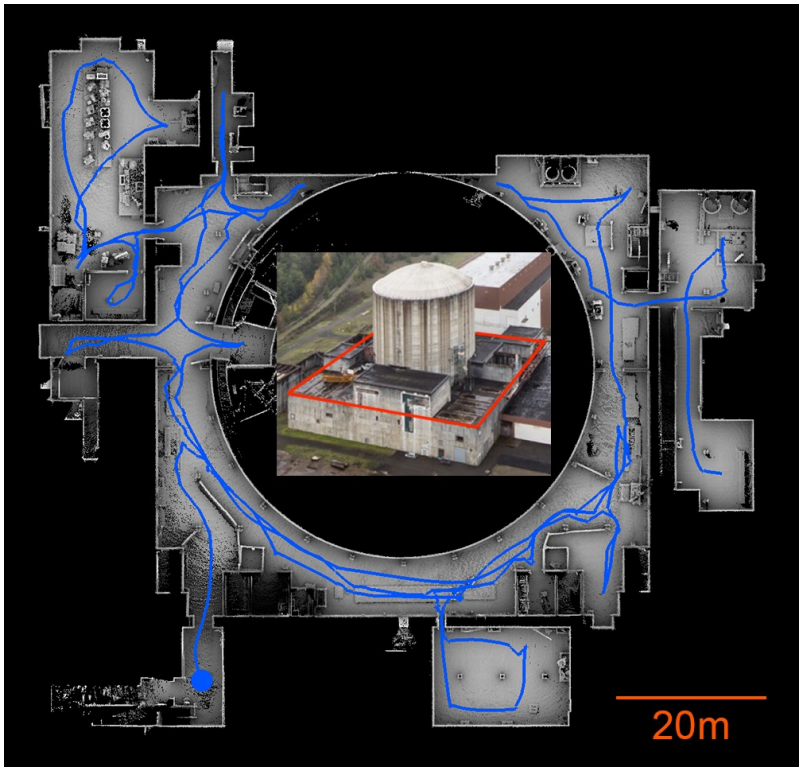



Figure 26: TARE planner exploration during DARPA SubT challenge

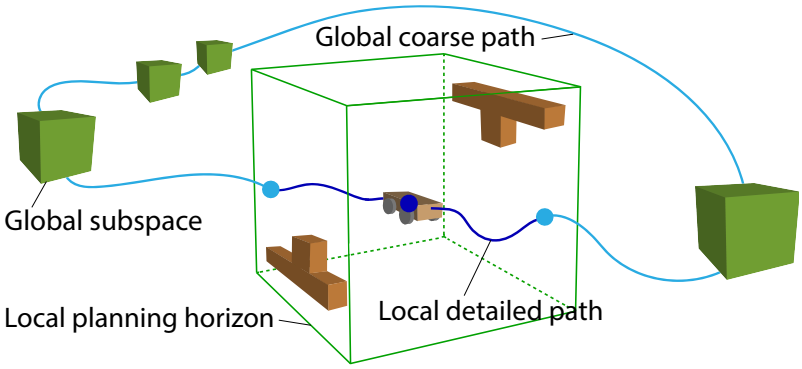



Figure 27: Local and global paths of the TARE planner, a dense map is conserved only for the local planning horizon, while a sparse map is kept for the global planning

### 3.3.3 MUI-TARE planner

The MUI-TARE planner [31] (Multi-agent TARE with Unknown Initial position) is an extended version of the previously described TARE planner that was used by the CMU team during the multi robot exploration stage of the DARPA SubT challenge. It uses AutoMerge [33], a framework meant for merging multiple large scale point clouds. MUI-TARE follows the hierarchical exploration strategy of TARE, where the entire exploration path is considered, maximizing the exploration rate.

Until maps get initially merged, and their relative positions to one another is unknown, each agent plans its exploration independently. Once the maps are merged, the agents then cooperate in the exploration effort, by continuously sharing their maps. If communication is lost, the agents will resynchronize their maps by going back to a known meeting point. The meeting point is continuously updated while

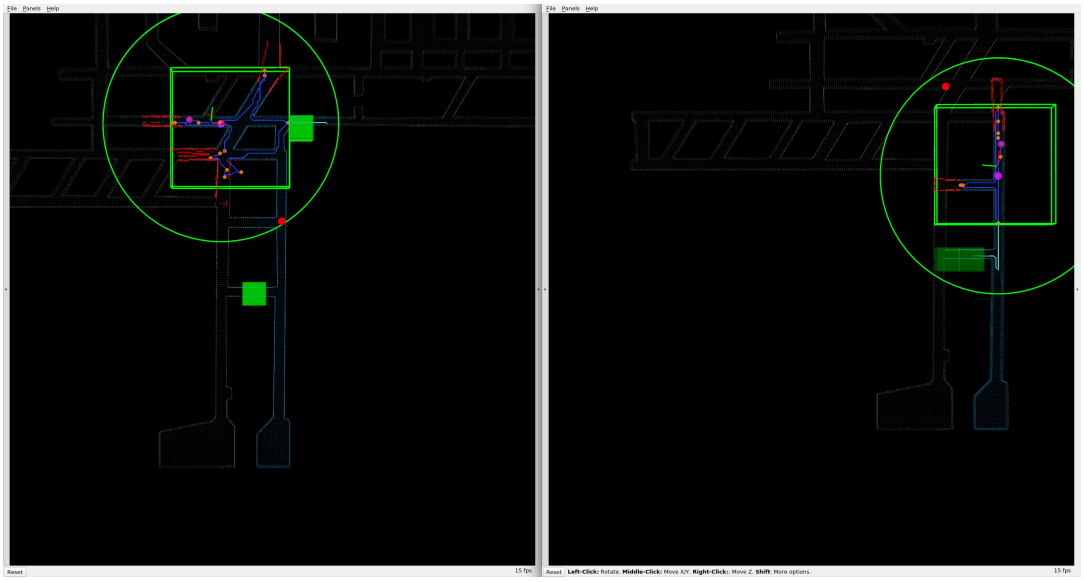


Figure 28: Simulation of two agents exploring a tunnel environment, the agents' simulated communication ranges are represented by the green circles

communications are active, but only after a certain amount of time has passed without communication will the agents go back to the last known meeting point. This ensures that the agents can continue to explore distant areas without having to go back to the meeting point too often.

As a simulation environment is provided, the MUI-TARE planner was first tested in simulation, allowing myself to understand the planner and to test the planner in a controlled environment. Such test can be seen in Figure 28. The simulation environment is provided as a Docker image

While this algorithm was designed for the DARPA SubT challenge, it can be used in other exploration scenarios, such as the exploration of a large building in construction. That is why efforts were made to make the planner compatible with the quadruped and wheeled platforms. However, its high complexity and it's need to run a combination of ROS1 and ROS2 nodes made it difficult to integrate. Additionally, the communication method used during the DARPA SubT challenge was not disclosed, and time constraints made it impossible to implement a communication method that would work. We considered mesh networks, like the one used by the LoRaWAN [24] protocol, but the high latency and low bandwidth made in unsuitable to send pint clouds. Some teams solutions were provided [25] but often rely on expensive and complicated hardware, often having multiple people per team dedicated to the communication aspect of the challenge.

As such, the MUI-TARE planner was not integrated into the robots but remains a good candidate for the future. Additionally, the planner is proven to work in simulation, and has been tested during the DARPA SubT challenge, showcasing its robustness and efficiency. We believe this planner could thus be used in future exploration scenarios with the platforms already built, or with new platforms, by simply adding the necessary communication hardware and software.

### 3.3.4 Comparison of exploration algorithms

# 4 Simulation

In robotics, simulation is a key part of the development process. Unlike traditional software development, the testing of algorithms needs to be done on the physical platform, which can be time consuming and costly. Simulation allows for the testing of algorithms in a controlled environment, where the behavior of the robot can be closely mimicked. It also removes any risks associated with testing on the physical platform, such as damaging the robot or causing injury.

## 4.1 Choosing a simulation environment

To test the algorithms and the coordination of the different platforms, a simulation environment was needed. The choice of the simulation environment was based on the following criteria : being able to simulate multiple platforms, being able to simulate the sensors we had on the physical platforms, and the portability of the simulation environment.

### 4.1.1 Gazebo

Gazebo is a well known simulation environment in the robotics community. It is widely used and has a large community. It is open source and has a lot of plugins available, for simulation sensors, motion platforms and more. As I was already familiar with it, I first looked at it to build a crude simulation of the 6 wheeled platform. Thanks to the use of ROS2 and ROS2-control for the drive train, I was able to quickly build a simulation of the platform.

I was also able to find a working simulation of the Livox Mid-360 LIDAR sensor we chose for every platform. The package [16] was a fork of the original simulation package from Livox [15] and was modified to work with the specific LIDAR we are using.

### 4.1.2 Nvidia Isaac Sim

Isaac sim is a high fidelity simulation environment developed by Nvidia. This simulation environment uses a PhysX based physics engine and is able to simulate multiple platforms at once. It is also able to simulate sensors like cameras, LIDARs and IMUs. The main advantage of Isaac sim is its high fidelity and parallelization. However, this also brings a lot of complexity and the need for an RTX GPU to run it.

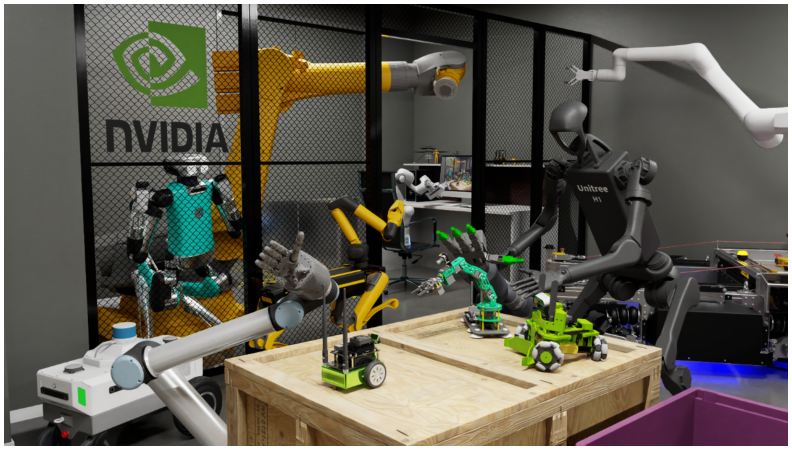



Figure 29: View of some built-in robots in Isaac Sim

The high overhead, complexity, low portability and the fact that it was not open source made me choose Gazebo over Isaac sim for the simulation of the platforms.

In addition, the LIDAR we are using has a non-repetitive pattern, which is not currently supported by Isaac sim and would have required a lot of work to simulate.

I did experiment with Isaac sim to simulate the quadruped platform, as it was the most complex platform to simulate and there existed a simulation of the robot in Isaac sim.

Some built-in robots in Isaac sim are shown in Figure 29. We can also observe the high fidelity of the simulation environment and the lighting effects that Isaac Sim is capable of.

I did end up experimenting with Isaac Sim for simulating the quadruped platform, as examples are available for reinforcement learning of locomotion for quadrupeds. The simulation of the quadruped platform in Isaac Sim can be seen in Figure 30.

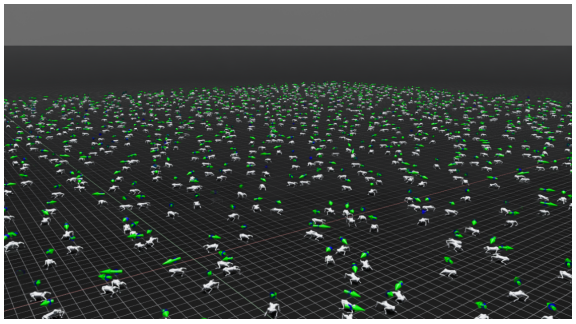


Figure 30: Quadruped platform reinforcement learning training in Isaac Sim

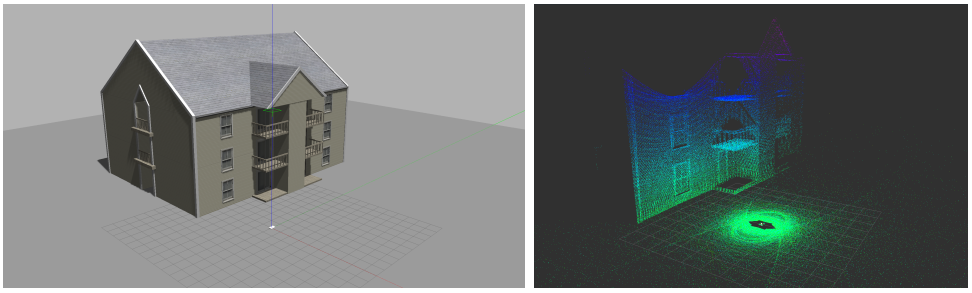
## 4.2 Simulating sensors

To have an accurate simulation that closely mimics the physical platform, the sensors on the physical platform need to be simulated. The sensors that were simulated were the LIDAR and the IMU. Even though not necessary, the camera and depth camera were also simulated to have a complete simulation of the platform.

### 4.2.1 LIDAR

Gazebo comes with a way of simulating 2D LIDARs, but no built-in way of simulating 3D LIDARs. This functionality was introduced in a recent version of Gazebo,

but it's incompatibility with older Gazebo worlds and robot model made it difficult to justify changing version when a lot of work was already invested in the simulation. Moreover, the LIDAR we are using has a non-repetitive pattern, which is not currently supported by Gazebo build in GPU LIDAR that is present in the newer version.



(a) Gazebo simulation environment

(b) Point cloud from simulated LIDAR

Figure 31: Simulated LIDAR in Gazebo and point cloud in RVIZ2

That is why I relied on a fork [27] of Livox original simulation plugin was used. This plugin offers a simulation of the Mid-360 LIDAR that we are using, and offers a way to simulate the exact messages that the LIDAR driver would send. Additionally, the package offers a close simulation of the non repeating pattern of the LIDAR by using recorded laser angle measured from the real LIDAR and using this data to simulate the LIDAR in Gazebo. As we can see in Figure 31, the LIDAR simulation is close to the real LIDAR point cloud, and does not feature repeating line of traditional multi-line 3D LIDARs. No effort were made to make this simulation of the LIDAR more accurate, however, we can note that in Figure 31, the point cloud does not feature the same blind spots as the real LIDAR as can be seen in Figure 32. This is a point that could be improved in the future.

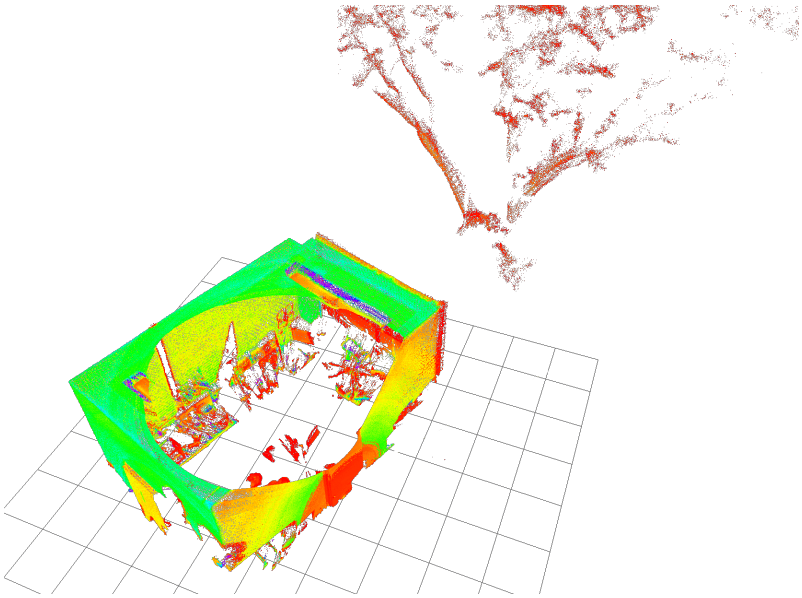


Figure 32: Blind spots in the ceiling and the floor, in a real static LIDAR scan

### 4.2.2 IMU

Gazebo has a built-in IMU sensor that can be added to the robot's description, which our mapping algorithm needs for point cloud deskewing and alignment as discussed in 3.1.2. The IMU sensor can be configured to have a certain update rate, noise, and bias. Noise can be added to the IMU sensor to simulate the noise in the IMU data. An example of an IMU sensor configuration can be seen in listing 1.

```
<gravity>true</gravity>
<sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize>
    <topic>default_topic</topic>
    <plugin filename="libgazebo_ros_imu_sensor.so" name=
       "imu_plugin">
    <topicName>imu</topicName>
    <bodyName>imu</bodyName>
    <updateRateHZ>100.0</updateRateHZ>
    <gaussianNoise>0.0</gaussianNoise>
    <xyzOffset>0 0 0</xyzOffset>
    <rpyOffset>0 0 0</rpyOffset>
    <frameName>imu</frameName>
    <initial_orientation_as_reference>false</
       initial_orientation_as_reference>
    <noise>
        <type>gaussian</type>
        <rate>
            <x>0.01</x>
            <y>0.01</y>
            <z>0.01</z>
        </rate>
        <acceleration>
            <x>0.01</x>
            <y>0.01</y>
            <z>0.01</z>
        </acceleration>
    </noise>
    </plugin>
</sensor>
```

Listing 1: IMU Sensor Configuration

### 4.2.3 Camera

In gazebo, cameras can be simulated by modifying the robot's description and adding a sensor tag to the robot's description. The camera sensor can be configured to have a certain field of view, resolution, and clipping planes. The camera sensor can also be configured to always be on, to have a certain update rate, to visualize the image, and to publish the image to a certain topic. An example of a camera sensor configuration can be seen in listing 2.

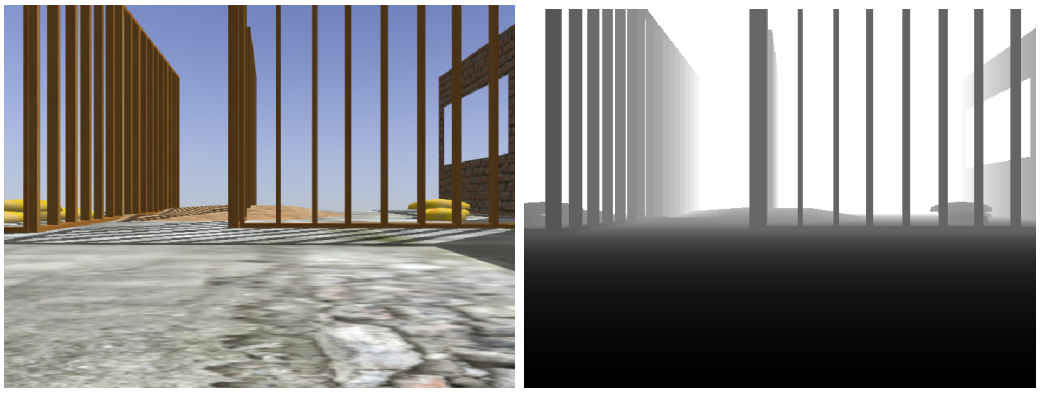
```
<sensor name="camera" type="camera">
    <camera>
        <horizontal_fov>1.047</horizontal_fov>
        <image>
            <width>320</width>
            <height>240</height>
        </image>
    </camera>
    <always_on>1</always_on>
    <update_rate>30</update_rate>
    <visualize>true</visualize>
    <topic>camera</topic>
</sensor>
```

Listing 2: Camera Sensor Configuration

Similarly, a depth camera can be simulated, as we intend to add am Intel RealSense D435 to the quadruped platform. The depth camera can be configured to have a certain field of view, resolution, clipping planes, and update rate. As depth camera tend to be noisy in the depth data, the noise can be added to the depth data. An example of a depth camera sensor configuration can be seen in listing 3. Here, the noise is set to be Gaussian with a mean of 0.0 and a standard deviation of 0.01. Those values were not measured but are a good starting point for the simulation. In the case of the depth camera, the clipping planes are set to 0.05 and 8.0 meters, as the depth camera is not able to see further than 8 meters. The depth camera is also configured to have a reduced resolution of just 640x480 pixels. Both color and depth views can be seen in Figure 33.

```
<sensor name="camera" type="depth">
    <visualize>true</visualize>
    <update_rate>10</update_rate>
    <camera>
        <horizontal_fov>1.089</horizontal_fov>
        <image>
            <format>R8G8B8</format>
            <width>640</width>
            <height>480</height>
        </image>
        <clip>
            <near>0.05</near>
            <far>8.0</far>
        </clip>
    </camera>
    <plugin name="camera_controller"
    filename="libgazebo_ros_camera.so">
    </plugin>
</sensor>
```

Listing 3: Depth Camera Sensor Configuration

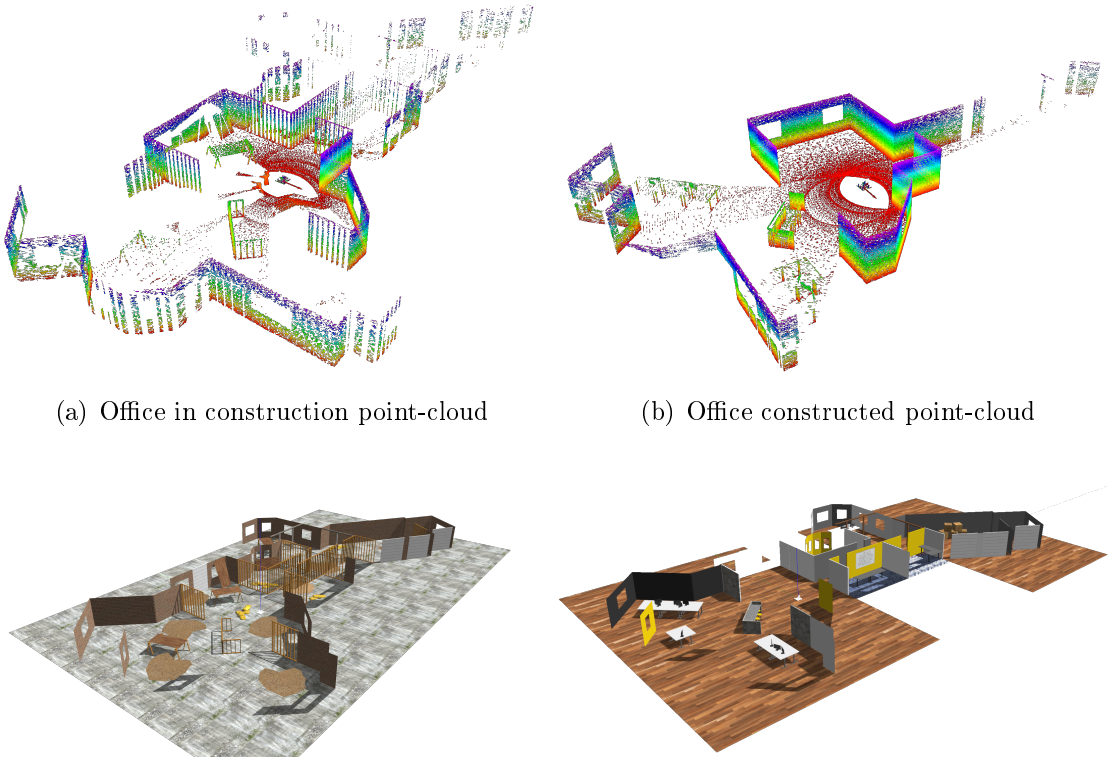


(a) RGB camera view (b) Depth camera view

Figure 33: RGB and depth camera simulation view

### 4.2.4 Simulated world

As the target application of this multi agent exploration is to map a construction environment, a simulated world of an office in construction was created. The world is mainly based on the Gazebo office world by Clearpath Robotics and has two versions : one in construction with walls being built and construction equipment all over, and one where the office is constructed as can be seen in figure 34. The figure also shows the acquired pint cloud from a stationary LIDAR in the office world.



(a) Office in construction point-cloud (b) Office constructed point-cloud

(c) Office in construction gazebo (d) Office constructed

Figure 34: Office world in construction and constructed

To test the planning algorithms, a more complex environment with more ramps, stairs, and difference in level would have been interesting to develop. However, a lack of time meant that the simulation remained simple, opting for prebuilt environment like the one developped for the Falco paper [34]. They provide 5 environments like a

multi-level parking garage, a university campus, a tunnel network, onw with indoor corridors, and a forest.

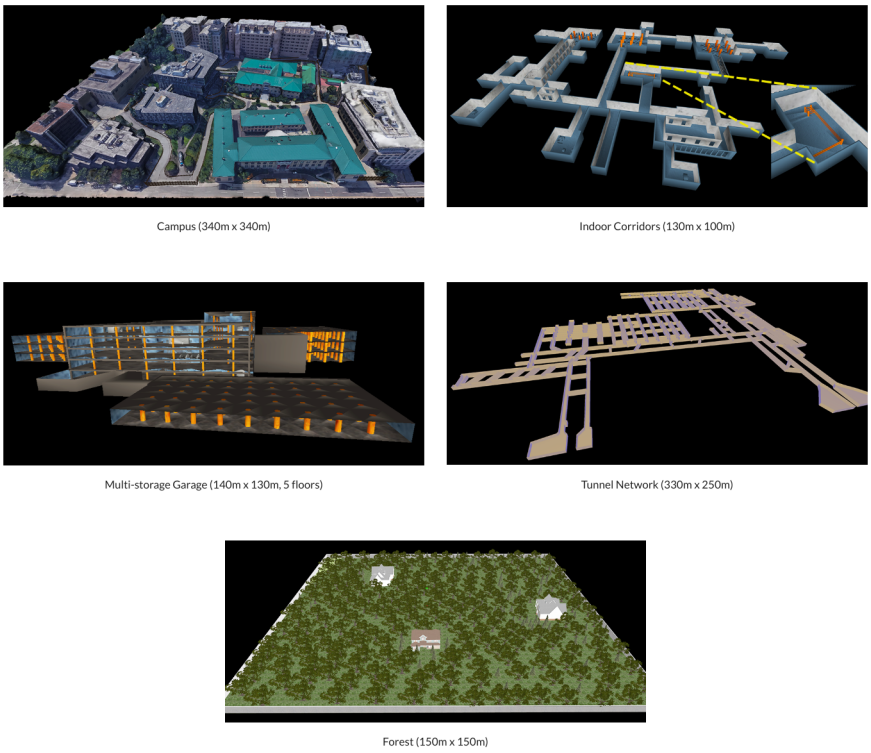



Figure 35: Environments used in the Falco paper

These environments (Figure 35), offer different settings like narrow passages, dead-ends, or large areas to explore. They were subsequently used to test the exploration algorithms like TARE and MTARE.

# 5 Contributions

With this internship, I bring a low-cost exploration platform that can autonomously map and explore unknown environments. The quadruped platform is especially interesting as it uses a dedicated LIDAR for navigation, exploration, and planning, and a second LIDAR for mapping. This relives some effort from the embedded computer that can perform most of the decisions using a low point per second LIDAR, while the second LIDAR requires a very low amount of processing for mapping. Furthermore, the quadruped platform can be used to explore areas that are difficult to reach for wheeled platforms, like stairs, ramps, or uneven terrain. Stairs remain difficult for a platform of this size, but other platform like Boston Dynamics' Spot, Anybotics' Anymal, or Unitree B2 and B2W have shown tremendous capabilities in this area.

Additionally, during this internship several contributions to open source projects were made. Following is a list of commits and pull request made to open source projects during this internship :

| Repository | Type | Description | Status |
|---|---|---|---|
| gazebosim/gz-gui | pr | Added dark mode for drawer and menu buttons | merged |
| gazebosim/docs | pr | Update ros_installation.md as Jazzy is now released | merged |
| gazebosim/gz-msgs | issue | MessageFactory::Types returns duplicate message types | open |
| gazebo-web/gzweb | pr | Add missing primitive model support | merged |
| ros2/ros2_documentation | pr | Changed the description of the clang compile setup | merged |
| ros-navigation/nav2 | pr | precomputeDistanceHeuristic is now computed once | merged |
| ros-control/gz_ros_control | pr | Changed command for gui launch to gz instead of ign | merged |
| FreeCAD/FreeCAD | issue | Start: Thumbnail is not displayed if name of file contains dot | open |
| FreeCAD/FreeCAD | pr | Gui: allow LineEdit to grow with text - fixes #17747 | open |
| FreeCAD/FreeCAD | code review | Assembly: prescribed motions | open |

Table 4: Contributions to open source projects

The reverse engineering efforts of the GO2 battery and the internal firmware that were documented in section 2.4 were also shared with the community. These efforts aim to help in the development of an open source firmware for the GO2.

The work on making the TARE planner work with the Unitree GO2 will also help accelerate the development of exploration algorithms for the quadruped platform, with our solution being a low cost and open source alternative to other solutions based on expensive hardware and proprietary software. The configuration we used for the TARE planner with the GO2 can be found in Listing 4.

# 6 Conclusion

Conclusion

# References

[1] Semih Beycimen, Dmitry Ignatyev, and Argyrios Zolotas. A comprehensive survey of unmanned ground vehicle terrain traversability for unstructured environments and sensor technology insights. *Engineering Science and Technology, an International Journal*, 47:101457, 2023.

[2] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon" next-best-view" planner for 3d exploration. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1462–1468. IEEE, 2016.

[3] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5):1104–1119, 2012.

[4] Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang. Tare: A hierarchical framework for efficiently exploring complex 3d environments. In *Robotics: Science and Systems*, volume 5, page 2, 2021.

[5] Chao Cao, Hongbiao Zhu, Zhongqiang Ren, Howie Choset, and Ji Zhang. Representation granularity enables time-efficient autonomous exploration in large, complex worlds. *Science Robotics*, 8(80):eadf0970, 2023.

[6] Wen Chen, Hongchao Zhao, Qi Shen, Chao Xiong, Shunbo Zhou, and Yun-Hui Liu. Inertial aided 3d lidar slam with hybrid geometric primitives in large-scale environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11566–11572. IEEE, 2021.

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[8] DARPA. Darpa subterranean challenge. `https://www.darpa.mil/program/darpa-subterranean-challenge`. Accessed: 24-11-2024.

[9] Mihir Dharmadhikari, Tung Dang, Lukas Solanka, Johannes Loje, Huan Nguyen, Nikhil Khedekar, and Kostas Alexis. Motion primitives-based path planning for fast and agile exploration using aerial robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 179–185. IEEE, 2020.

[10] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *Aaai/iaai*, 1999(343-349):2–2, 1999.

[11] Patrick Geneva, Kevin Eckenhoff, Yulin Yang, and Guoquan Huang. Lips: Lidar-inertial 3d plane slam. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 123–130. IEEE, 2018.

[12] Dongjiao He, Wei Xu, Nan Chen, Fanze Kong, Chongjian Yuan, and Fu Zhang. Point-lio: Robust high-bandwidth light detection and ranging inertial odometry. *Advanced Intelligent Systems*, 5(7):2200459, 2023.

[13] Dongjae Lee, Minwoo Jung, Wooseong Yang, and Ayoung Kim. Lidar odometry survey: recent advancements and remaining challenges. *Intelligent Service Robotics*, 17(2):95–118, 2024.

[14] Kangcheng Liu. An enhanced lidar-inertial slam system for robotics localization and mapping. *arXiv preprint arXiv:2212.14209*, 2022.

[15] Livox. Livox laser simulation. `https://github.com/Livox-SDK/livox_laser_simulation`. Accessed: 21-09-2024.

[16] Livox and Enway. Livox lidar simulation. `https://github.com/enwaytech/livox_laser_simulation`. Accessed: 21-09-2024.

[17] Steve Macenski, Matthew Booker, and Josh Wallace. Open-source, cost-aware kinematically feasible planning for mobile and surface robotics. *Arxiv*, 2024.

[18] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021.

[19] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[20] NVIDIA. Tensorrt. `https://developer.nvidia.com/tensorrt`. Accessed: 24-11-2024.

[21] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[22] Open Robotics. Ros 2 humble hawksbill installation instructions. `https://docs.ros.org/en/humble/Installation.html`. Accessed: 24-11-2024.

[23] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[24] Habib Ur Rahman, Mudassar Ahmad, Haseeb Ahmad, and Muhammad Asif Habib. Lorawan: state of the art, challenges, protocols and research issues. In *2020 IEEE 23rd International Multitopic Conference (INMIC)*, pages 1–6. IEEE, 2020.

[25] Tomáš Roucek, Martin Pecka, Petr Cızek, Tomáš Petrıcek, Jan Bayer, V Šalansky, Teymur Azayev, Daniel Hert, Matej Petrlık, Tomás Báca, et al. System for multi-robotic exploration of underground environments ctu-cras-norlab in the darpa subterranean challenge. *arXiv preprint arXiv:2110.05911*, 2021.

[26] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[27] stm32f303ret6. Livox laser simulation ro2. `https://github.com/stm32f303ret6/livox_laser_simulation_RO2`. Commit 58ae16b, Accessed: 24-11-2024.

[28] Tamas Foldi. go2-webrtc. `https://github.com/tfoldi/go2-webrtc`. Accessed: 24-11-2024.

[29] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, 2022.

[30] Wei Xu and Fu Zhang. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter. *IEEE Robotics and Automation Letters*, 6(2):3317–3324, 2021.

[31] Jingtian Yan, Xingqiao Lin, Zhongqiang Ren, Shiqi Zhao, Jieqiong Yu, Chao Cao, Peng Yin, Ji Zhang, and Sebastian Scherer. Mui-tare: Cooperative multi-agent exploration with unknown initial position. *IEEE Robotics and Automation Letters*, 8(7):4299–4306, 2023.

[32] Haoyang Ye, Yuying Chen, and Ming Liu. Tightly coupled 3d lidar inertial odometry and mapping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3144–3150. IEEE, 2019.

[33] Peng Yin, Shiqi Zhao, Haowen Lai, Ruohai Ge, Ji Zhang, Howie Choset, and Sebastian Scherer. Automerge: A framework for map assembling and smoothing in city-scale environments. *IEEE Transactions on Robotics*, 2023.

[34] Ji Zhang, Chen Hu, Rushat Gupta Chadha, and Sanjiv Singh. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *Journal of Field Robotics*, 37(8):1300–1313, 2020.

# List of Figures

# Annexe

```
tare_planner_node:
  ros__parameters:
    sub_start_exploration_topic_ : /start_exploration
    sub_terrain_map_topic_ : /terrain_map
    sub_terrain_map_ext_topic_ : /terrain_map_ext
    sub_state_estimation_topic_ : /state_estimation_at_scan
    sub_registered_scan_topic_ : /registered_scan
    sub_coverage_boundary_topic_ : /sensor_coverage_planner/
       coverage_boundary
    sub_viewpoint_boundary_topic_ : /navigation_boundary
    sub_nogo_boundary_topic_ : /sensor_coverage_planner/nogo_boundary
    sub_joystick_topic_ : /joy
    sub_reset_waypoint_topic_ : /reset_waypoint
    pub_exploration_finish_topic_ : exploration_finish
    pub_runtime_breakdown_topic_ : runtime_breakdown
    pub_runtime_topic_ : /runtime
    pub_waypoint_topic_ : /way_point
    pub_momentum_activation_count_topic_ : momentum_activation_count

    kAutoStart : true
    kRushHome : true
    kUseTerrainHeight : true
    kCheckTerrainCollision : true
    kExtendWayPoint : false
    kUseLineOfSightLookAheadPoint : false
    kNoExplorationReturnHome : false
    kExtendWayPointDistanceBig : 1.0
    kExtendWayPointDistanceSmall : 0.5
    kKeyposeCloudDwzFilterLeafSize : 0.2
    kRushHomeDist : 0.05 # 5.0
    kAtHomeDistThreshold : 0.75 # 0.5
    kTerrainCollisionThreshold : 0.5
    kLookAheadDistance : 8.0
    kUseMomentum : false
    kDirectionChangeCounterThr : 6
    kDirectionNoChangeCounterThr : 5
    kResetWaypointJoystickAxesID : 2

    # PlanningEnv
    kUseFrontier : true
    kFrontierClusterTolerance : 2.0 # 1.0
    kFrontierClusterMinSize : 5 #10
    kUseCoverageBoundaryOnFrontier : false
    kUseCoverageBoundaryOnObjectSurface : false

    # Rolling occupancy grid
    rolling_occupancy_grid/resolution_x : 0.2
    rolling_occupancy_grid/resolution_y : 0.2
```

```
    rolling_occupancy_grid/resolution_z : 0.2

    kSurfaceCloudDwzLeafSize : 0.3 # 0.3
    kCollisionCloudDwzLeafSize : 0.2 # 0.2
    kKeyposeCloudStackNum : 5 # 5
    kPointCloudRowNum : 20 # 50
    kPointCloudColNum : 20 # 50
    kPointCloudLevelNum : 30
    kMaxCellPointNum : 100000 # 100000
    kPointCloudCellSize : 5.0 # 18.0
    kPointCloudCellHeight : 1.8 # 1.8
    kPointCloudManagerNeighborCellNum : 5 # 5
    kCoverCloudZSqueezeRatio : 2.0 # 2.0

    # KeyposeGraph
    keypose_graph/kAddNodeMinDist : 1.0
    keypose_graph/kAddNonKeyposeNodeMinDist : 0.5
    keypose_graph/kAddEdgeConnectDistThr : 3.0
    keypose_graph/kAddEdgeToLastKeyposeDistThr : 3.0
    keypose_graph/kAddEdgeVerticalThreshold : 1.0
    keypose_graph/kAddEdgeCollisionCheckResolution : 0.4
    keypose_graph/kAddEdgeCollisionCheckRadius : 0.4
    keypose_graph/kAddEdgeCollisionCheckPointNumThr : 1

    # ViewPointManager
    viewpoint_manager/number_x : 30 # 50
    viewpoint_manager/number_y : 30 # 50
    viewpoint_manager/number_z : 1
    viewpoint_manager/resolution_x : 0.2 # 0.4
    viewpoint_manager/resolution_y : 0.2 # 0.4
    viewpoint_manager/resolution_z : 0.0
    kConnectivityHeightDiffThr : 0.25
    kGreedyViewPointSampleRange : 3 # 3
    kLocalPathOptimizationItrMax : 10 # 10
    kViewPointCollisionMargin : 0.25 # 0.6
    kViewPointCollisionMarginZPlus : 0.5
    kViewPointCollisionMarginZMinus : 0.5
    kCollisionGridZScale : 1.0
    kCollisionGridResolutionX : 0.2
    kCollisionGridResolutionY : 0.2
    kCollisionGridResolutionZ : 0.0
    kCollisionPointThr : 1
    kLineOfSightStopAtNearestObstacle : true
    kViewPointHeightFromTerrain : 0.3 # 0.75
    kViewPointHeightFromTerrainChangeThreshold : 0.2 # 0.6
    kCheckDynamicObstacleCollision : false
    kCollisionFrameCountMax : 3

    kSensorRange : 8.5 # 3.5
    kNeighborRange : 1.5 # 3.0
```

```
    kCoverageOcclusionThr : 0.1
    kCoverageDilationRadius : 0.9

    # Grid World
    kGridWorldXNum : 50 # 121
    kGridWorldYNum : 50 # 121
    kGridWorldZNum : 50 # 121
    kGridWorldCellHeight : 3.0 # 3.0
    kGridWorldNearbyGridNum : 5 # 5
    kMinAddPointNumSmall : 1 # 30
    kMinAddPointNumBig : 1 #60
    kMinAddFrontierPointNum : 1 #20
    kCellExploringToCoveredThr : 1
    kCellCoveredToExploringThr: 10 # 10
    kCellExploringToAlmostCoveredThr: 10 # 10
    kCellAlmostCoveredToExploringThr: 20 # 20
    kCellUnknownToExploringThr: 1

    # Visualization (parameters not working I think)
    kExploringSubspaceMarkerColorGradientAlpha : true
    kExploringSubspaceMarkerColorMaxAlpha : 0.8
    kExploringSubspaceMarkerColorR : 0.0
    kExploringSubspaceMarkerColorG : 1.0
    kExploringSubspaceMarkerColorB : 0.0
    kExploringSubspaceMarkerColorA : 1.0
    kLocalPlanningHorizonMarkerColorR : 0.0
    kLocalPlanningHorizonMarkerColorG : 1.0
    kLocalPlanningHorizonMarkerColorB : 0.0
    kLocalPlanningHorizonMarkerColorA : 1.0
    kLocalPlanningHorizonMarkerWidth : 0.05
    kLocalPlanningHorizonHeight : 3.0
```

Listing 4: TARE planner parameter configuration for Unitree GO2