Final Year Project Report

****

Multi Agent Scene Exploration and Mapping for Continuous Digital Twin Creation

****

BELPOIS Vincent
Under the supervision of Dr. Ivan MUTIS

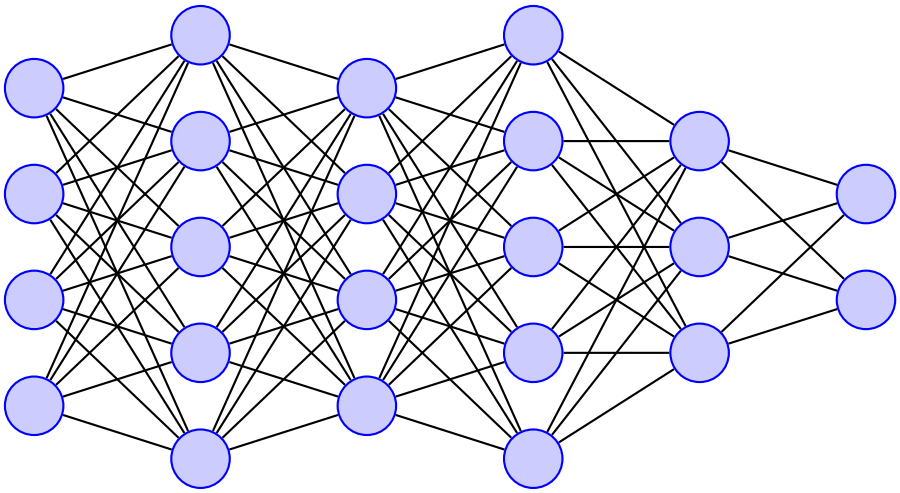
2024

Acknowledgments

Acknowledgments.

Contents

1 Introduction



$$\int_{\alpha}^{\beta} f(x)dx$$

[5]

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 six-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 Six wheeled platform setup

The six-wheeled platform was chosen for its stability and its ability to carry heavy loads. The intent was to have it carry a robotic arm for other projects. Some previous work had already been done on the platform but 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 1). Additionally, a mounting solution for the motor drivers inside the chassis was essential (see Figure 2). These modifications were designed and implemented to ensure proper integration of all components while maintaining the structural integrity of the platform (see Figure 3).

LIDAR MOUNT CAD

Figure 1: Drawing of CAD model of the LIDAR sensor and embedded computer mount

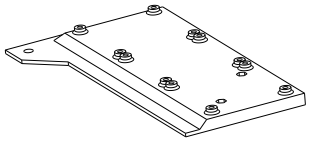


Figure 2: Drawing of CAD model of the motor driver mounting solution

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, 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 4.

The three motor drivers were connected to a microcontroller. The connection can be seen in 2.1.2. I chose to use a Raspberry Pi Pico microcontroller for its many outputs,

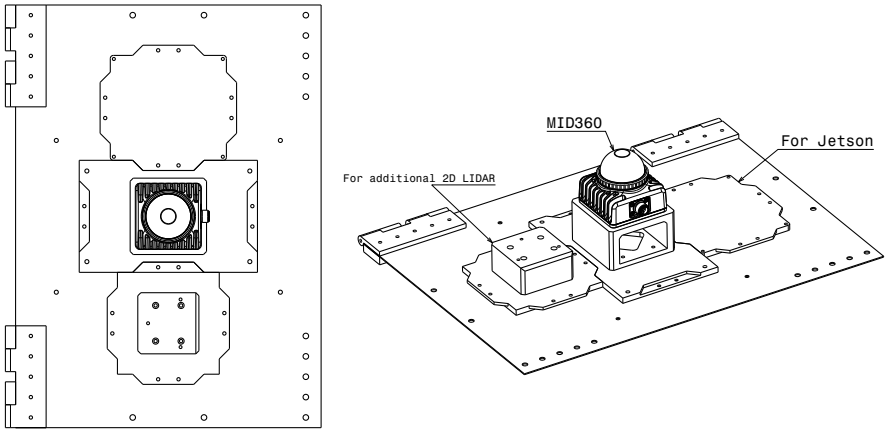


Figure 3: Drawing of CAD model of the modified six-wheeled platform's top

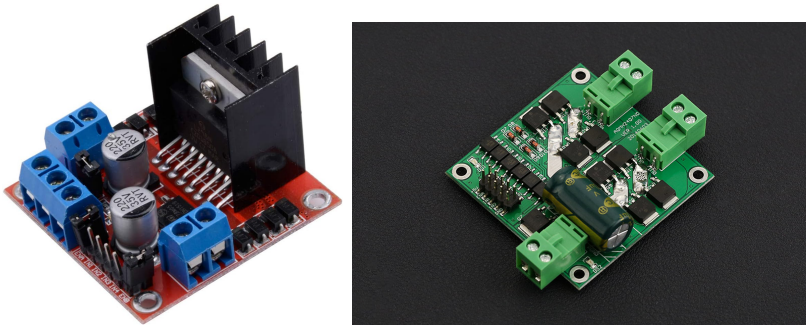


Figure 4: L298N (a) and DFRobot 7A dual DC Motor Driver (b)

totaling 26 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. 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.

Not including the power distribution and regulation system, 5 shows the electrical connections of these components on the modified six-wheeled platform.

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

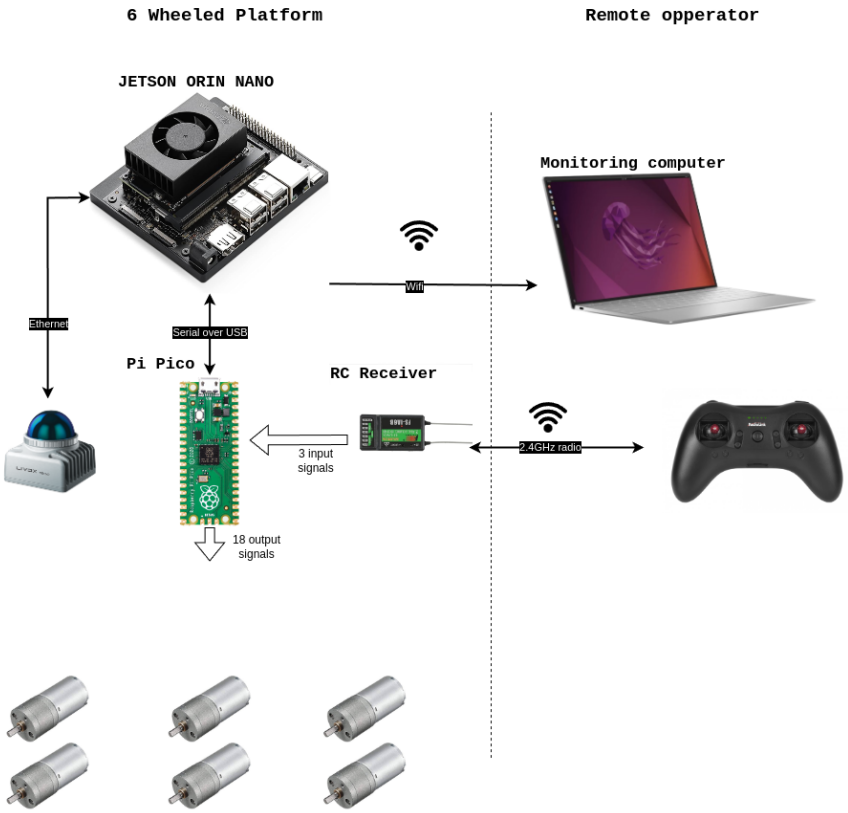


Figure 5: Overall electrical system, excluding power distribution and regulation

Driver to PICO connection diagram

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.

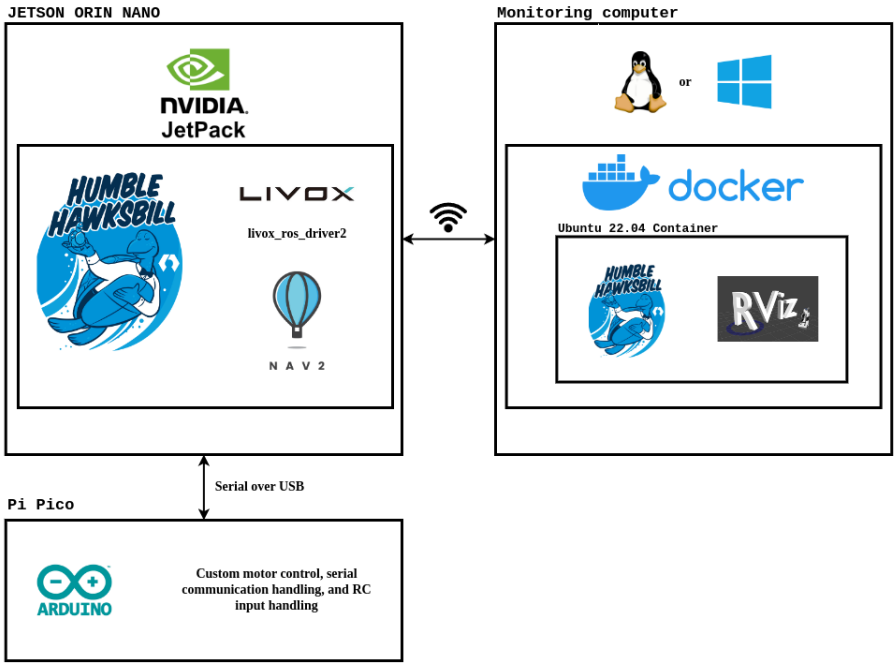


Figure 6: Software architecture

2.1.4 ROS 2 setup

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. I then 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.

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

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

Figure 7: Rover version 1 and 2

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 one that is much simpler and uses Velcro straps as can be seen in Figure 8. This resulted in a saving of XXX grams and thus increased the flight time of the drone.

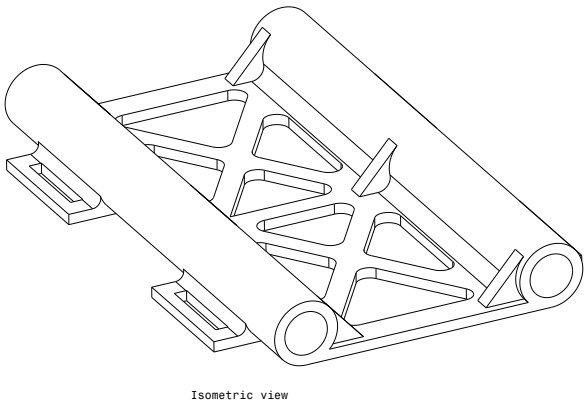


Figure 8: Drawing of CAD model of new battery tray

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

2.2.4 Issues encountered

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 with a connector Amass XT-30 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 as can be seen in the figure 9.

Figure 9: 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 10. 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.

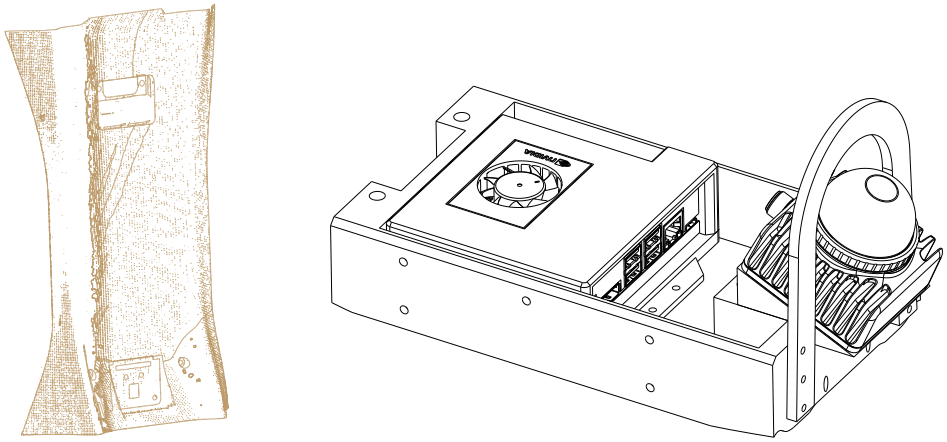


Figure 10: Scanned top of GO2 robot and drawing of CAD model of the compute backpack

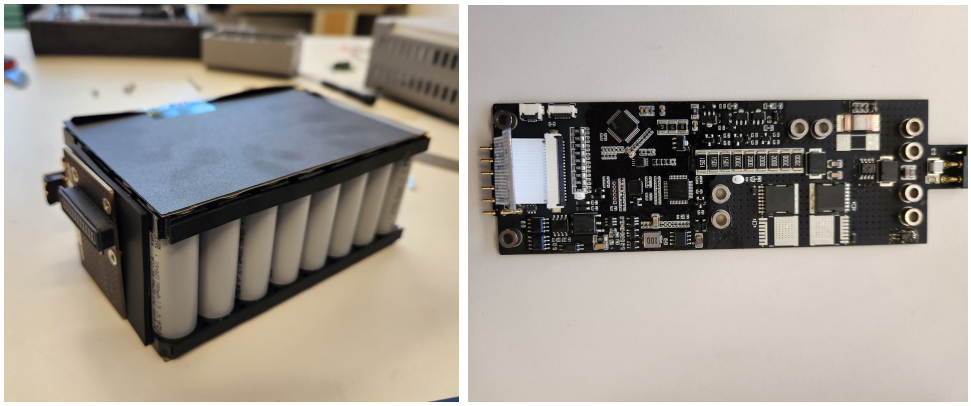
IMAGE OF COMPUTE BACKPACK

2.3.2 Communication with the quadruped

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 11) 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 the robot was jailbroken to allow for more control over the robot.



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

Figure 11: 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. 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 ti unlock every functionality of the robot. The new update brought many new challenges as every binary was not 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

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

`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 the table 2.4.

3 Mapping, planning, and exploration algorithms

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 goal. Exploration uses the two former process as 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 [1], or graph based approaches [4] are used and have not been lately improved. In 3D, the problem is much more complex and the algorithms are much more computationally expensive.

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. Registration 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 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 [5], 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.

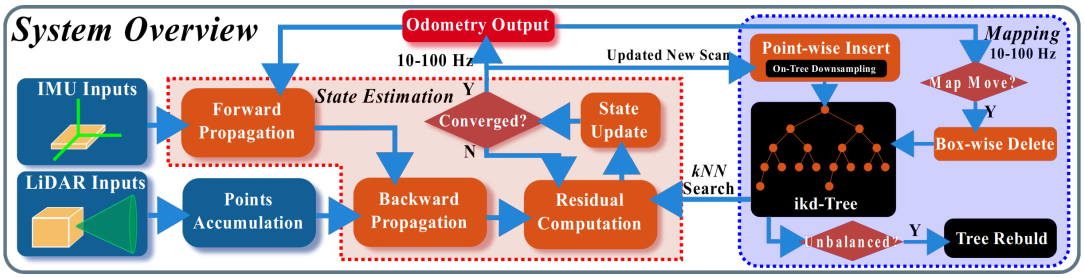
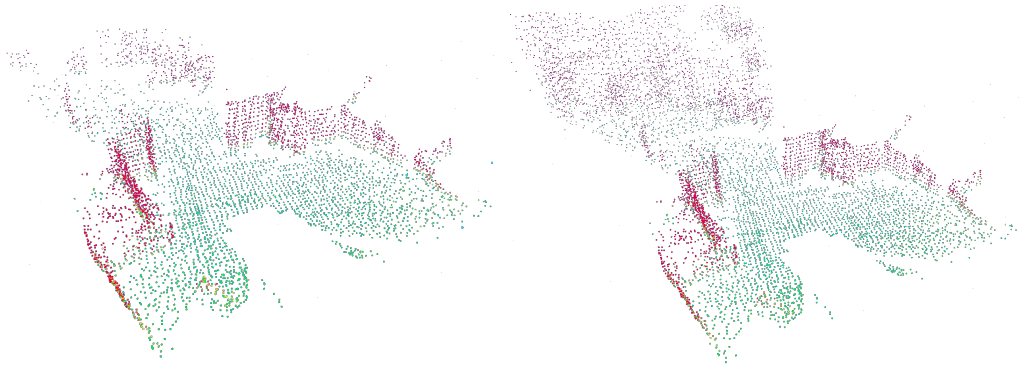


Figure 12: FAST-LIO2 algorithm

3.1.3 Comparison of LIDAR inertial odometry algorithms

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.



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

Figure 14: Comparison of close range and extended range terrain analysis

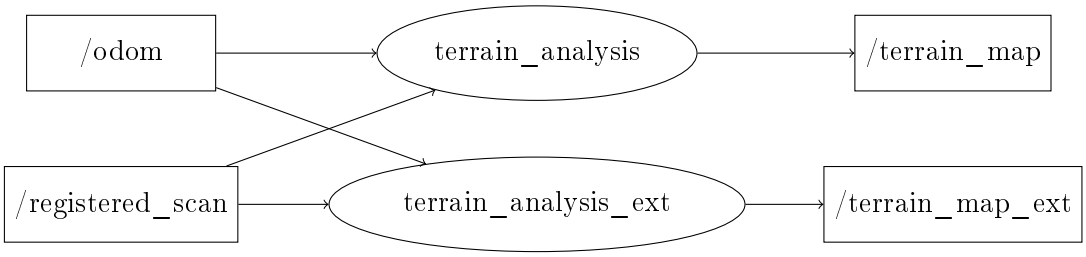


Figure 13: Diagram showing the subscriptions and publications of terrain_analysis and terrain_analysis_ext

3.2 Planning

3.2.1 Path planning

3.2.2 2D path planning

3.2.3 Comparison of 3D planning algorithms

3.3 Exploration

3.3.1 Metrics for exploration

3.3.2 TARE-PLANNER

3.3.3 MTARE-PLANNER

3.3.4 Comparison of exploration algorithms

4 Simulation

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 [3] was a fork of the original simulation package from Livox [2] 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 a RTX GPU to run it.

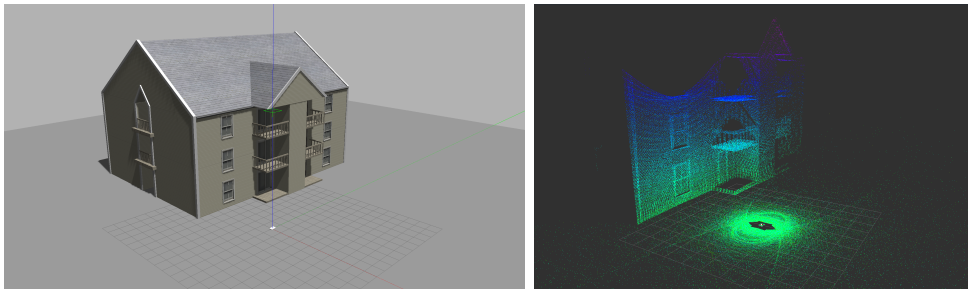
The high overhead, complexity, low portability and the fact that it was not open source made me choose Gazebo over Isaac sim for the simulation of the platforms.

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.

4.2 Simulating sensors

4.2.1 LIDAR



(a) Gazebo simulation environment (b) Point cloud from simulated LIDAR

Figure 15: Simulated LIDAR in Gazebo and point cloud in RVIZ2

INSERT PICTURE OF THE OFFICE WORLD IN CONSTRUCTION AND CONSTRUCTED

Figure 16: Office world in construction and constructed

4.2.2 IMU

4.2.3 Camera

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 wall being build and construction equipment all over, and one where the office is constructed as can be seen in figure 16.

5 Conclusion

Conclusion

References

[1] 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.

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

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

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

[5] 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.

List of Figures

Annexe

Uncomment input annexe when needed