

TECHNICAL UNIVERSITY OF KOŠICE
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS

**AUTONOMOUS CONTROL OF UAV IN GPS DENIED
ENVIRONMENT**
Dissertation thesis

2023

Ing. Stanislav Alexovič

TECHNICAL UNIVERSITY OF KOŠICE
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS

**AUTONOMOUS CONTROL OF UAV IN GPS DENIED
ENVIRONMENT**
Dissertation thesis

Study program: Industrial electrical engineering
Field of study: Electrical engineering
Department: Department of Electrical Engineering and Mechatronics (KEM)
Supervisor: doc. Ing. Milan Lacko, PhD.
Consultant:

2023 Košice

Ing. Stanislav Alexovič

Abstrakt v SJ

Táto dizertačná práca sa zameriava na vývoj 3D mapovacieho systému s využitím bezpilotných lietajúcich prostriedkov (UAV). Práca je rozdelená do štyroch hlavných častí: vytvorenie 3D mapy prostredia pomocou manuálne ovládaného UAV, návrh samo-navigujúceho UAV v známej 3D mape, vývoj UAV schopného plne autonómneho preskúmania a vytvorenie pozemnej stanice pre UAV. Prvá časť zahŕňa manuálne ovládanie UAV a 3D mapovanie pomocou hĺbkovej kamery a palubného počítača. Druhá časť vyžaduje, aby UAV lietalo autonómne s použitím známej 3D mapy, zatiaľ čo tretia časť zahŕňa autonómny let UAV v neznámom prostredí. Posledná časť je zameraná na vývoj počítačovej aplikácie, ktorá komunikuje s autonómnym UAV a poskytuje vizualizačné nástroje zobrazovanie naskenovaných údajov a ovládacie prvky na zasielanie príkazov pre UAV. Práca prezentovaná v tejto dizertačnej práci demonštruje potenciál UAV pri vytváraní presných a efektívnych 3D máp vnútorných prostredí.

Kľúčové slová v SJ

3D skenovanie, 3D mapovanie, UAV, Autonómna navigácia, SLAM, Preskúvanie, Mračno bodov

Abstract

This dissertation focuses on the development of a 3D mapping system using unmanned aerial vehicles (UAVs). The work is divided into four main parts: creating a 3D map of an environment using a manually controlled UAV, designing a self-navigated UAV in a known 3D map, developing a UAV capable of fully autonomous exploration, and creating a ground station for the UAV. The first part involves manual control of the UAV and 3D mapping using a depth camera and mission computer. The second part requires the UAV to fly autonomously using a known 3D map, while the third part involves the UAV's autonomous flight in an unknown environment. The final part is the development of a PC application that interacts with the autonomous UAV and provides visualization tools for scanned data and controls to send orders to the UAV. The work presented in this dissertation demonstrates the potential of UAVs in creating accurate and efficient 3D maps of indoor environments.

Keywords

3D scanning, 3D mapping, UAV, Autonomous navigation, SLAM, Exploration, Point cloud

Statutory declaration

I hereby declare that I have written the whole paper independently using the professional literature listed in the references.

Košice, 02. August 2023

.....

signature

Acknowledgments

I would like to express my gratitude to my wife and daughters, who have tolerated my absence during our shared time. Additionally, I am thankful for my supervisor, who offered support throughout the process of writing this work.

Table of Contents

List of Figures	9
List of Tables.....	11
List of Symbols and Abbreviations	12
Introduction	14
Thesis.....	17
1. Overview of Related Technologies.....	18
1.1. 3D Scanning.....	18
1.1.1. Depth cameras	19
1.1.2. LiDAR	23
1.2. 3D model representation	26
1.2.1. Point cloud	26
1.2.2. 3D mesh.....	27
1.2.3. Voxel-based models	29
1.2.4. Depth map.....	29
1.2.5. RGB-D	31
1.3. 3D engines (simulation).....	32
1.3.1. Gazebo.....	33
1.3.2. RViz.....	33
1.3.3. Unity	34
1.4. Navigation	35
1.4.1. SLAM.....	36
1.4.2. RTAB-Map.....	37
1.4.3. Navigation Mesh	37
1.4.4. Exploration	38
2. Proof of Concept: Handheld 3D scanner for Validating 3D Mapping Features	43
2.1. Requirements for hardware components.....	43
2.2. Sensors	44

2.3.	Battery and Powering of the Internal Components	46
2.4.	Conversion of Battery Voltage to 5V.....	47
2.5.	Battery Protection and Consumption Measurement	48
2.6.	Mission computer	49
2.7.	USB peripherals.....	49
2.8.	GPIO peripherals	49
2.9.	Wireless Communication	50
2.10.	Software	50
2.11.	Testing handheld 3D scanner	51
3.	Designing and Developing an Autonomous Quadrotor Drone for Indoor 3D Scanning	53
3.1.	Drone construction	53
3.1.1.	Frame	53
3.1.2.	Battery.....	54
3.1.3.	Power distribution.....	55
3.1.4.	Motors and ESC.....	56
3.1.5.	Flight controller	57
3.1.6.	Mission computer	58
3.1.7.	Depth and tracking camera.....	60
3.1.8.	Remote control.....	60
3.1.9.	Telemetry radio	61
3.1.10.	Protection cage	62
3.1.11.	Assembling of a drone.....	64
3.1.12.	Power consumption calculation.....	67
3.2.	Mission Computer Software	68
3.2.1.	Simulation	68
3.2.2.	Complete TF tree.....	69
3.2.3.	Transforming points from the depth camera.....	71
3.2.4.	Video streaming	72

3.2.1.	3D reconstruction from the depth image	74
3.2.2.	Localization and navigation	75
3.2.3.	Exploration	77
3.3.	Flight controller software	80
3.3.1.	Communication with mission computer	80
3.3.2.	Communication middleware	81
3.3.3.	Offboard mode	83
3.3.4.	Pose stabilization	84
3.4.	Ground station	85
4.	Results	88
	Conclusion	90
	Benefit for the further development of science and technology	93
	References	94
	List of the author's publications	99
	Appendix	101

List of Figures

Fig. 1 Stereo vision	20
Fig. 2 Laser triangulation	21
Fig. 3 Structured light	22
Fig. 4 Time of Flight (phase-based and a pulse laser)	23
Fig. 5 360-degree LiDAR	24
Fig. 6 Optical phase array used in LiDAR	25
Fig. 7 A 3D point cloud of a scanned building	27
Fig. 8 A 3D mesh of a scanned building.....	28
Fig. 9 Voxelated model of a building	29
Fig. 10 Depth map based on top-bottom view	31
Fig. 11 RGB raster image	32
Fig. 12 Depth channel associated to RGB image.....	32
Fig. 13 Gazebo and RViz	34
Fig. 14 Warehouse simulated in unity game engine	35
Fig. 15 Navigation mesh with all possible paths	38
Fig. 16 Wall following algorithm	39
Fig. 17 Frontier-based exploration.....	39
Fig. 18 Path planning in potential field	41
Fig. 19 Coverage path planning in an unstructured environment.	42
Fig. 20 The 3D handheld camera, left up) depth and tracking camera mounted on front of the handheld camera, left bottom) casing of handheld camera, right) inner components	44
Fig. 21 Intel RealSense depth camera D435i.....	45
Fig. 22 Intel RealSense tracking camera T256.....	46
Fig. 23 Battery drain for idle and scanning mode	47
Fig. 24 Universal BEC which converts 6-25V into 5V/6V	47
Fig. 25 Test of UBEC Voltage stability.....	48
Fig. 26 The measurement circuit Adafruit INA260.....	48
Fig. 27 Wi-Fi local network.....	50
Fig. 28 Example of a 3d scanned environment.	52
Fig. 29 Holybro X500V2 frame with propellers and camera mount [62]	54
Fig. 30 4S LiPo battery with capacity of 4500mAh.	55
Fig. 31 Power distribution board.....	56
Fig. 32 Motors and ESCs	57

Fig. 33 Pixhawk 6X flight controller.....	58
Fig. 34 Raspberry pi 4 model B.....	59
Fig. 35 Measurements of CPU temperature for overclocking purposes.....	60
Fig. 36 Futuba transmitter with receiver and charger.	61
Fig. 37 Pair of telemetry radios.	62
Fig. 38 3D printed parts of protection cage	62
Fig. 39 Design of protection cage in FreeCAD	63
Fig. 40 One fully assembled protection cage.	63
Fig. 41 Block scheme of all components used for a drone.....	65
Fig. 42 Drone with component descriptions.	66
Fig. 43 Basic drone specification.	66
Fig. 44 Drone model in simulated environment.	69
Fig. 45 Part of the TF tree showing relations among transform matrixes	70
Fig. 46 Transforming the pose of the seen object to the <i>base_link</i> frame	72
Fig. 47 Video stream. up) image from RGB sensor, down) image from a depth sensor	73
Fig. 48 Measurement of video stream latency	74
Fig. 49 Point cloud of a room created by RtabMap	75
Fig. 50 Navigation and path following using 2D costmap	76
Fig. 51 Start of exploration. Blue line defines frontier which needs to be explored.	78
Fig. 52 Exploration progress.....	78
Fig. 53 Exploration progress.....	79
Fig. 54 Exploration progress.....	79
Fig. 55 Network configuration file of mission computer	80
Fig. 56 Content of net.cfg file.....	81
Fig. 57 Ping from mission computer to flight controller.	81
Fig. 58 Relation between the microRTPS client and microRTPS agent.....	82
Fig. 59 QGroundControl application with video from a drone	86
Fig. 60 Flying drone	88
Fig. 61 Stream from drone shown in QGroundControl application	89
Fig. 62 A point cloud showing the elevation of an indoor environment.	89

List of Tables

Tab. 1 Usage of lidars depending on wavelengths.....	24
Tab. 2 Intel RealSense D435i specification.....	44
Tab. 3 Intel RealSense T265 specification	45
Tab. 4 Raspberry Pi model B specifications.....	49
Tab. 5 Test report of motor with propeller.....	67

List of Symbols and Abbreviations

BEC	Battery Elimination Circuit
BLDC	Brush Less DC
DC	Direct Current
DWA	Dynamic Window Approach
ESC	Electronic Speed Controller
FPS	Frames Per Second
GPIO	General Purpose Input/Output
GPS	Global Position System
I ² C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
IPC	Inter Process Communication
LiPo	Lithium Polymer
MAV	Micro Aerial Vehicle
MOS	Metal Oxide Semiconductor
MRI	Magnetic Resonance Imaging
OPA	Optical Phased Arrays
OS	Operation System
PCL	Point Cloud Library
PDB	Power Distribution Board
PPM	Pulse Position modulation
PWM	Pulse Width Modulation
RGB-D	Red Green Blue – Depth
ROS	Robot Operation System
ROS2	Robot Operation System version 2

RPC	Remote Procedure Call
RTAB-Map	Real-Time Appearance-Based Mapping
SLAM	Simultaneous Localization And Mapping
SSH	Secure SHell
TCP/IP	Transmission Control Protocol/Internet Protocol
ToF	Time of Flight
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
UBEC	Universal Battery Eliminator Circuit
UDP	User Datagram Protocol
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity (wireless network protocol)

Introduction

Over the past few decades, unmanned aerial vehicles (UAVs), or drones, have become increasingly popular in a variety of applications, such as surveillance, search and rescue, agriculture, and delivery services. These versatile machines have the potential to revolutionize the way we approach many tasks, offering an alternative to manned aircraft and ground vehicles that can be more efficient, cost-effective, and safer.

One of the most important aspects of drone operation is precise and reliable control. In recent years, advances in robotics, computer vision, and machine learning have enabled us to develop sophisticated control systems that allow drones to fly autonomously [1], navigate complex environments [2], and perform a range of tasks with high accuracy and efficiency [3].

However, controlling drones in indoor environments presents a unique set of challenges, such as the absence of GPS and the need for precise localization and stabilization [4]. To address these challenges, researchers have developed various techniques for indoor drone navigation, such as using tracking cameras, depth sensors, and SLAM algorithms [5][6]. These techniques have shown promising results in enabling drones to navigate and map indoor environments, but there is still much room for improvement.

Demand for such technology is rational because a fully autonomous UAV capable of operating in GPS denied environment would open up new opportunities in automation. Some possible use cases would fit into this list.

- Inspection of enclosed or interior constructions – almost all industrial constructions or publicly used buildings, such as bridges require permanent inspection [7].
- Monitor area of interest – instead of installing multiple cameras which will monitor some area, an autonomous UAV could be used for patrolling in an area [8].
- Measure objects or environment properties – depending on attached sensor, an autonomous UAV can measure e.g., temperature, humidity, pressure, etc. of the surrounding environment or just check some specific aspects of a controlled object [9].
- Cargo delivery – area of its use could be extended also for indoor delivery. E.g., inside a warehouse or factory to deliver required components [10].
- Explore an unknown territory – A good exploration algorithm could change autonomous UAV into autonomous explorer which will be able to create 3D maps of previously unknown environments, such as caves or mines. Likewise, it might create 3D scans of building interiors [11].

This work focuses on the development of a drone system for indoor 3D mapping and exploration of unknown environments. The system utilizes a drone equipped with a tracking camera and a depth camera and is capable of generating accurate and detailed 3D maps of indoor environments.

The overview chapter begins with a description of 3D scanning technologies and related sensors, which are needed for sensing the surrounding environment. Possible representations of already scanned objects, which are important for processing and evaluating, are also described here. The next subchapter is dedicated to 3D engines which are mainly used for visualization of scanned objects or in simulations. The chapter ends with a subchapter dedicated to navigation in a previously scanned environment.

The second chapter of this dissertation focuses on the design and development of a handheld 3D scanner for validating 3D mapping features, which serves as a proof of concept for the development of an autonomous quadrotor drone for indoor 3D scanning discussed in the following chapter. In this chapter, the hardware components and sensors required for the handheld 3D scanner are discussed, including the battery and power components, mission computer, USB and GPIO peripherals, and wireless communication. The software used for testing the handheld 3D scanner is also discussed in this chapter.

Chapter three of the dissertation thesis is dedicated to the design and development of an autonomous quadrotor drone for indoor 3D scanning. The chapter starts with the description of the construction of the drone, including the frame, battery, power distribution, motors, flight controller, mission computer, depth, and tracking cameras, remote control, telemetry radio, and protection cage.

Afterward, the software for both the mission computer and flight controller is discussed. The mission computer software includes simulation, a complete transformation frame tree, transforming points from the depth camera, video streaming, 3D reconstruction from the depth image, localization, navigation, and exploration. The flight controller software includes communication with the mission computer, communication middleware, offboard mode, and pose stabilization.

Finally, the chapter concludes with a brief discussion of the ground station used to control the drone during its missions.

Chapter four provides an overview of a specific test flight conducted to highlight the operational capabilities and navigation of the drone in an indoor environment. This chapter delves into the

details of the flight, illustrating how the drone effectively maneuvers and showcasing its control mechanisms. The test flight serves as a valuable demonstration of the drone's performance and its potential applications in real-world scenarios.

The last chapter is dedicated to the conclusion, which includes an evaluation of the completeness of each thesis. It also provides proposals for improvements and outlines potential steps for future work.

Thesis

The main goal of this work is to develop and implement a system for 3D mapping using a UAV, which can operate both autonomously and under manual control. The system should be capable of creating a 3D map of an environment using a depth camera, navigate autonomously using the generated map, and explore previously unknown environments while mapping them.

1. Create a 3D map of an environment using a manually controlled UAV.

The UAV will be fully controlled by the pilot, its movement dependent on transmitted commands from the joystick.

3D mapping is done by the attached depth camera and mission computer processing camera outputs and reconstructing the scanned environment.

2. Self-navigated UAV in a known 3D map.

The UAV should be able to fly autonomously from its current position to a position defined by the pilot.

It must be able to determine its current position in the previously scanned map.

Navigation is done by scanning an environment and matching it to the known map which will result in a position related to the known map.

3. UAV capable of fully autonomous exploration.

UAV is capable of autonomous flying in a previously unknown environment with the goal to map all unknown and reachable parts of this environment.

4. Create a ground station for the UAV.

A PC application for interaction with the autonomous UAV.

It should provide tools for visualizing the data scanned from the UAV, as well as controls for sending orders to the UAV.

1. Overview of Related Technologies

1.1. 3D Scanning

3D scanning refers to technology which accurately transfers information about a real-world object into a computer environment, building a digital copy with the same properties as its real-world counterpart primarily its shape and dimensions. In scanning, a digital copy of a scanned object is referred to as a model [12].

This technology opens up a broad range of possible use cases. Once a 3D model is obtained, there are various purposes for which it may be used. It gets even more interesting with objects not easily observable in the real world. Thus, 3D scanning might be applicable in the contexts below, doing the following:

- Measure properties, area, volume, distance
- Examine surfaces for possible defects
- Design new objects which will be created and later inserted in a pre-existing environment
- Archive and compare models
- Share models over network

Nowadays, 3D scanning is often used in the areas, such as:

Medicine: Here, mainly scanners which are able to penetrate the outer layers of the human body to inspect its inner parts are needed. Now, it is a common practice to perform e.g., Magnetic Resonance Imaging (MRI) in order to examine a patient's internal organs and make their diagnosis or use ultrasound scanners as a monitoring tool for fetus development during pregnancy. [13][14]. On top of that, taking scans of visible body parts found usage in this field as well. For example, mouth 3D scanners are used by dentists to create teeth and jawbone models. With such models, practitioners can prepare more precise dental implants or dentures for their patients [15].

Architecture: In architecture, scanners are used in the opposite process, where an architect designs a building which is then built by construction workers. In such cases, 3D scanning could be very useful, as buildings often need to be placed into pre-existing environments with various levels of complexity. Thus, the more complicated it would be, the more useful the 3D model would become. This is especially true of building refurbishments. If 3D scanning was cheap and could be performed often during the building phase, it might be used to track progress on the building or to validate any structural changes made to the initial blueprint [16].

Industry: Automation is an important part of industry which is still growing. This growth comes with the help of newer sensors and increasing computer power. The same holds true for scanning and digitization. If a manufactured part can be scanned and its virtual model created, a computer program can determine what should be performed next, like scanning manufactured parts for defects or grasping and manipulating with unordered parts in bins [17][18].

Robotics: Robotics is based on some kind of 2D or 3D scanning already, but instead of creating a digital model of a real-world object, robots create a 3D model of the environment surrounding them. Thus, if a robot can scan the environment, it can interact with it [19][20].

Entertainment: Lots of games have their own virtual world, where players can do things, which are normally considered impossible in the real world. Here, 3D scanning is also used mainly for transferring real-world objects into the virtual world. It helps game designers with creating models, later used in games. If game developers did not make use of 3D scanning, they would have to create the whole game world from scratch [12].

The same goes with movies, where scans of real-world objects can be enhanced with additional features on screen during post-production.

1.1.1. Depth cameras

Standard digital cameras are able to capture 2D images of the environment, which lies in front of the lens. This is possible because of semiconductor image sensors based on metal oxide semiconductor (MOS) technology. Depth cameras can also capture images in this way, but with additional information, like the depth of each pixel in the image. Depth can be defined as a distance between the camera and a captured object. Depth information could be obtained using various methods [21].

1.1.1.1. Stereo vision

This is a principle which is used by humans. Human beings are able to perceive depth of field and distance of an object in it thanks to binocular vision. This method is also used in technology with the help of two cameras scanning 2D images of the target, positioned in such a way that they approach the object from two different angles. Distance between the cameras is referred to as a baseline. With the help of a specialized software, disparities of the two plane images are compared and through triangulation, a full 3D point cloud created. This method is fast, it can cover a large field of view and relies on ambient light. Therefore, there is no need for another source of light, making stereo vision a good method for large distance scanning, as well as in outdoor settings [22].

Despite the advantages described above, stereo vision is not comparable with human vision when it comes to quality of the image. The main problem lies in finding the correspondence between pixels in the scanned 2D images. On one hand, if one of them has, say, a lot of irregularities or rich texture, corresponding pixel values can be easily detected. On the other hand, however, matching pixel values becomes more problematic with featureless regions in one or both of the scanned pictures. As a solution to this shortcoming, a projector might be used to help reduce, or in ideal cases, eliminate the inability of the cameras to scan featureless regions on the images and pinpoint individual pixels in the resulting pictures by introducing some irregularities in form of a random pattern and projecting them on the surface of the regions of the image(s). The pattern would be perceived by cameras, creating active stereo vision [23].

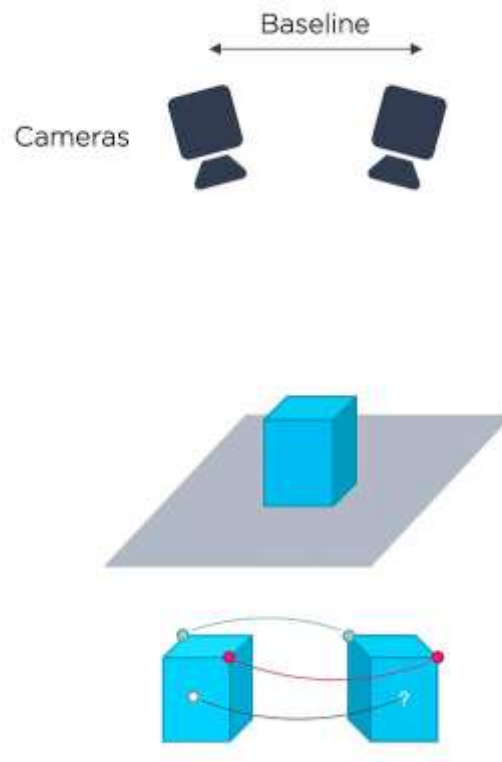


Fig. 1 Stereo vision

1.1.1.2. Laser triangulation

Laser triangulation is an old, simple, cheap, and highly accurate scanning technique. It works with a line laser and a camera. The laser is projected on a scanned object and the camera records the laser reflected from the scanned object back to its lens. As the object is being scanned, the line is deformed by the objects' surface and by comparing it to a reference point, which is the straight line, non-deformed deviations are measured. The data are then used to create a 3D point cloud.

The laser can only scan one line at a time. As a result, the object is scanned gradually, turning scanning into a time-consuming, multiple-step process, and a major drawback of this scanning method. Not only that, this setup does not provide any color information and can struggle to acquire data from shiny and dark object surfaces. As for speeding the process up, one solution would be to either move the laser, or the scanned object. The other disadvantages become less evident when looking at the performance of laser triangulation in ambient light settings, where it is both very robust and accurate. Despite its limitations, this technology has found its use mainly in scanning objects moving on a conveyor belt [24].

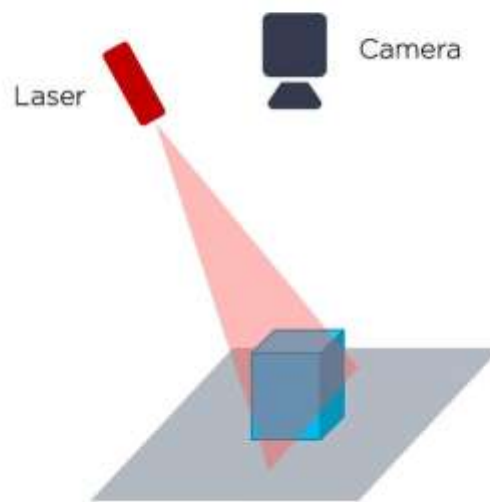


Fig. 2 Laser triangulation

1.1.1.3. Structured light

Structured light is like laser triangulation, with the main difference lying in the fact that structured light refers to scanning the whole field. Unlike laser triangulation, it can provide a full 3D image of an object and not just one cross-sectional line. The camera records distortions of a known light pattern projected onto a scanned object. Again, this distortion and succeeding triangulation resolves in a full 3D point cloud. It is similar to active stereo vision, except with only one camera and the projected pattern, which, in this case, is not random but structured. This method has the same disadvantages as laser triangulation, where reflective or very dark surfaces could produce scanning errors [25][26].

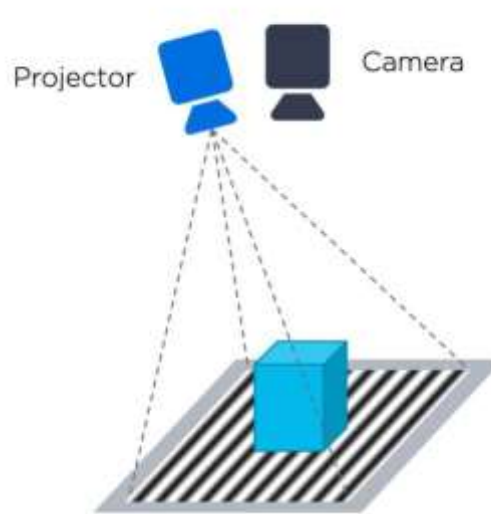


Fig. 3 Structured light

1.1.1.4. Time of Flight

The name of this method implies that a time domain approach is used. A camera does not measure spatial distortions but time delay between the light being emitted from the source and reflected from the surface of the scanned object. Since the speed of light is constant, the camera can very precisely measure the distance according to the time delay [22].

This relatively new technology has some advantages over other scanning methods. It is very fast, scans the whole image at once, and performs well in dark environments. All this is possible due to a special sensor which can measure distance of each scanned pixel independently. On the other hand, its resolution is relatively low. This method includes two approaches to gathering data.

Pulse laser light system: it measures the time delay between the emitted laser pulse and the received pulse, and because the time is directly proportional to the distance travelled, a relative position is calculated.

Phase-based system: it works by modulating a sine wave on the emitted laser beam and measuring the phase difference between the wave emitted by the device and the wave reflected from the object. Integrating it over a more extended time compared to the pulse-based system typically provides better precision. The downside is that the shorter the measurement range, the higher the sensitivity to ambient lighting such as sunlight and reflections. Because the light may reach the object along several paths, the measured distance may be longer than the actual distance [27][28][29].

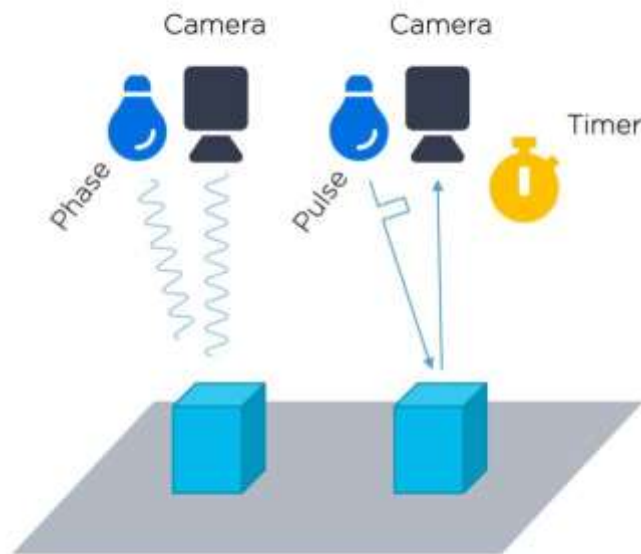


Fig. 4 Time of Flight (phase-based and a pulse laser)

1.1.2. LiDAR

LiDAR is an abbreviation for Light Detection And Ranging. This device is able to measure distance by measuring time which elapses from transmission to reflection from a distant object back to the sensor. Again, with the speed of light being constant, time delay is proportional to distance [30].

In LiDARs, laser works as the source of light. It is pointed at an object, measuring its distance, but because the laser beam can only illuminate one point in the distance at a time, it needs to move to measure the distance of other points on the object as well. This can be achieved either by using a static laser, sensor and a rotating mirror or by rotating the laser with the sensor. If the laser beam is not limited by the construction of lidar, its observation field can be as high as 360 degrees. Laser with such construction is called a 2D lidar and it can retrieve 2D image of the surrounding environment on one plane. An important parameter of this 2D scan is the angular resolution, which defines the angle between particular scanned points.

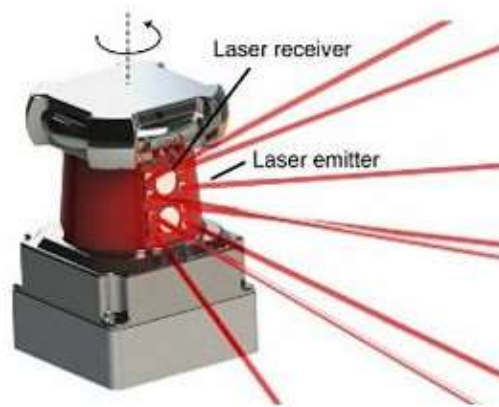


Fig. 5 360-degree LiDAR

To add the 3rd dimension, either lidar or its mirror need to rotate on two axes. There is also an option to add other laser beams directed at different angles, creating what is called a vertical field of view. This produces a 3D image of the scanned environment, but because lidar can obtain only one point at a moment, it needs to spin fast enough to get a full image in reasonable time (from 1Hz to 100 Hz).

The wavelength of the laser is an important parameter of any LiDAR. Different wavelength ranges are used depending on usage, as shown below:

Wavelength [nm]	usage
1500 –2000 (infrared)	meteorology/Doppler LiDAR – Scientific uses
850 -940 (near-infrared)	terrestrial mapping
500 –750 (blue-red)	bathymetry
250 (ultraviolet)	meteorology

Tab. 1 Usage of lidars depending on wavelengths

The most commonly used wavelength is near-infrared with power falling within Class 1 laser. It can be used for outdoor applications, where it performs well in rainy, snowy, or foggy conditions.

The biggest disadvantage of LiDAR is that if it is to provide good accuracy and reliability, it contains moving parts which increase its price. This creates big demand for cheap LiDAR that would find its use mainly in automotive industry, robotics and other specialized fields.

Recently, solid-state lidar has been developed. It does not contain moving parts, which makes it more affordable so it can be mass-produced and used in a wide area of applications. As for the mechanism, solid-state lidar is no different from a standard lidar. It measures the time between light from the moment it is emitted from the source until it reflects back to the sensor (ToF).

However, what is different is the light emitter. Instead of using a laser, solid-state lidar uses Optical phased arrays technology (OPA). Similar technology is already well-established in radio and radar technologies. It uses an array of antennas which can change the direction of the output signal by interference and phasing of each antenna signal. The same goes for optics. OPA can produce directed laser beam only with precise phasing of each small light emitter. By controlling light interference, this technology could steer the laser beam and point it to anywhere in the field of view. The receiver then captures reflected light and because the transmitter and emitter are synchronized, the embedded electronics can produce a final 3D image of the scanned area [31].

Optical phased array is a semiconductor component manufactured as any other chip [32]. Therefore, its price could be drastically reduced by mass production, possibly lowering lidar price as a result. Above all else, solid-state lidar has a major advantage in prolonging the lifespan of the device. Whereas, in common lidars, the lifespan is somewhere around 2000 hours, a device with a solid-state LiDAR can last up to 100,000 hours.

With affordable LiDAR technology, it is expected that it will be used in more and more industries. It is already in great demand in automotive industry [30], where it is used in self-driving cars, although there are many other areas of its possible application. Among them, robotics, object detection, tracking, collision avoidance, and more.

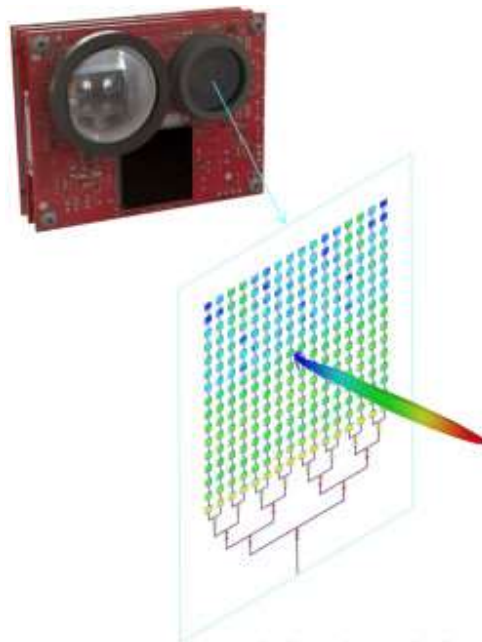


Fig. 6 Optical phase array used in LiDAR

1.2. 3D model representation

In the previous chapter, methods used for 3D scanning were described. After a scan has been taken, the data are further processed by transforming them into a 3D model representation, which fits specific areas of application [33][34].

In general, 3D models can be divided into two categories:

Solid: These models define the volume of the object they represent. Solid models are mostly used in engineering and medical simulations and are usually built with Constructive Solid Geometry or voxel assemblies.

Shell/boundary: These models represent the surface of an object or entity. The boundary of the object is a bit like an eggshell and forms the object's infinitesimally thin shell. Almost all visual models used in games and films are shell models, with surface properties applied.

1.2.1. Point cloud

A point cloud could be considered raw data from a 3D scanner. It is a set of data points in a three-dimensional coordinate system. These points are spatially defined by coordinates on X, Y, Z axes and often represent the envelope of an object. Quality and completeness of the model depend on specific attributes of the 3D scanner [35][36].

3D point clouds can be used in some areas directly without further transformation into a different model representation, as it provides some benefits over other types of representations.

Pros:

- Fast rendering
- Exact representation
- Fast transformations

Cons:

- Numerous points (obj. curve, exact representation)
- High memory consumption
- Limited combination operations

Operations:

- Transformations: points in the point list with linear transformation matrices may be multiplied.
- Combinations: "Objects" can be combined by merging several point lists together.

- Rendering: Projects and draws the points onto an image plane



Fig. 7 A 3D point cloud of a scanned building

While fast rendering and transformations make a direct inspection of a point cloud handy, they are often not directly integrated into commonly used three-dimensional applications. A common process is to derive a mesh using a suitable surface reconstruction method. There are several ways of transforming a point cloud into a three-dimensional explicit surface, some of which are mentioned below.

1.2.2. 3D mesh

A mesh is a geometric data structure that allows surface subdivisions to be represented by a set of polygons. Meshes are used particularly in computer graphics, to represent surfaces, or in modeling, to discretize a continuous or implicit surface. A mesh is made up of vertices (vertex), connected by edges making faces (facets) of a polygonal shape. When all faces are triangles, it is called triangular meshing. [37].

Quadrilateral meshes are also very interesting but often obtained through mesh optimization methods to get more compact representations. It is also possible to use volumetric meshes, which connect the vertices by tetrahedrons, hexahedrons (cuboids), and prisms. These so-called meshes are based on the boundary representation, which depends on the wire-frame model (The object is simplified by 3D lines, each edge of the object is represented by a line in the model).

Pros:

- Well-adopted representation

- Model generation via “new-gen scanning”
- Transformations are quick and easy

Cons:

- High memory requirements
- Expensive combinations
- Curved objects are approximated

Operations:

- Transformations: All points are transformed as with the wire-frame model (Multiply the points in the point list with linear matrices). Besides, the surface equations or normal vectors can be transformed.
- Combinations: Objects can be combined by grouping point lists and edges to each other; Operations on polygons (Divide based on intersections, Remove the redundant polygons, combine them together).
- Rendering: Hidden surface or line algorithms can be used because the surfaces of the objects are known so that visibility can be calculated.

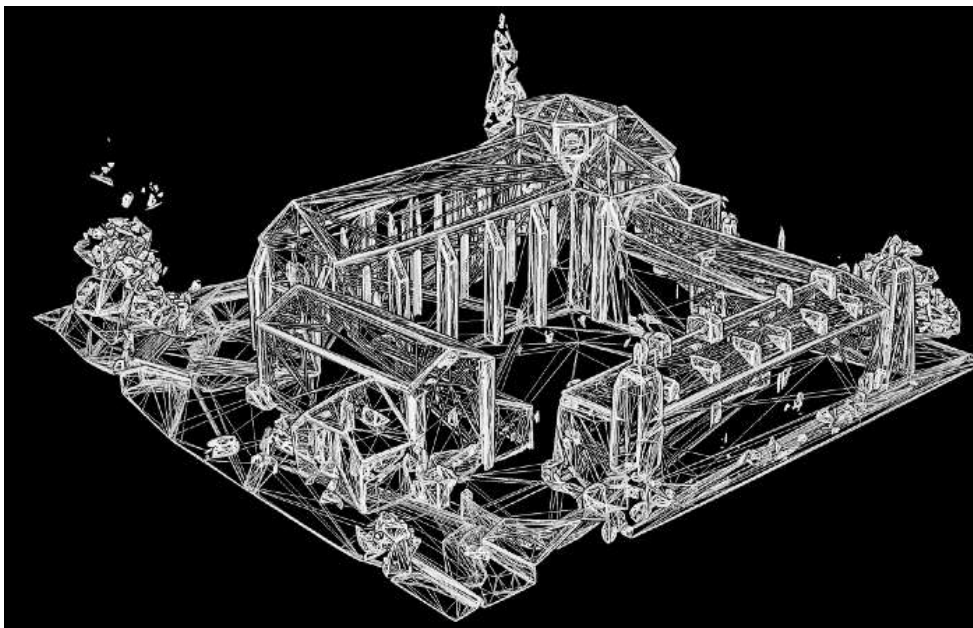


Fig. 8 A 3D mesh of a scanned building

Meshes are a great way to explicit the geometry of a point cloud, and often permit to greatly reduce the number of needed points as vertices. On top of it, they facilitate and help understand the sense of the relationships between objects through the facet connectivity. However, meshing is an interpolation of the base point cloud geometry and can only represent the data to a certain

degree, linked to the complexity of the mesh. There is a multitude of strategies to best mesh a point cloud, but this often requires having some theoretical background and knowing which parameters to adjust for optimal results.

1.2.3. Voxel-based models

A voxel can be seen as a 3D base cubical unit that can be used to represent 3D models. Its 2D analogy is a pixel, the smallest raster unit. As such, a voxel-based model is a discretized assembly of “3D pixels” and is most often associated with solid modeling [38].

In the case of point cloud data, each point can be represented as a voxel of size x , to get a “filled” view of empty spaces between points. It is mostly associated with data structures such as octrees and it permits to average a certain number of points per voxel unit depending on the level of refinement needed.

While this is practical for rendering and smooth visualization, it approximates the initial geometry coupled with aliasing artifacts and can thus give false information if the volume information is used improperly. However, due to very structured grid layouts of voxel models, they can be very handy for processing tasks such as classification through 3D convolutional neural networks.

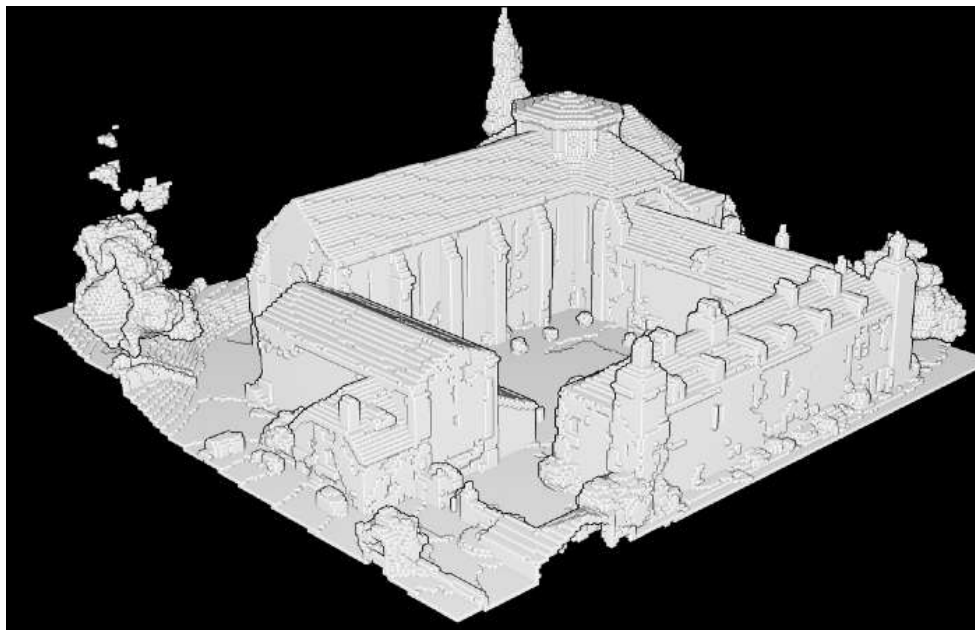


Fig. 9 Voxelated model of a building

1.2.4. Depth map

This representation belongs to the category of raster-based point cloud representations. A depth map is an image or an “image channel” that contains information related to the distance of the points constituting the scene from a single viewpoint. While RGB images are used the most

commonly, the simplest form of expressing the depth is to color-code on one channel, with intensity values. Bright pixels then have the highest values and dark pixels have the lowest values. And that is it. A depth image just presents values according to how far the objects are, where pixels color gives the distance from the camera [39].

This form of point cloud representation is adequate for surface information linked to a known viewpoint, for instance, in autonomous driving scenarios where one can very quickly map the environment at each position through a 360 projected depth map. However, the major drawback of this method is that rather than 3D data, 2.5D data are used, simply because a single line of sight cannot be represented by 2 different values.

Pros:

- Low memory requirements
- Very well-known raster format
- Transformations are quick and easy

Cons:

- Essentially a 2.5D representation
- It is unsuitable for a full 3D scene on its own
- Weak topology

Operations:

- Transformations: Multiply the pixels in the image with linear transformation matrices
- Combinations: Objects can be combined by merging the point lists.
- Rendering: Draws pixels on the image plane

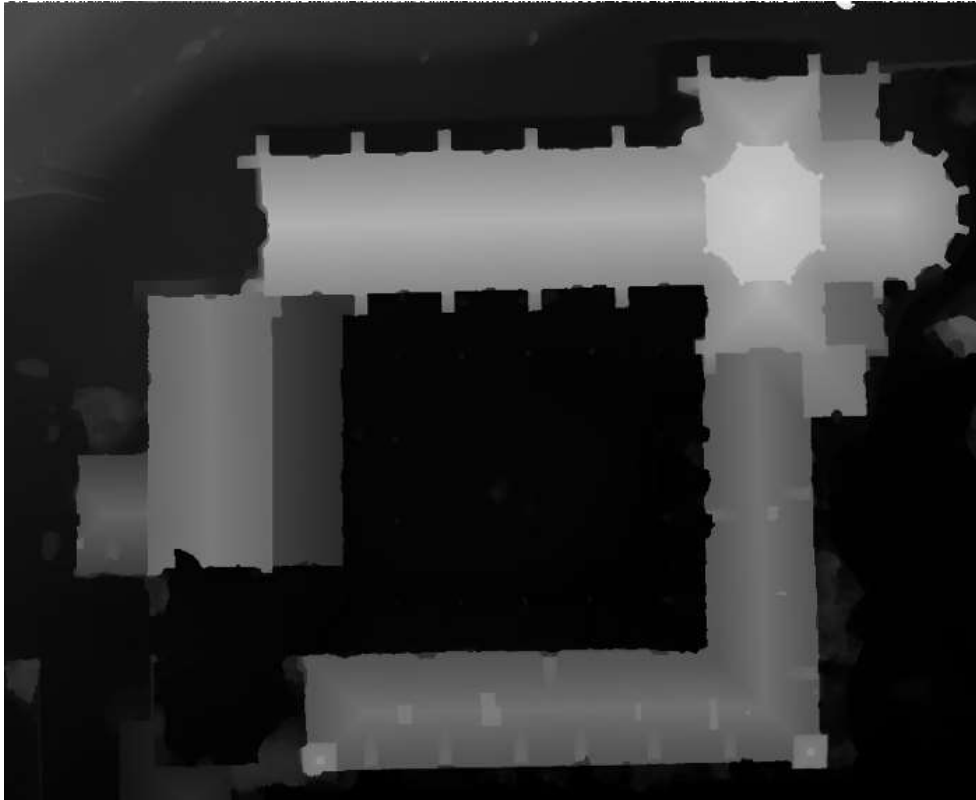


Fig. 10 Depth map based on top-bottom view

1.2.5. RGB-D

RGB-D is a special case of a depth map. It has become popular in recent years thanks to the popularity of RGB-D sensors. RGB-D data provide a 2.5D information about the captured 3D object by attaching the depth map along with 2D color information (RGB) [40].

Besides being inexpensive, RGB-D data are simple, yet effective representations for 3D objects that can be used for different tasks, such as identity recognition, pose regression, and correspondence. The number of available RGB-D datasets is huge compared to other 3D datasets like point clouds or 3D meshes and, as such, it is the preferred way of training deep learning models through extensive training datasets.



Fig. 11 RGB raster image



Fig. 12 Depth channel associated to RGB image

1.3. 3D engines (simulation)

At first glance, “3D engine” usually means a game engine which is used for game development. In simple terms, 3D engine is the core of the game. It provides a “world” for a game with its own laws. 3D engines, besides other functionalities, provide features for manipulation with 3D objects which could be created from scratch in some 3D object creation tool or captured from reality by a 3D scanner. The main reason why this matter is because 3D engines are also widely used in robotics, where they can show what is seen by a 3D scanner in real time and add some additional features to the projection. This is becoming ever more popular, as Augmented Reality is increasing in its dominance, and covers the same field.

3D engines are also used for simulations, providing an invaluable tool for further advancements in robotics. It is possible mainly because robots, which are the primary focus of robotics with their two main features –perception of the environment and interaction with the environment – which could be based to a large extent on a 3D engine that would create a virtual environment. The

major difference is only that a real robot depends on its own sensors. If these could be simulated inside a virtual environment, all logic behind sensors could be the same for both real and virtual robots, making testing much more feasible, cheaper and safer, as often no real hardware would be needed.

The main advantages of simulations are:

Repeatability: Some scenarios could be exactly repeated, and results of robot behavior validated

Scalability: attributes of the virtual world can be altered or changed. E.g., speed of time, preparation of rare situations to test edge cases, etc.

Visualization: robot attributes can be visualized e.g., the path of the robot, battery state of charge, robot's velocity, etc.

In the next parts of this chapter, some of the simulation tools used in the field of robotics will be described [41].

1.3.1. Gazebo

Gazebo is a popular simulation tool based on the OGRE 3D engine [42]. It provides realistic rendering of environments, including high-quality lighting, shadows, and textures [43]. Other features include:

Physics engine: High-performance physics engine

Sensors: Generate sensor data with or without noise from laser range finders, 2D/3D cameras

Plugins: Possibility to create custom plugins or use third party plugins

Robot models: Many robot models are included in the standard package

TCP/IP transport: Possibility to run gazebo on a remote server and communicate with it through message passing protocol.

1.3.2. RViz

RViz is not as much a simulation tool as it is a visualization tool [44]. It means it is used for visualization of various data which are used by robots. It belongs to the ROS framework which will be described in the following sections. Like Gazebo, RViz is also based on the OGRE 3D engine. The main advantage of RViz is that it can visualize standardized types which are used in robotics e.g., Point cloud, camera, Math, Pose, TF, etc.

It can be used with simulated or real-world robots and since its only job is to visualize the input data, it does not matter what the source is.

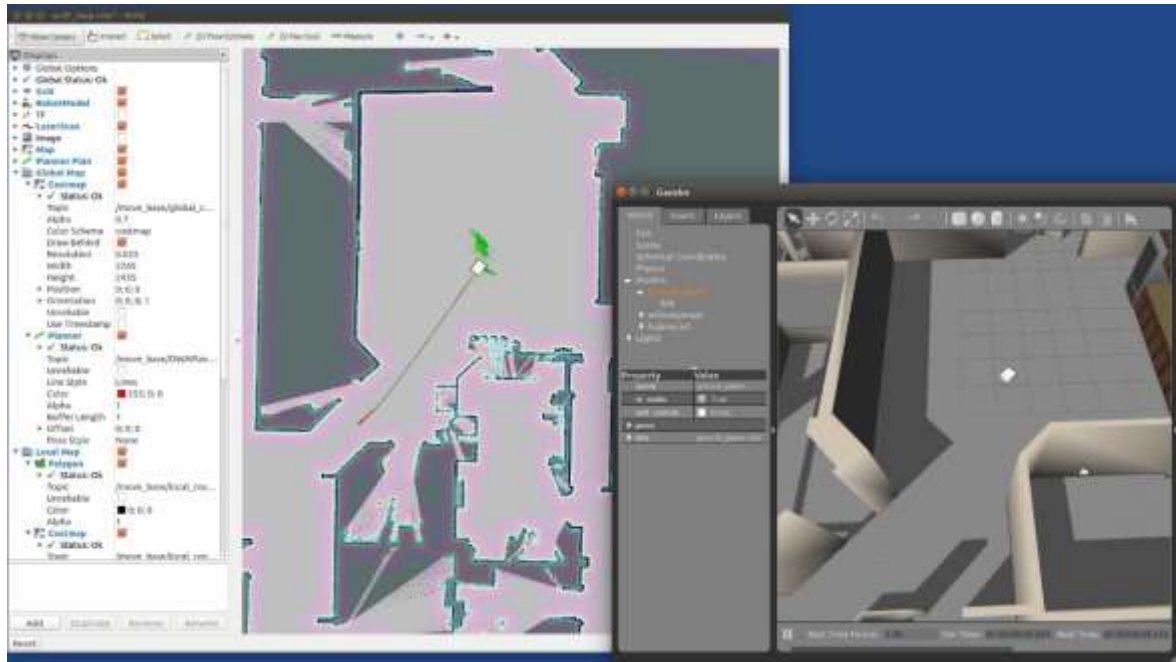


Fig. 13 Gazebo and RViz

1.3.3. Unity

Even though Unity is a relatively new tool in this area, it has already established itself in being the most popular game engine in the world. Creators of Unity decided to enrich the robotics field with the possibilities of Unity. This comes as a no surprise because Unity offers interesting features. Beside the standard simulation tool for a virtual world where you can test navigation, collision avoidance, and interaction with other objects, Unity offers Artificial Intelligence and machine learning for robotics applications. To top it all, it includes cutting edge graphics [45].

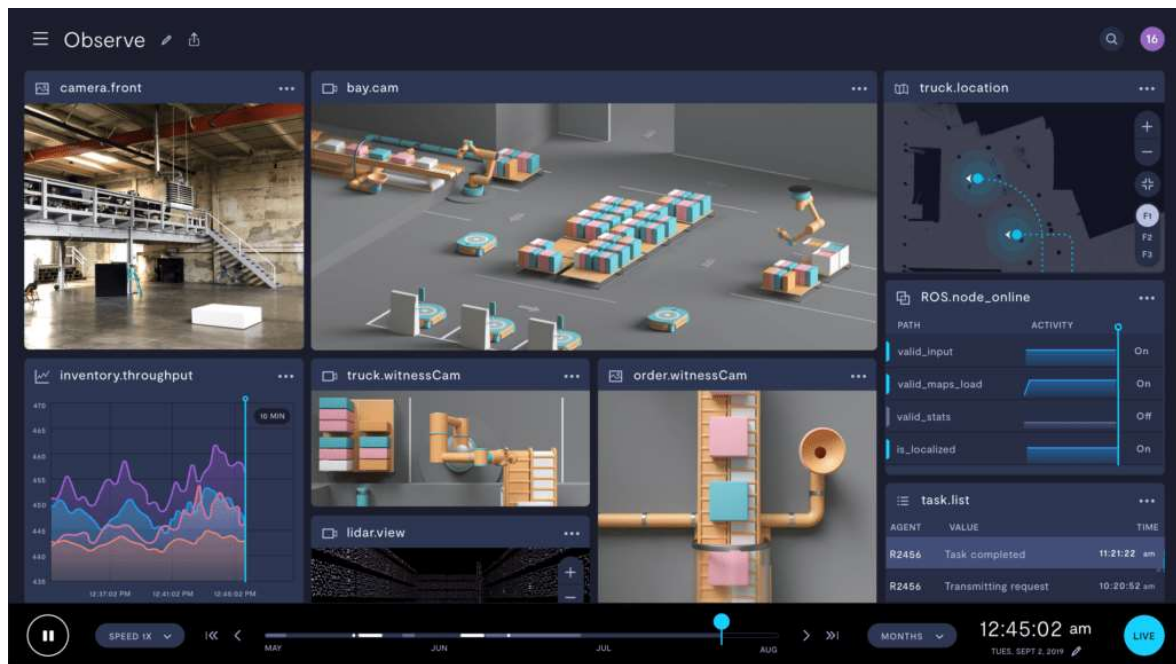


Fig. 14 Warehouse simulated in unity game engine

1.4. Navigation

Navigation is the process of looking for a path from point A to point B, while at the same time tracking the current position. The ability of a navigating device to track its current position is an essential requirement of navigation and there are multiple ways of achieving it. Probably the simplest solution is to use GPS, however, there is one catch. GPS navigation is accessible only in outdoor applications. Indoor applications must rely on another approach.

There is an option to use a known position of a signal beacon [46] like Wi-Fi access points. If a robot is able to receive signals from at least three beacons, then relative position to these beacons could be determined. This process is called trilateration. For trilateration to work, the robot must be able to determine the beacon's linear distance. This might be done measuring time which is required for signals to travel from the beacon to the robot, with distance defining the perimeter of possible positions. If a robot is able to collect three of these distances, then position is the intersection of all three perimeters.

Another possibility is to use a relatively simple solution for position tracking, called wheel odometry, although it can only be used with a wheeled mobile robot [47]. Wheel odometry means that a mobile robot is precisely tracking the movement of its wheels. If a mobile robot is able to measure both linear and angular velocities, then it should be able to determine its own position in time. However, this technique has one big disadvantage, the precision of measurement. In the real-world, measurement is never ideal and error is increasing with time. As

a result, the longer this technique is used, the bigger the inaccuracy in position. Therefore, this is usually used only a complementary tracking method along with a more reliable approach.

Another way to obtain a robot's position is to scan the surrounding 2D or 3D environment. This method has already been described in section [3D Scanning]. In this context, it is used as the eyes of the robot which are continuously scanning the environment and determining its relative position to the environment, creating a method called Simultaneous Localization And Mapping (SLAM).

1.4.1. SLAM

SLAM is a computational problem of constructing or updating a map of an unknown environment, while simultaneously keeping track of the agent's location within it. Input for SLAM are sensors, mainly 3D cameras or lidars. If the agent can “see” its environment, then SLAM algorithms can determine its position relative to the seen objects. Consequently, if the agent starts to move, then seen objects are observed from different angles and the SLAM algorithm determines the new position of the agent after moving, making the agent’s movement trackable. However, this is only the case in a static environment where everything maintains its fixed position at all times. It gets much harder to determine what exactly is moving once the objects in the environment change their positions as well, the agent or the observed object(s). For this reason, SLAM must also use other sensors for detecting its own movement, like accelerometers, gyroscopes, and magnetometer, creating the Inertial Measurement Unit (IMU).

IMU can provide the scanning agent with its own orientation, position, and velocity. This way, if the agent with SLAM can detect its own position change as well as any environment changes caused by object movement, then SLAM can calculate the agent’s exact position inside the observed environment.

To successfully operate an agent in some environment, two methods must be combined. The first one, SLAM, would map a new unknown environment and create a map of it. This could be done either manually, with controlled movement of the agent, or automatically, where the agent would roam around to discover the new environment. After successfully creating a full map of the environment, the operator might perform the second stage, where they would specify some points in the now-known map telling the agent where to go to perform a certain action. As far as dynamic environment is concerned, IMU makes it possible to update the map continuously during the second stage and help the agent adapt to new positioning of the observed objects [48].

1.4.2. RTAB-Map

Real-Time Appearance-Based Mapping (RTAB-Map) is a Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. It can combine RGB-D, Stereo and Lidar to create a 3D point cloud representation of the scanned environment. It uses loop closure detectors for reducing errors in generated graphs. In essence, loop closure is a method which can find previously scanned points and accordingly correct errors in a generated map graph.

The best way to present the feature of loop closure is to scan premises inside the building and return to the starting point by taking a route different to that which had been scanned. The already scanned environment is then detected and the current position aligned with the previously scanned environment. This alignment is back propagated to previously scanned parts.

RTAB-Map can operate in two regimes. One is mapping and one is localization. While mapping is used for scanning the unknown environment, the second regime, localization, is used for determining position in a known (previously scanned) environment. It is a key component for navigation, as it is essential for determining the actual position of the agent.

1.4.3. Navigation Mesh

Navigation Mesh is a data structure describing a surface on which an agent can move. The surface is overlaid with polygons usually projected in 2.5D space. Therefore, such a surface can include slopes as well as places of higher or lower altitudes. For this reason, Navigation Mesh is mainly used in ground navigation for autonomous mobile robots.

One of the advantages of navigation mesh is that it can be created quite quickly from a 3D model which represents the surrounding environment. Input for mesh creating is typically a 3D floor model with some constraints such as the highest possible slope angle or agent dimensions. Then, a navigation mesh is created and all accessible places for the agent to visit are defined. This makes it possible for each agent to have its own navigation mesh according to which they could move.

Lastly, a navigation mesh tends to be used for finding the path from point A to point B. For such purposes, some path planning algorithms as Dijkstra or A* are often used [49].

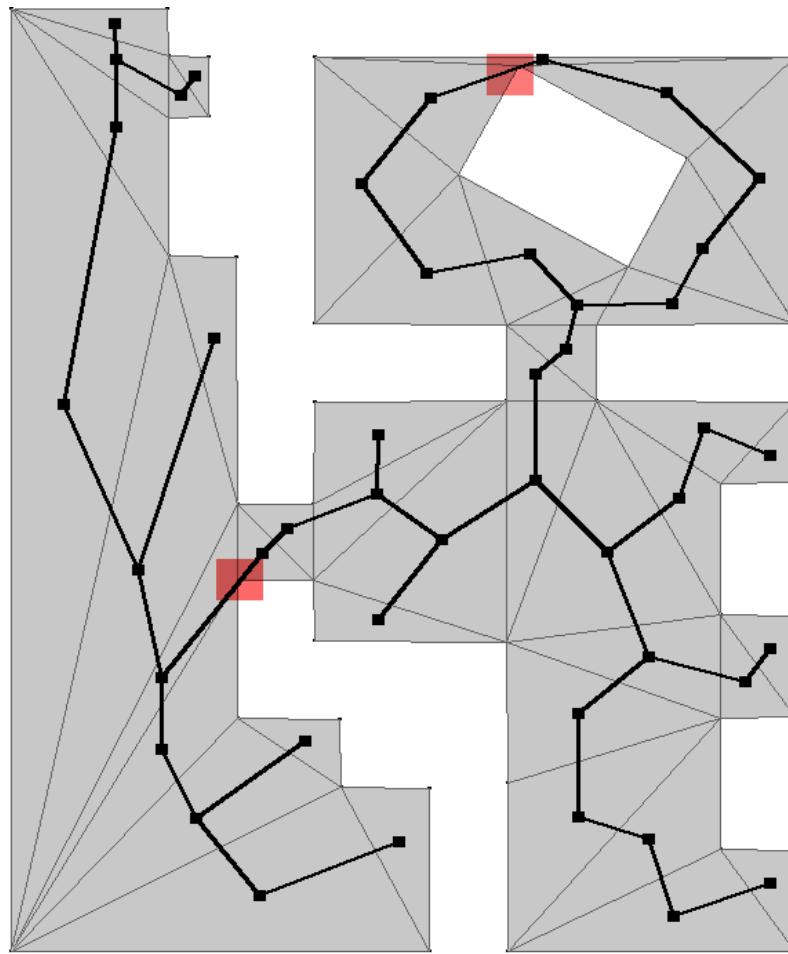


Fig. 15 Navigation mesh with all possible paths

1.4.4. Exploration

Exploring is the process of autonomous movement in an unknown environment with the intention of minimizing unseen areas. The result of exploration is a map which could be used later for localization or just for the map itself which is describing previously unexplored environment. To accomplish this goal, various methods can be used.

1.4.4.1. Wall following

Wall following is based on constant tracking of a wall. It could be explained with a simple example of a person trying to find a path through a maze. If such a person constantly holds one of their hands on one of the walls (left or right). Eventually, they will either find the way out of the maze, or they will return to their initial position. This approach might be used by an agent which needs to be able to sense a wall somehow by using a 2D laser scanner or a 3D camera. However, this could be used only for 2D-like environments [51].

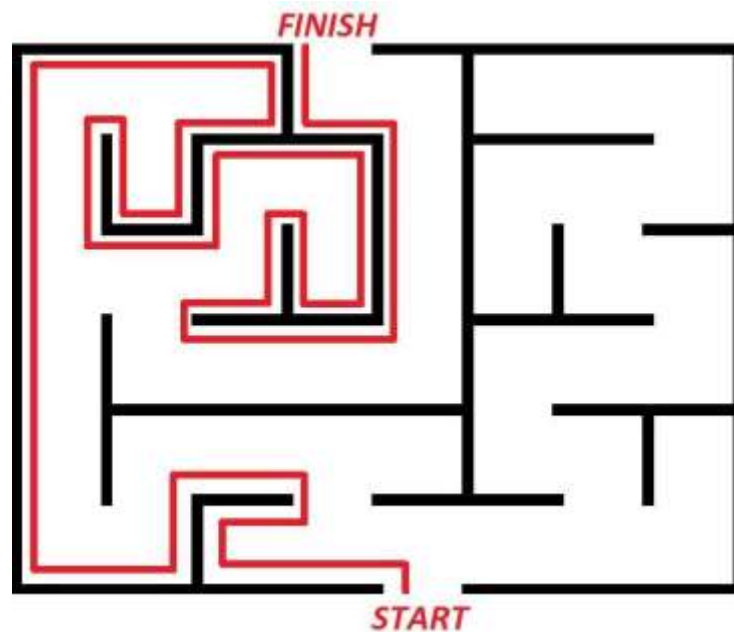


Fig. 16 Wall following algorithm

1.4.4.2. Frontier-based approach

Frontier-based approach relies on finding frontiers. Frontiers are defined as regions on the boundary between open space and unexplored space with specific width, usually defined by agent's dimensions [52]. After a frontier is found, an agent can move into the frontier to explore further unknown areas and identify new frontiers. When all frontiers have been visited, the map is considered fully explored. In frontier-based approach, optimization is a key feature in exploring an environment and it is done by making the agent only explore the frontiers which will contribute to exploration the most [53][54].

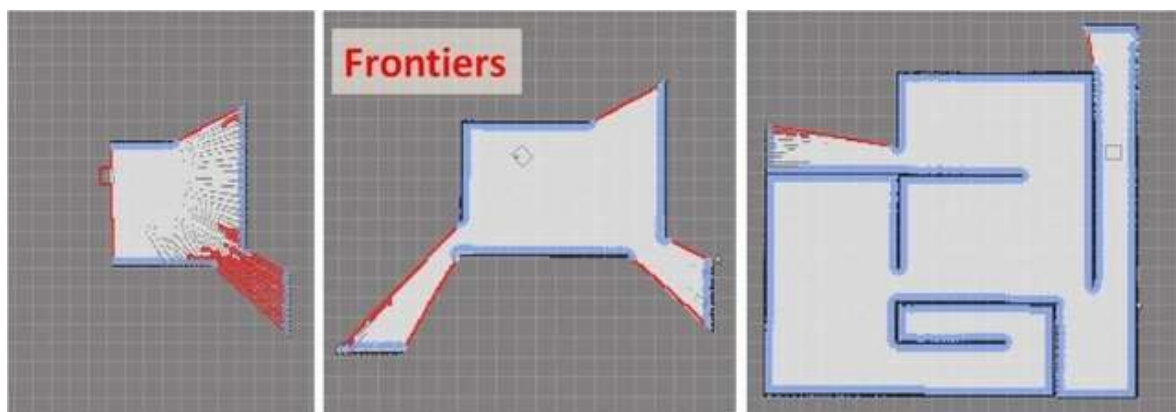


Fig. 17 Frontier-based exploration

It is possible to use a frontier-based approach in 3D exploration, although with some modifications. If the map was represented as an octo map, its frontiers would be regions of cells from the octo map between open and unexplored space [55].

1.4.4.3. Random exploration

Random exploration is a simple but inefficient method for exploration where the agent moves randomly in the environment. In this method, the agent does not have any prior knowledge of the environment, and it moves in random directions until it discovers something new. The disadvantage of this method is that it is very time-consuming and can require a lot of energy, especially if the environment is large. However, it can be useful in environments where there is no prior information available, and the agent needs to explore the area thoroughly. It can also be used as a baseline method to compare the performance of other exploration methods.

1.4.4.4. Potential fields

In this approach, the agent is attracted to the goal location and repelled by obstacles in the environment. The potential fields algorithm works by creating a virtual force field around the agent, with the goal location being the center of the field and obstacles creating repulsive forces. The agent moves in the direction of the resulting net force, which is the sum of the attractive and repulsive forces. Potential fields exploration has some advantages over other exploration algorithms. It is simple to implement, computationally efficient, and does not require an accurate map of the environment. However, it also has some limitations. One is that it can get stuck in local minima, where the attractive force from the goal is outweighed by the repulsive forces from nearby obstacles. Another is that it can generate erratic and unpredictable paths, especially in cluttered environments [56].

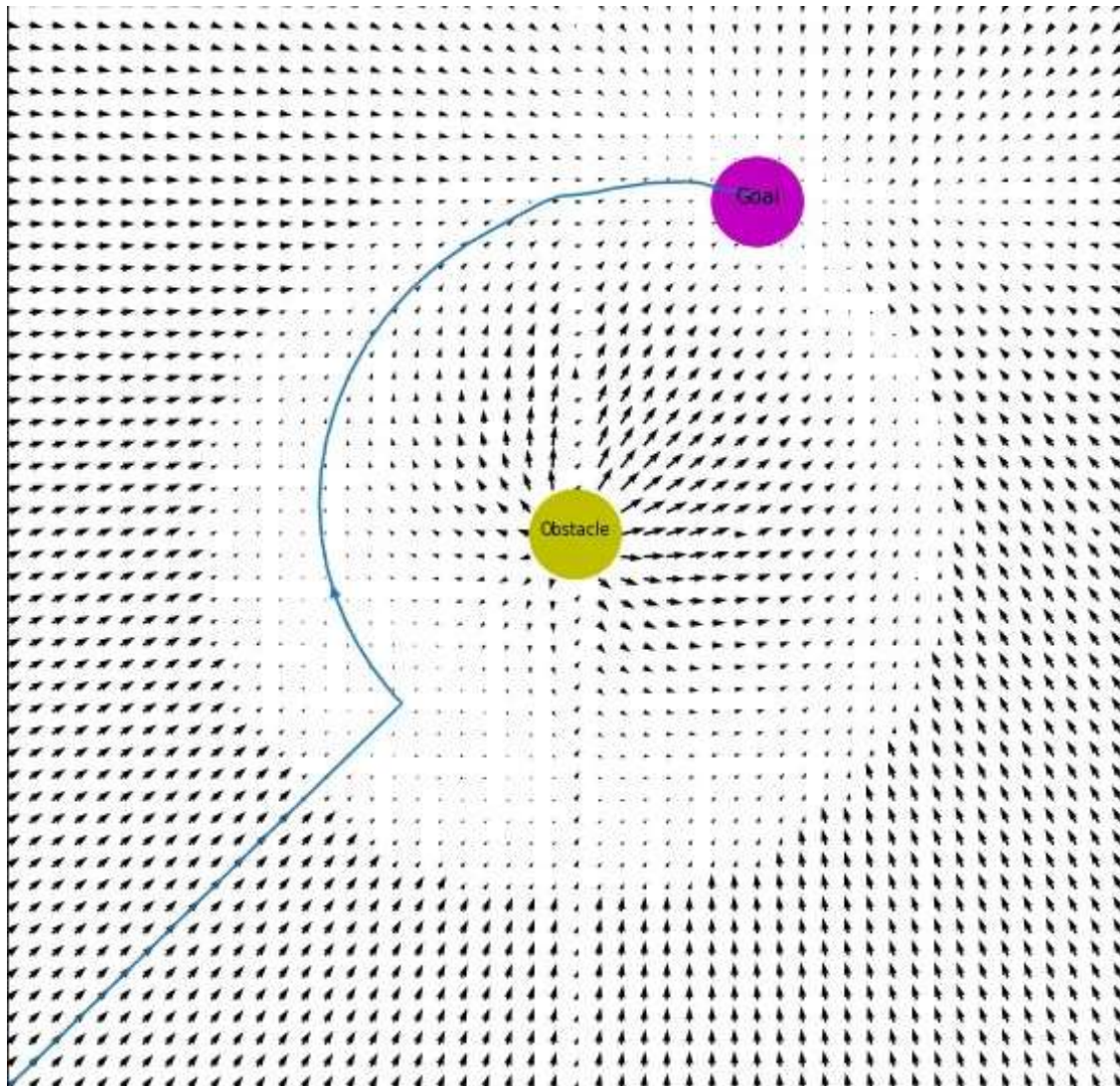


Fig. 18 Path planning in potential field

1.4.4.5. Coverage path planning

Coverage path planning is a type of exploration algorithm that is commonly used in applications such as agricultural spraying, environmental monitoring, and search and rescue. The goal of coverage path planning is to plan a path that will allow the agent or other autonomous vehicle to traverse a region of interest while ensuring that every point in that region is visited at least once.

To achieve this goal, coverage path planning algorithms divide the region of interest into a grid or other geometric pattern and then generate a path that will allow the agent to visit each cell in the grid. There are several different algorithms for generating such paths, but most involve calculating a coverage metric that estimates the amount of uncovered area in each cell and then using this metric to prioritize the order in which cells are visited.

Some coverage path planning algorithms use a complete coverage approach, in which the agent is required to pass directly over each cell in the grid. Others use a partial coverage approach, in

which the agent is allowed to skip over cells if the coverage metric indicates that they have already been sufficiently covered by adjacent cells [57].

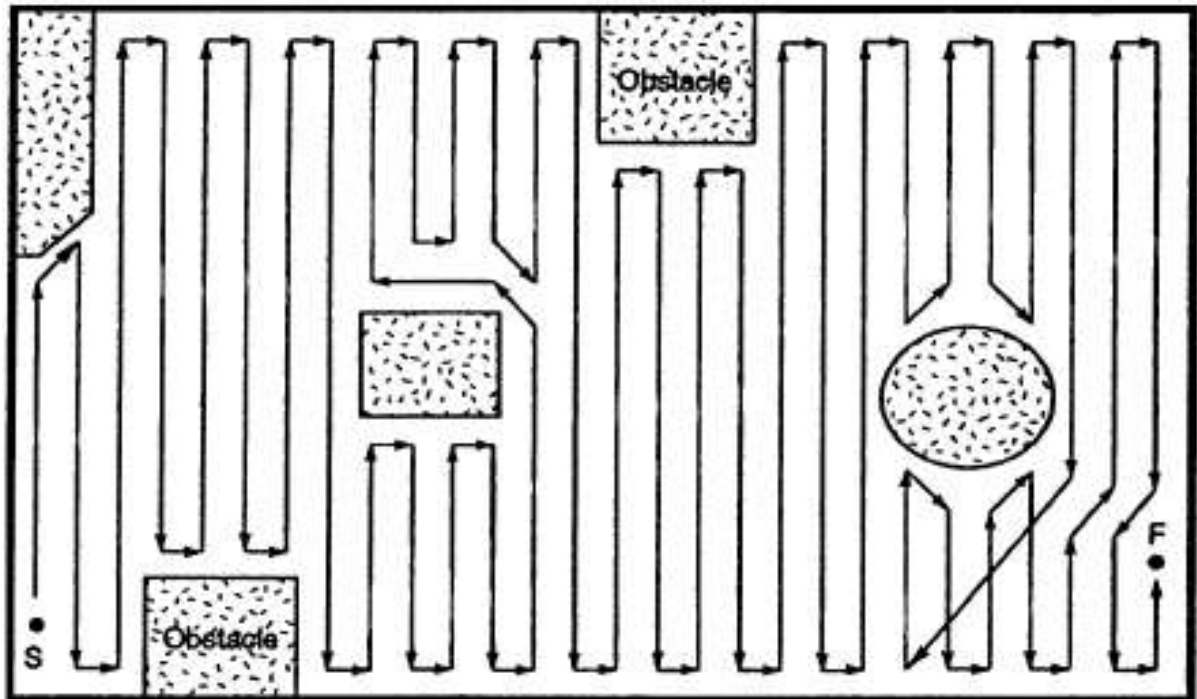


Fig. 19 Coverage path planning in an unstructured environment.

2. Proof of Concept: Handheld 3D scanner for Validating 3D

Mapping Features

In the first stage of the dissertation thesis, the main components were selected, validated, and designed to be integrated into the final autonomous UAV product. To achieve this, a handheld 3D camera was created by the author to validate the 3D mapping features. The handheld camera represents a turned-off drone that can be manually moved around to create a 3D map of an indoor environment. The effectiveness and efficiency of the 3D mapping feature were ensured in this proof of concept. Once the proof of concept is successfully validated, a real flying drone can be constructed to perform the same function. The mapping of indoor environments using its onboard 3D camera system will be autonomously performed by the drone, which will be controlled by a pilot over an RC transmitter.

2.1. Requirements for hardware components

Many things need to be considered when creating a 3D scanner capable of scanning and reconstructing an indoor environment. Besides all else, the extra limitation is that the components should be reused for autonomous drone development.

- Affordability - Most used components can be found in RC models which are easy to obtain.
- Energy efficiency - All electrical components must be powered by a local battery. Therefore, they should have the lowest possible power consumption.
- Battery capacity - The battery must have enough capacity to allow the device to function for a reasonable time.
- Wireless communication - Wired communication is too restrictive, so wireless technology must be used.
- Computing power - A computer that is required to host all of the logic necessary for 3D reconstruction must have the ability to process all necessary algorithms.



Fig. 20 The 3D handheld camera, left up) depth and tracking camera mounted on front of the handheld camera, left bottom) casing of handheld camera, right) inner components

2.2. Sensors

Sensors are fundamental to 3D scanning because their features affect the quality of scanned objects. The Intel RealSense D435i depth camera was used as the primary sensor in the project [58]. It was designed primarily for robotics, and it has the appropriate features for the purpose of this work. The camera is small and lightweight and can be used indoors as well as outdoors. D435i's depth-sensing technology is based on active stereo vision, which is built from two infrared cameras, one RGB camera and an infrared projector casting a random pattern on scanned objects. An Inertial Measurement Unit (IMU), which can sense the acceleration and rotation of the

Specification	Value
Depth Field of View (FOW)	87° x 58°
Depth output resolution	Up to 1280 x 720
Depth frame rate	Up to 90 fps
RGB sensor FOV (H x V)	69° x 42°
RGB frame resolution	1920 x 1080
RGB frame rate	30 fps
Weight	75g
Dimensions	(90 x 25 x 25) mm

Tab. 2 Intel RealSense D435i specification

camera, is also included among the features. This unit is used in Visual Simultaneous Localisation and Mapping (V-SLAM) algorithms. However, in this project, a separate component was selected to track its movements. Tab. 2 lists the main features of this product.



Fig. 21 Intel RealSense depth camera D435i

The setup includes a tracking camera from the same manufacturer as the first camera. The Intel RealSense T265 tracking camera can run Visual SLAM and outputs the exact location of itself in the scanned environment [59]. It features a dual fisheye lens that has a combined 163° field-of-view and a low-power visual processing unit capable of running V-SLAM. It also integrates IMU, which is essential for V-SLAM. Although the depth camera can provide input for V-SLAM, a dedicated camera used for localization is used instead. It provides greater accuracy and even lifts some load from the mission computer, which does not have to run CPU-intensive algorithms.

Specification	Value
FOW	163° x 99°
Output resolution	Up to 1920 x 1080
Odometry output frequency	Up to 200 Hz
Weight	55g
Dimensions	(108 x 25 x 13) mm
Power consumption	1.5W

Tab. 3 Intel RealSense T265 specification



Fig. 22 Intel RealSense tracking camera T256

Both the depth and tracking cameras are mounted together to allow for their joint movements. Since the information about the position comes from the tracking camera and scanned point cloud from depth camera, a coordinate transformation matrix must be defined to align both inputs into one coordinate system. Both cameras are facing the same direction therefore no rotation needs to be considered. It is enough to define the matrix T (1) which shifts the position of each scanned voxel to a new position v (2). This allows the handheld scanner to capture the exact position of the scanned point cloud.

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x + t_x \\ v_y + t_y \\ v_z + t_z \\ 1 \end{bmatrix} \quad (2)$$

2.3. Battery and Powering of the Internal Components

The Handheld scanner must be capable of working independently so it must have its own source of power. A Li-Polymer 3S 5000 mAh battery is used for this purpose. This battery is commonly found in small RC and UAV models. Its 5000 mAh capacity allows for 11 hours of idle time and 3.5 hours of active scanning (see Fig. 23).

The 3S battery specifications refer to the number and type of cells used. Each of them has a nominal voltage of 3.7V. The total battery voltage is therefore 11.1V, which also refers to the nominal value and minimum usable value. There are a few limitations to LiPo batteries. They cannot be overcharged, exceeding 4.2V per cell or over-discharged, dropping below 3.0V per cell

as it can lead to battery damage, permanent loss of capacity, fire, and even cell explosions. To prevent the battery from shorting, an XT60 connector was used.

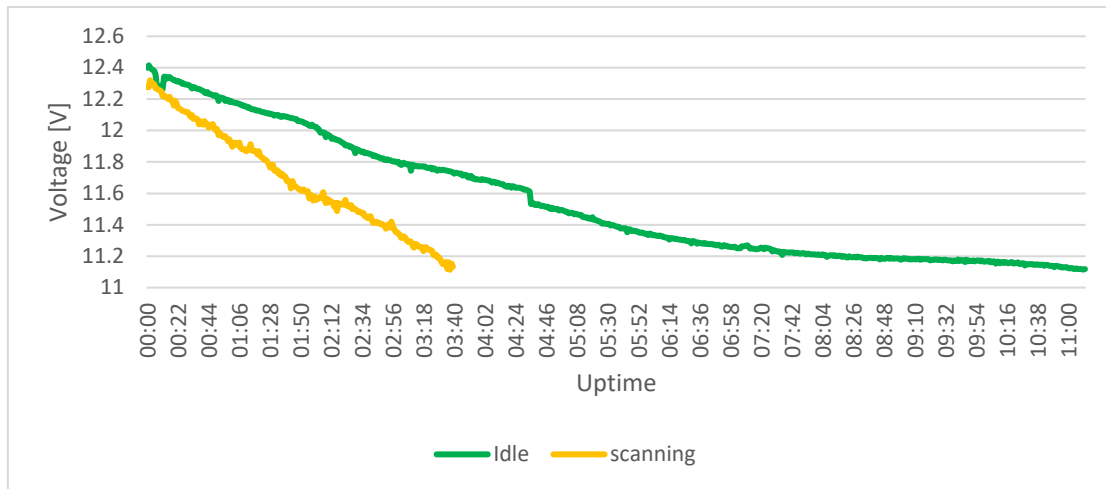


Fig. 23 Battery drain for idle and scanning mode

2.4. Conversion of Battery Voltage to 5V

Most batteries do not supply the 5V standard output because the output voltage of a battery is determined by the number of cells. One cell supplies 3.7V, while three cells provide 11.1V. However, a fully charged battery will supply $3 \times 4.2V$ or 12.6V, which far exceeds the 5V needed to power the mission computer and its peripheral sensors.

For this purpose, the Universal Battery Eliminator Circuit (UBEC) (see Fig. 24) is used to convert the output voltage of the battery to the desired level. It also stabilizes the output voltage level which should be the same regardless of the load. The UBEC 6A has been used in this project. It can produce a stable output voltage of up to 6A. However, stability is questionable because testing proved that load bigger than 3A resulted in a voltage drop (see Fig. 25). Despite the voltage decline, it proved usable because the overall current hardly ever exceeded the 1A threshold.



Fig. 24 Universal BEC which converts 6-25V into 5V/6V

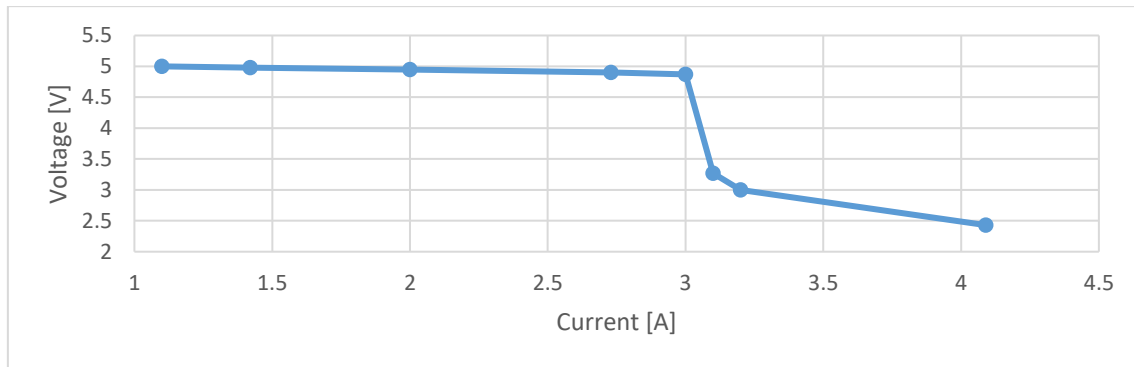


Fig. 25 Test of UBEC Voltage stability

2.5. Battery Protection and Consumption Measurement

Battery, as a main source of energy, must be protected against accidental damage which can be caused if the battery is over discharged. To achieve this, a measurement circuit was attached to the scanner to constantly measure the voltage and current. That is why the Adafruit INA260 board (see Fig. 26) was chosen as it can measure DC voltage up to 36V and current up to 15A. The primary component of this board is a shunt resistor with small resistance of only 2m Ω . The precision of this device is 1.5mA, which is much more than necessary. Power for measurement board is supplied by 3.3V or 5V and data communication handled by an I²C protocol. This board also provides one extra pin which might later be configured to activate if a certain condition occurs, for example if voltage exceeded some predefined threshold. Therefore, it could be used to automatically turn off the whole device to avoid damage to the battery.



Fig. 26 The measurement circuit Adafruit INA260

In this project, however, a different solution was used to achieve the same effect. The mission computer automatically launches a Python script to read the data from the battery measurement circuit via the I²C protocol. The computer should turn off whenever the voltage falls below 11.1V.

Although not a perfect solution (certain components can drain the battery even when the computer is idle), it will suffice for the purposes of the project. On top of that, the handheld scanner will not operate without the user and therefore the user will be able to notice the problem and turn the device off, using the master switch.

2.6. Mission computer

The main computer housed within the handheld scanner, known as the mission computer, is the Raspberry Pi 4 Model B for this project [60]. This single-board computer provides ample computational power necessary for processing data from the depth camera.

Specification	Value
CPU	Quad core Cortex-A72 (ARM v8) 1.5 Ghz
RAM	8 GB
Wireless	2.4 GHz and 5.0 GHz IEEE 802.11ac
Bluetooth	5.0, BLE
Ethernet	1 Gbit
USB	2x v2, 2x v3

Tab. 4 Raspberry Pi model B specifications

2.7. USB peripherals

A set of USB ports are available on the Raspberry Pi for connecting to peripherals. The tracking camera is connected to USB3 and the depth camera to USB2. A different USB version is required because the tracking device has a bug that makes it not work as soon as it is turned on. It is recognized as a separate device by the computer. This problem can be solved by simply restarting the camera before it is used. It can be done with the tool called `uhubctl` which will restart one specific USB port. However, in the case the Raspberry Pi, it can only restart the entire USB hub. Therefore, after a computer is started an internal script ensures that USB3 port is automatically restarted. The depth camera, on the other hand, cannot be restarted as it stops working and therefore it must be connected to USB2 port.

2.8. GPIO peripherals

General Purpose Input Output is another useful peripheral of the mission computer. It is used for communication with the INA260 measuring board, which keeps track of battery consumption. The data are sent over I²C protocol and processed by a Python script running on the mission computer. Whenever battery level decreases under its nominal value this script forces mission computer to shut down.

2.9. Wireless Communication

Since the handheld scanner needs to be moved freely in certain environments, a wireless connection must be established with it. For this purpose of Wi-Fi acts as the main communication channel between the handheld scanner and the user's computer. The device was pre-configured to always connect to the Wi-Fi router which was also set to always assign the device the same IP address. The mission computer can then be reached automatically using a specific IP address once they have started.



Fig. 27 Wi-Fi local network

2.10. Software

Programs which drive the handheld scanner are centralized into the mission computer. This computer runs Linux Ubuntu Mate 20 operating system. It is a variation of Ubuntu which is adjusted for embedded applications. Having a standard operating system in this device is largely beneficial for several reasons, and mainly because it can provide all tools used in development. Among them, C++ compiler, Python interpreter and many other command-line interface tools.

The first installation was done using a monitor, keyboard, and mouse set-up. After the first stage, a Wi-Fi network was established and after that all work was done only via SSH, making work on the device almost identical to a regular computer use.

The data from scanning needs to be processed. The core program responsible for this task is called a RTAB-Map. RTAB-Map is a Graph-Based SLAM processing approach based on an incremental appearance-based loop closure detector. It can combine RGB-D, stereo, and LiDAR to create a 3D representation of point clouds in the scanned environment. Its loop closure detector helps reduce error rate in the resulting graph. In short, a loop closure is a method that can find previously scanned points and correct errors in the resulting map graph.

The best way to use a loop closure function is to scan a room in the building and then return to the starting point, using a path other than the one that had been scanned. When the scanner returns to its initial position, the scanned environment is immediately recognized, and the current position is compared with the previously scanned environment. This produces some misalignment which is immediately corrected and extends back to the previously scanned area.

RTAB-Map can work in two modes. One is mapping and the other is localization. While the mapping is used to scan the unknown environment, the second mode, localization, is used to determine the position in a known environment.

RTAB-Map also offers a wrapper for Robot Operating System (ROS) which is the primary software framework [61]. It is not an operating system, but rather a collection of robotic packages with common configurations and communication interfaces. The main advantage of this framework is that it can run "nodes" independently from each other. The ROS core functions as a central broker that connects all the nodes. They communicate using a publish/subscribe, or remote procedure calls (RPC) model. Each node sends messages to topics or receives messages from other nodes one by one by processing them separately as Python scripts or C++ executables. On top of that, nodes can run on any computer, even the one used for scene visualization. This way, RViz, the main visualization tool, runs on the user's computer and receives updates from the handheld scanner to visualize the captured scene.

2.11. Testing handheld 3D scanner

The handheld 3D scanner was tested in a house with good Wi-Fi coverage. Upon turning on the scanner, RViz launched on the user's computer with an environment variable specifying the IP address of the ROS core. The depth camera video stream was displayed in RViz, and any camera-related issues were resolved. The primary output of the scanner was a point cloud, which grew larger as more data was received. Testing with only the depth camera and RTAB-Map visual SLAM revealed low-quality data capture and frequent loss of odometry, primarily due to the performance of the mission computer. However, when the tracking camera was used for visual SLAM and RTAB-Map was fed by odometry from the tracking camera, good-quality point clouds were obtained.

The quality of the Wi-Fi connection was also found to have a significant impact on the resulting point cloud. The signal varied depending on the distance from the router and the presence of interfering walls. Tests revealed that sending the entire point cloud over Wi-Fi to RViz was not a good approach. Even though the point cloud was generated quickly enough, it was impossible to view its real-time generation on the user's computer. Therefore, the main frequency of RTAB-Map

was lowered to 0.5 Hz, and the scanner had to be moved slowly and steadily to avoid generating the point cloud too quickly. However, even if the point cloud was not displayed properly on the user's computer, it was still available offline directly on the device. This means that the data can be used later to measure the scanned area or automatically generate a floor plan.



Fig. 28 Example of a 3d scanned environment.

3. Designing and Developing an Autonomous Quadrotor Drone for Indoor 3D Scanning

Based on the findings from building a handheld 3D scanner, a quadrotor drone was constructed with some components reused and some replaced. The handheld scanner project provided insights into the key factors to consider when selecting components, such as affordability, energy efficiency, battery capacity, wireless communication, and computing power. Some of the components used in the handheld scanner, such as the depth camera, tracking camera, and mission computer were reused in the construction of the quadrotor drone. However, some new components were added, such as the frame, motors, propellers, and flight controller. Additionally, some components had to be replaced to ensure optimal performance, such as the battery and power distribution components. Overall, the insights gained from building the handheld 3D scanner project were invaluable in the construction of the autonomous quadrotor drone.

3.1. Drone construction

Unmanned Aerial Vehicles (UAVs) can be broadly categorized into fixed-wing UAVs, flapping-wing UAVs, or rotorcrafts. For the purposes of this work, the best option is a small rotorcraft UAV, typically known as a Micro Aerial Vehicle (MAV). However, UAV described in this work is bigger than common MAV therefore the word “drone”, which is also widely used, will be used further to describe UAV used in this work. The reason for choosing this small rotorcraft UAV is that it can perform vertical takeoffs, hover in position, and fly slowly, all of which are crucial for indoor flying. The market currently offers a wide variety of drones with various features. However, the most desirable aspect is having a drone constructed from readily available components that meet the required specifications. In the subsequent sections, the primary components of the drone will be explored in greater detail.

3.1.1. Frame

The drone frame is a fundamental component of the quadrotor drone, as it serves as the skeleton that holds all of the other components together. The frame is typically made of lightweight materials such as carbon fiber, aluminum, or plastic to reduce the overall weight of the drone.

For the selected quadrotor drone, the Holybro X500V2 was chosen as the frame (see Fig. 29). It is a sturdy and lightweight frame made of carbon fiber, making it both durable and easy to transport. The frame features an X-shaped design, which allows for a more stable flight and better payload capacity. It has a diagonal wheelbase of 500mm and can accommodate up to 15-inch

propellers. The frame also has ample space for mounting additional components such as a flight controller, motors, and batteries. In addition to the frame, the kit contains a camera mount that will be used to hold the depth camera and tracking camera.



Fig. 29 Holybro X500V2 frame with propellers and camera mount [62]

3.1.2. Battery

For the autonomous quadrotor drone, the main source of power used was a 4S LiPo battery with a capacity of 4500mAh (see Fig. 30). This battery was chosen instead of an already purchased 3S battery because the motors used in the Holybro X500V2 kit required a 4S battery.

A 4S battery is a lithium-ion battery consisting of four cells connected in series. The nominal voltage of each cell is 3.7 volts, so the nominal voltage of a 4S battery is 14.8 volts (4 cells x 3.7 volts/cell). The minimal safe voltage for a 4S battery is around 12 volts (3.0 volts/cell). Discharging a lithium-ion battery below this voltage can damage the cells and reduce their capacity, and in extreme cases, it can cause the cells to fail or become unstable. The maximal safe voltage for a 4S battery is around 16.8 volts (4.2 volts/cell). Charging a lithium-ion battery above this voltage can

also damage the cells and reduce their capacity, and in extreme cases, it can cause the cells to overheat or catch fire.

The 4S LiPo battery with a capacity of 4500mAh is a commonly used battery in the quadrotor drone industry, as it provides a good balance between weight and flight time. With this battery, the autonomous quadrotor drone can fly for approximately 15 to 20 minutes, depending on the payload and flight conditions.



Fig. 30 4S LiPo battery with capacity of 4500mAh.

3.1.3. Power distribution

Power distribution is a critical aspect of any quadrotor drone, as it ensures that all of the components receive the appropriate voltage and current to function properly. The PM03D Power Module was used to facilitate power distribution in the selected quadrotor drone, and it serves multiple functions (see Fig. 31).

Firstly, the power module measures the battery level and provides the user with real-time feedback on the battery voltage and current draw. This information is essential in ensuring that the drone operates within safe limits and that the battery is not over-discharged, which can damage the battery and reduce its overall lifespan.

Secondly, the power module converts the battery voltage to a lower voltage level that is safe for the other components. In the case of the selected quadrotor drone, the voltage is reduced from 14.8 volts to 5 volts, which is the standard voltage level required for most electronic components used in the drone.

Finally, the power module also serves as a power distribution board, providing direct battery voltage to the electronic speed controllers (ESCs). The ESCs regulate the power to the motors, adjusting the power to each motor based on the flight controller's commands.

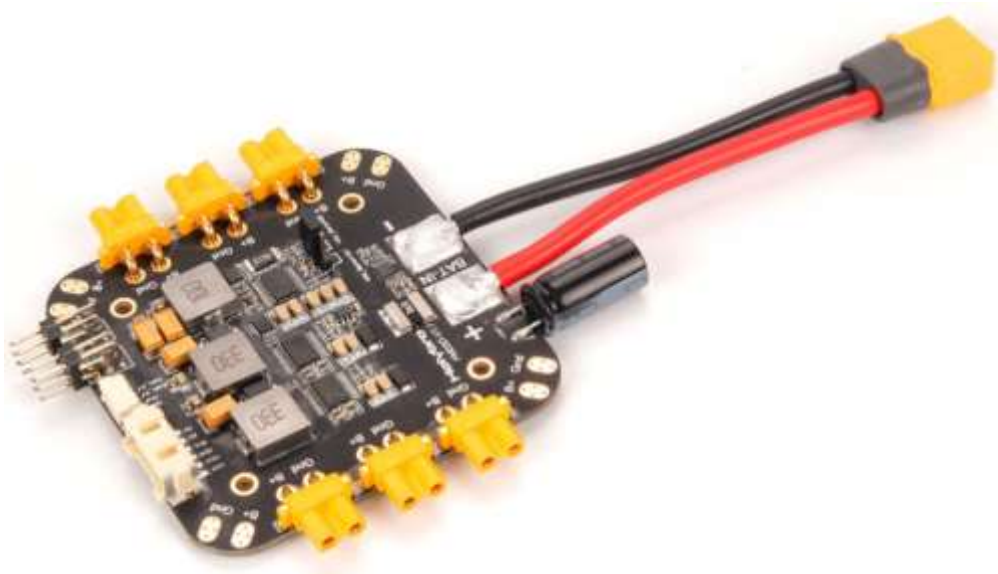


Fig. 31 Power distribution board.

3.1.4. Motors and ESC

The Holybro X500V2 quadrotor drone uses four AIR 2216 KV920 BLDC motors (see Fig. 32). These motors are designed specifically for multirotor drones and are ideal for high-performance flying. The 2216 size of the motor refers to the motor's dimensions, with a diameter of 22mm and a length of 16mm. The KV920 rating indicates that the motor's no-load speed is 920 RPM per Volt. The motor is made of high-quality materials and features a low resistance and high efficiency design, which allows it to generate more power while using less energy.

The BLHeli S ESC 20A is used to regulate the power to the motors. These ESCs are compact, lightweight, and powerful, providing an efficient and reliable way to control the motor's speed. They use high-quality MOSFETs and have a fast response time, allowing for precise control of the motor's speed. The ESCs have a maximum continuous current of 20A, making them suitable for use with the AIR 2216 KV920 motors. The BLHeli S firmware used in these ESCs provides advanced features such as motor braking and active freewheeling, which improve the overall performance of the quadrotor drone.

Before use, the ESC needs to be calibrated to ensure accurate control of the motor's speed and prevent damage to the motor or other components. Calibration involves setting the minimum and maximum throttle endpoints of the ESC to match the transmitter's throttle output signal, allowing for smooth and accurate control of the drone.

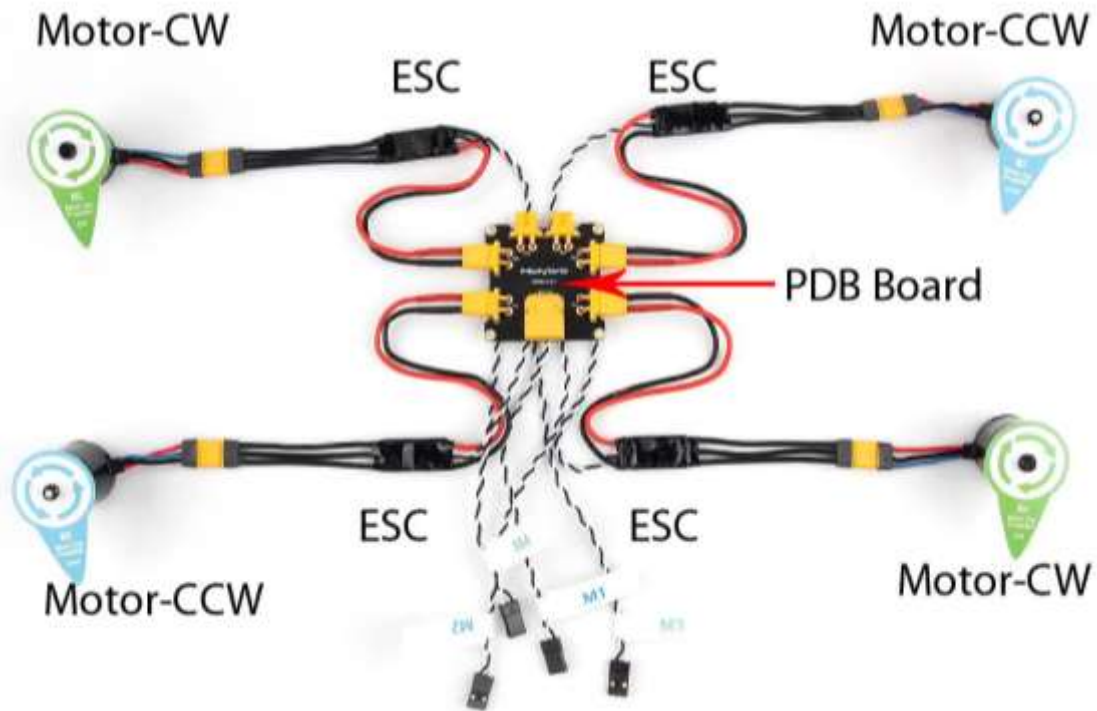


Fig. 32 Motors and ESCs

3.1.5. Flight controller

The next essential component used in the autonomous quadrotor drone is the flight controller, which is responsible for controlling the motors and stabilizing the drone's position in flight. The flight controller used in this work is the Pixhawk 6x (see Fig. 33), which is a highly reliable and versatile flight controller that runs on the PX4 firmware [63].

The Pixhawk 6x flight controller has several key features that make it an ideal choice for this project. Firstly, it receives commands from the transmitter and controls the motors through the ESC in such a way that the drone flies in the direction it is supposed to. It also processes data from the inertial measurement unit (IMU) and stabilizes the drone's position in flight.

In addition, the flight controller monitors the battery level and notifies the operator or performs an emergency landing if necessary, ensuring that the drone is operating within safe limits. If distance measurement sensors are attached, the flight controller can process the information and perform collision avoidance maneuvers, enhancing the safety of the drone.

Furthermore, the Pixhawk 6x flight controller can read data from a GPS sensor to determine the drone's absolute position, enabling precise navigation. The GPS data can also be used for mission planning and waypoint navigation, allowing the drone to fly autonomously and perform specific tasks.



Fig. 33 Pixhawk 6X flight controller.

3.1.6. Mission computer

The mission computer is a separate computer that performs higher logic of the autonomous quadrotor drone. Its functionality includes processing data, navigation, exploration, and communication with the flight controller and ground station. The Raspberry Pi 4 Model B has been selected as the mission computer due to its computation power and standard operating system, which make it easier for software development, debugging, and profiling.

The mission computer's main role is to process the output data from the 3D camera and create a model of the observed environment. It also determines the quadrotor drone's position in the environment using odometry from the tracking camera and performs exploration by finding uncovered regions and planning a trajectory to visit them. The mission computer communicates with the flight controller over the Ethernet port to adjust the quadrotor drone's flight path.

In addition, the mission computer communicates with the ground station over a Wi-Fi connection, allowing remote control and monitoring of the quadrotor drone. The ground station can receive telemetry data from the quadrotor drone, such as its position, speed, altitude, and battery level, and send commands to the mission computer to adjust its flight path or perform specific tasks.



Fig. 34 Raspberry pi 4 model B

To further enhance the performance of the Raspberry Pi, the processor was overclocked to 2 GHz. This increased the clock speed beyond the default speed set by the manufacturer, resulting in faster and more efficient processing of data. However, overclocking can also cause the Raspberry Pi to run at higher temperatures, which may lead to stability issues and potential damage to the system.

To address this issue, passive heat sink was implemented to help dissipate the heat generated by the processor. Passive heat sinks work by transferring heat away from the processor and into the surrounding air, using a combination of thermal conductivity and natural convection.

To ensure that the system was stable and did not overheat, a simple test was conducted to measure the temperature at various CPU loads. The CPU was initially set to its default speed of 1.5 GHz and then overclocked to 1.75 GHz and 2 GHz. For each frequency, the CPU was stressed at load values of 0, 1, 2, 3, and 4, with 0 indicating no load and 4 indicating all four cores running at 100% capacity. The temperature was measured by an internal thermometer which is accessible by a command-line tool *vcgencmd measure_temp*. The test results showed (see Fig. 35) that even at the highest CPU load for 2 GHz, the temperature did not exceed the limit of 85 degrees Celsius. This ensured that the system remained stable and reliable during extended periods of heavy use.

The placement of the mission computer near the propellers of the quadrotor drone also provided an additional cooling mechanism during flight. As the propellers rotate, they create a flow of air that passes over the components of the drone, including the mission computer. This results in an additional cooling effect that further helps to dissipate the heat generated by the processor.

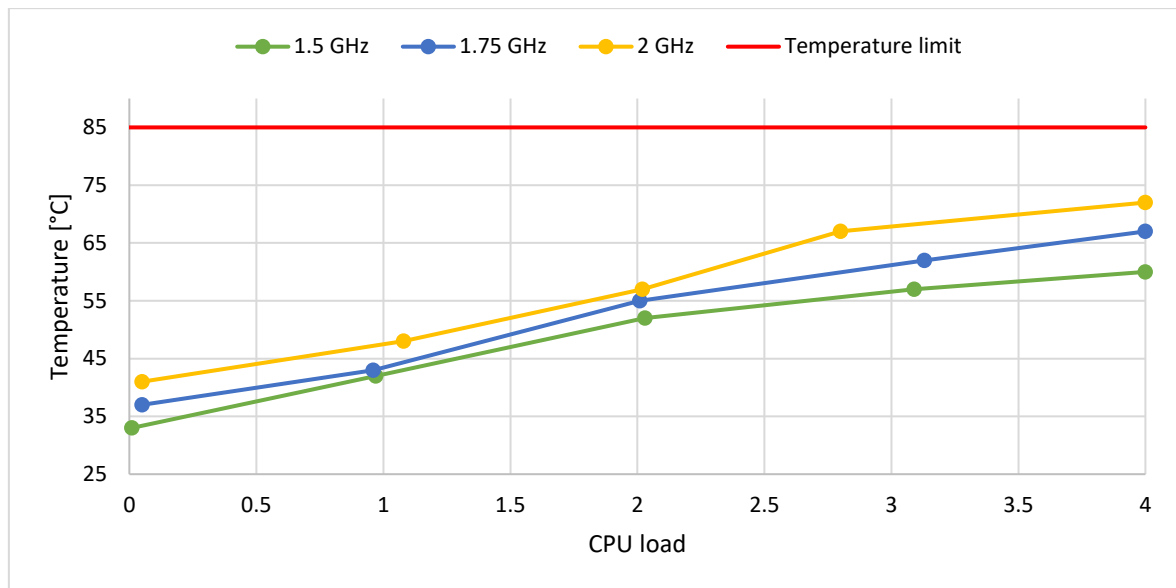


Fig. 35 Measurements of CPU temperature for overclocking purposes.

3.1.7. Depth and tracking camera

The Intel RealSense D435i (see Fig. 21) and T265 (see Fig. 22) cameras are both essential components of the autonomous quadrotor drone. The D435i provides high-quality 3D imaging and depth sensing capabilities, while the T265 uses visual inertial odometry (VIO) technology to accurately track the drone's position and movement in real-time. Both cameras are mounted together on a 3D printed camera mount that was purchased as part of the Holybro X500V2 kit, allowing them to work together seamlessly to provide the drone with the data necessary for creating a detailed 3D map of the environment and accurately navigating through it.

3.1.8. Remote control

The RC transmitter and receiver are essential components of the quadrotor drone's control system, allowing the pilot to remotely control the movement and direction of the drone. The transmitter used in this project is the Futaba T6EX, a popular and reliable transmitter with six channels of control (see Fig. 36).

The receiver used in this project is the R617FS receiver, which is designed to work with the Futaba T6EX transmitter. The R617FS receiver is a high-quality receiver with a fast response time and excellent signal reliability. It uses Frequency Hopping Spread Spectrum (FHSS) technology to ensure a strong and stable signal between the transmitter and receiver, even in environments with high levels of interference.

The Futaba T6EX transmitter and R617FS receiver work together to provide precise control over the quadrotor drone's movement and direction. The transmitter sends signals to the receiver,

which in turn sends those signals to the flight controller. The flight controller then translates those signals into specific actions by the drone's motors, adjusting their speed and direction to achieve the desired movement and control during flight. This allows the pilot to control the drone's altitude, speed, direction, and other parameters in real-time, providing a high level of precision and control during flight.



Fig. 36 Futaba transmitter with receiver and charger.

3.1.9. Telemetry radio

The final component of the quadrotor drone is the telemetry radio, which plays a critical role in enabling communication between the drone and the ground station. The specific telemetry radio used in this project is the SiK Telemetry Radio V3 100mW 433MHz (see Fig. 37).

The telemetry radio operates on the 433MHz frequency band and provides a range of up to several kilometers, depending on the specific environment and conditions. It allows the ground station to receive real-time telemetry data from the drone, including its position, speed, altitude, and battery level, as well as other important flight data.

In addition, the telemetry radio also enables two-way communication between the drone and the ground station, allowing the ground station to send commands to the drone to adjust its flight path or perform specific tasks. This bi-directional communication is essential for controlling the drone and ensuring its safe and effective operation during flight.



Fig. 37 Pair of telemetry radios.

3.1.10. Protection cage

The safety of the drone and the environment in which it operates is paramount. To achieve this, a protection cage was designed using freeCAD software and attached to the drone. The cage was 3D printed with a durable material, and lightweight carbon fiber rods were used to connect the various parts of the cage. The weight of one propeller cage is 55 grams. Additionally, a motor base plate was 3D printed to attach the cage securely to the drone frame, replacing the original one. The protection cage was crucial for the drone's operation in indoor environments, where collisions with walls or furniture could result in immediate damage to the drone's propellers, leading to a crash and potentially damaging other components. The protection cage allowed the drone to navigate indoor environments safely, providing increased protection for both the drone and its surroundings.

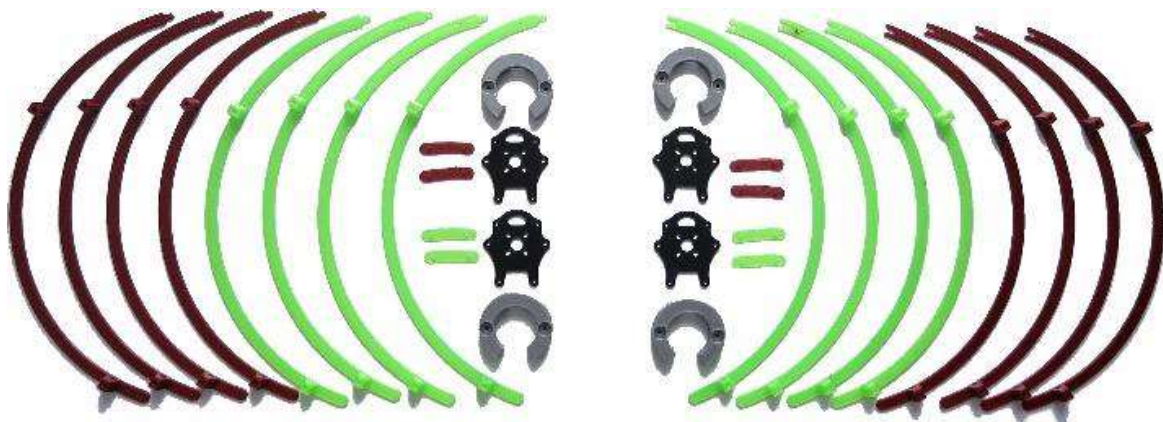


Fig. 38 3D printed parts of protection cage

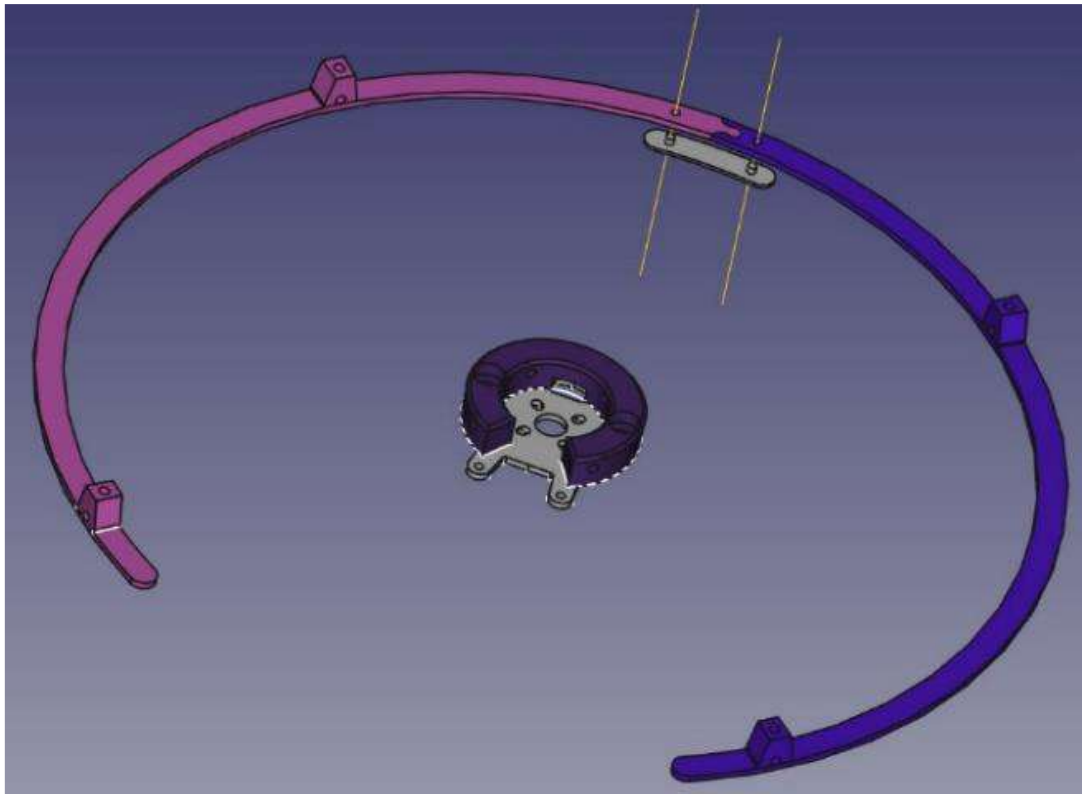


Fig. 39 Design of protection cage in FreeCAD



Fig. 40 One fully assembled protection cage.

3.1.11. Assembling of a drone

The assembly process was straightforward and guided by recommendations from the Holybro X500 V2 kit documentation. The frame contained pre-drilled holes that were used to attach the frame parts together with bolts and nuts. The middle of the drone contains base plates that are above itself. The bottom one holds landing legs and two carbon tubes that were attached using rubber and plastic holders. Those two tubes serve as holders for the battery mounting board and mission computer mount. The position of the mission computer was adjusted so that its passive cooler is located under the propellers, ensuring effective CPU cooling. On the other side of the bottom plate is the power distribution board. The second plate, which is above the first one, provides a platform for the flight controller, telemetry radio, and RC receiver. These three components are attached to the plate with Velcro, which is glued from the other side to the plate. Both main plates are bolted together in a way that they hold the drone arms, which consist of a tube in which the ESC is hidden, and on the farther side is the motor base plate that holds the motor with propeller. The motor base plate was replaced with a custom one that was 3D-printed to enable bolting of the propeller protection cage.

All cables and wires were connected to their corresponding slots according to the documentation, and there was no need for any soldering except for the battery connector, which was not provided with the purchased battery. The application for the battery can vary, so the manufacturer left this selection to the end-user. The power distribution board, which is the only component that directly connects to the battery wires, uses an XT60 connector, so the battery must also be provided with this connector. However, soldering connectors to the battery must be done with caution, as LiPo batteries are sold pre-charged and should never drop below a certain voltage limit to avoid damage. The procedure is simple - wires must be soldered one after another to ensure that each wire is properly insulated. Otherwise, it could lead to a short circuit and potentially cause damage to the battery or even injury.

The following diagram (see Fig. 41) illustrates the electrical and communication connections between the components of a drone. First, the battery supplies 14.8V to the PDB, which directly connects to the motors and also converts the voltage to 5V to power the flight controller and mission computer. Simultaneously, the flight controller communicates with the PDB via the I²C protocol to obtain information about the actual battery voltage and current, which is used to determine the battery level.

The flight controller also connects to all ESCs via PWM to precisely control motor speeds, ensuring stable flight. ESCs directly connect to motor coils to influence motor speed. Additionally, the flight

controller is responsible for communicating with ground stations through the UART protocol to the telemetry radio, which communicates with its counterpart at a frequency of 433MHz.

RC transmitter commands are directed to the flight controller, which connects to the RC receiver via the PPM protocol, using a 2.4GHz frequency for communication with the RC transmitter.

Finally, the flight controller is paired with the mission computer, which provides higher-level navigation logic through an Ethernet cable. The mission computer connects to both cameras via the USB protocol and uses its internal Wi-Fi antenna for communication with the ground station as well. This setup enables video streaming and SSH communication with the mission computer.

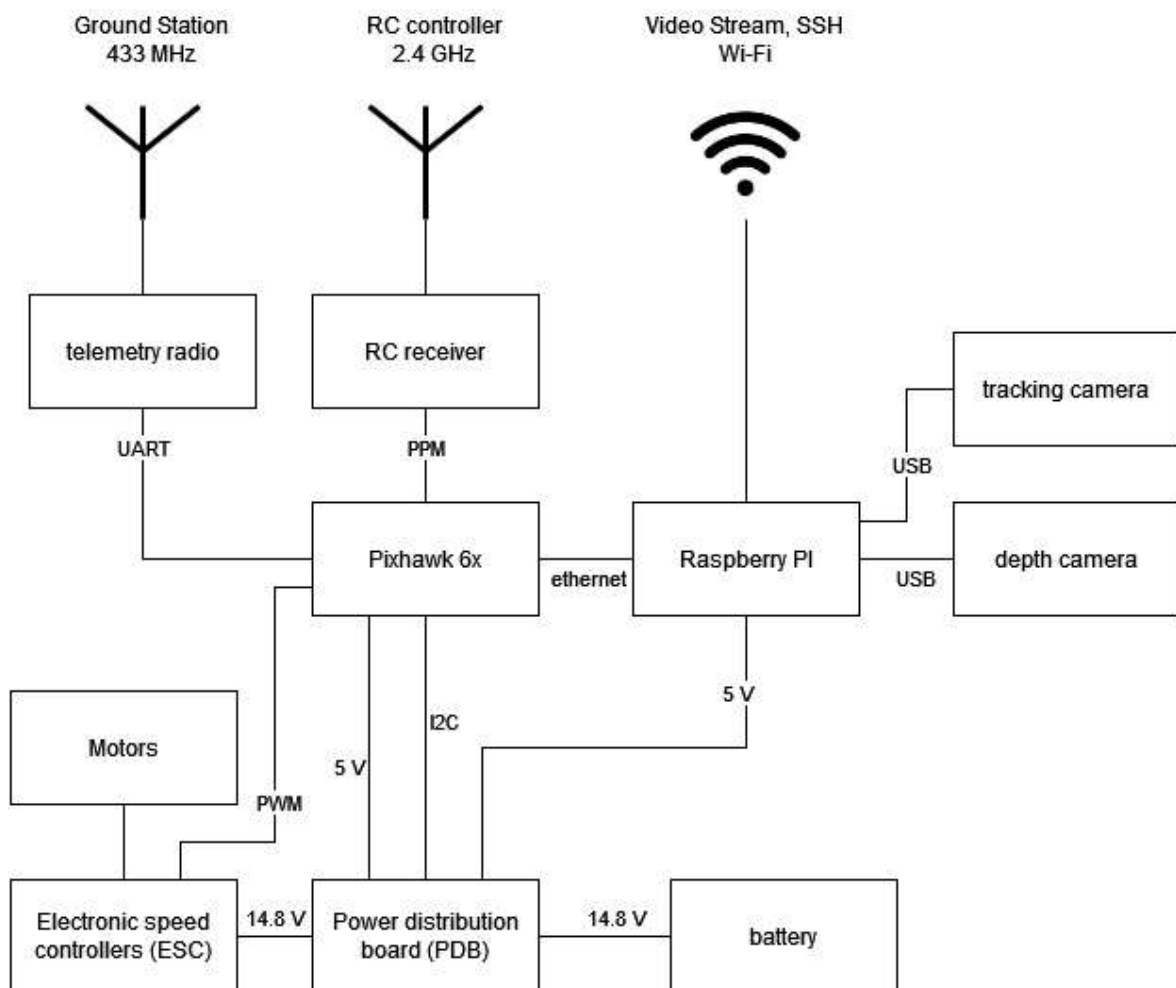


Fig. 41 Block scheme of all components used for a drone

Following two pictures (see Fig. 42 and Fig. 43) depict the drone components, their placement on the frame, and basic specifications.

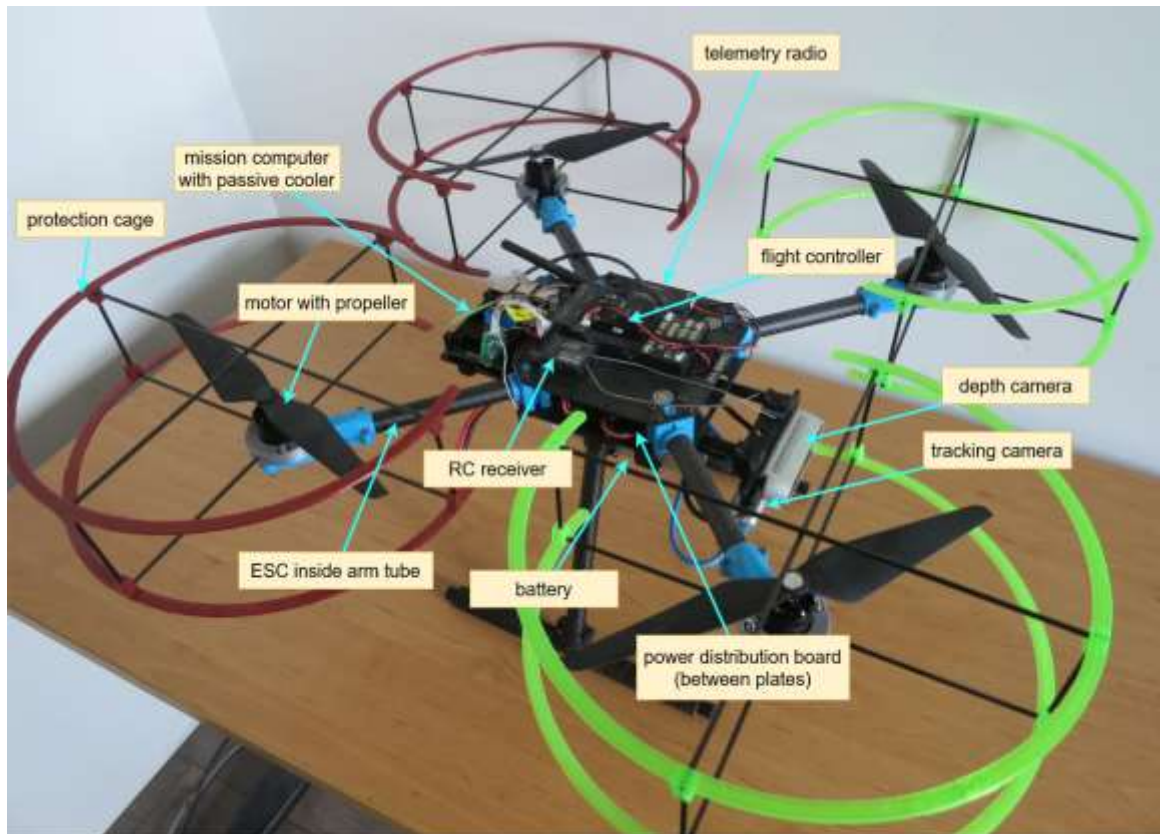


Fig. 42 Drone with component descriptions.

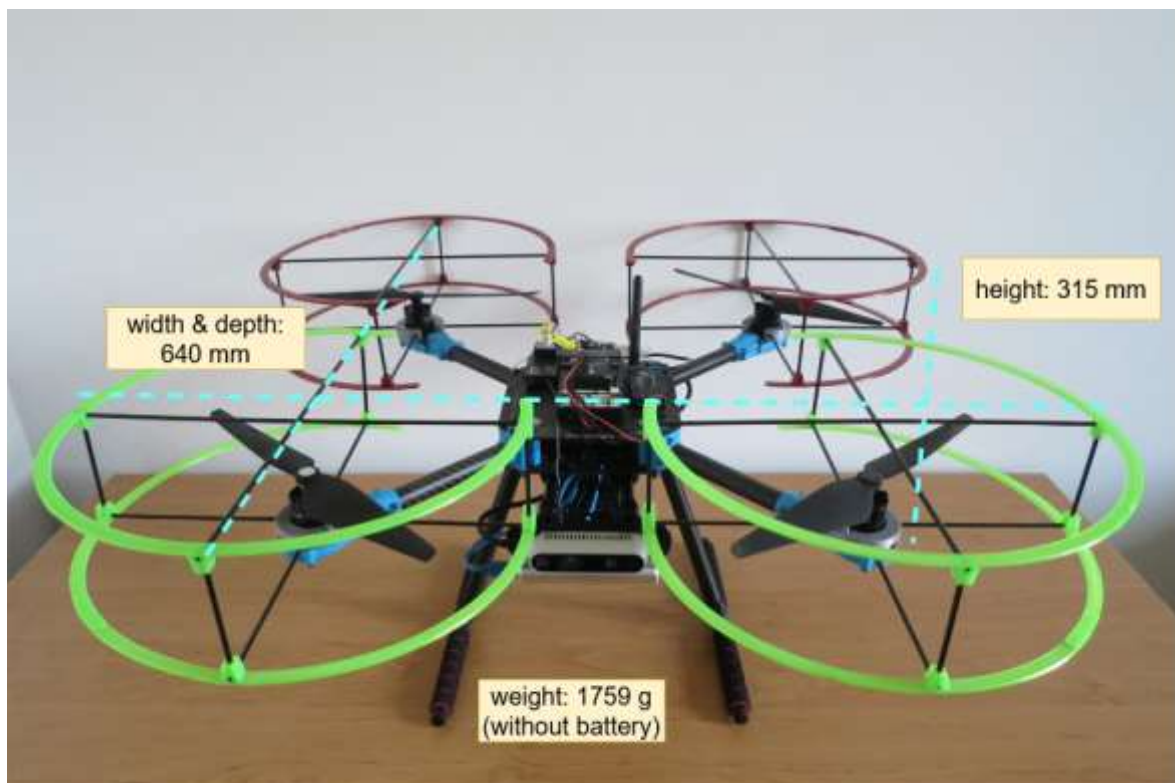


Fig. 43 Basic drone specification.

3.1.12. Power consumption calculation

The following table (see Tab. 5), provided by the motor manufacturer, contains information about the motor AIR2216 KV920 with a T1045 propeller test report, which can be used to determine the drone's power consumption. The operating temperature listed in the table is 80°C.

Throttle [%]	Voltage [V]	Thrust [g]	Torque [N*m]	Current [A]	RPM	Power [W]	Efficiency [g/W]
30	16	210	0.03	1.44	4042	23	9.12
35	16	259	0.04	1.87	4469	30	8.67
40	16	309	0.05	2.29	4855	37	8.45
45	16	373	0.05	2.86	5301	46	8.15
50	16	447	0.06	3.60	5780	58	7.76
55	16	536	0.08	4.53	6298	72	7.39
60	16	628	0.09	5.61	6800	90	7.01
65	16	729	0.10	6.78	7281	108	6.73
70	16	814	0.11	7.92	7679	126	6.44
75	16	906	0.12	9.20	8096	147	6.18
80	16	993	0.14	10.59	8468	169	5.88
85	16	1087	0.15	12.11	8867	193	5.65
90	16	1191	0.16	13.81	9257	219	5.43
95	16	1289	0.18	15.68	9675	249	5.18
100	16	1332	0.18	16.37	9857	260	5.13

Tab. 5 Test report of motor with propeller.

The weight of the drone is 1759 grams without the battery. The battery used has a capacity of 4500mAh and weighs 411 grams. The drone is equipped with four motors, resulting in a total maximum thrust of $4 * 1332g = 5328g$.

To achieve flight, the drone needs to lift a total of $1759g + 411g = 2170g$. Divided among the four motors, each motor must lift $2170g / 4 = 542.5g$. According to the table, this value can be reached with a throttle level of approximately 55%. This row displays the current and power consumption for one motor. Consequently, the total current will be $4 * 4.53A = 18.12A$, and the total power consumption will be $4 * 72W = 288W$. The battery has a capacity of 4.5Ah, so the total time required to deplete the battery is $4.5Ah / 18.12A = 0.248h$, which is 14.88 minutes.

Of course, this is a rough estimate that doesn't take into account other significant factors, such as the power consumption of other drone components, the gradually decreasing battery voltage, specific thrust during flight maneuvers, and motor temperature. However, it is sufficient to provide an approximation of the maximum flight time.

3.2. Mission Computer Software

The mission computer software runs on Ubuntu Mate 20 LTS with the ROS2 framework, providing a powerful and flexible environment for processing data, navigation, exploration, and communication with the flight controller and ground station. ROS2 is the successor to ROS and was chosen because it supports direct communication with PX4 and its data distribution model. Additionally, ROS2 provides several advantages over ROS, such as the use of the DDS communication protocol, which offers faster and more reliable communication, better real-time performance, a more modular architecture for developing and maintaining large-scale robotic systems, and support for multiple programming languages, including Python, C++, and Java.

One drawback of ROS2 is that it is a relatively new framework and not as mature as ROS. Additionally, many packages that were used in ROS have not yet been migrated to ROS2, and some parts of the software are not well-documented. However, all the required packages for this project were available in ROS2, making it a viable choice.

3.2.1. Simulation

Simulation is an important tool in the development of autonomous quadrotor drones for 3D scanning of indoor environments. In this project, simulation was used to test and validate the drone's software before deploying it on a physical drone. The simulation environment allowed to develop and refine the software without the risk of damaging the drone or causing harm to people or property.

The simulation environment was built using Gazebo (see Fig. 44), which is a popular open-source robotics simulator. To create a realistic simulation, the drone was modeled using URDF, which stands for Unified Robot Description Format. URDF is an XML format used to describe a robot's physical characteristics, such as links, joints, and sensors. To simplify the process of creating and maintaining URDF files, a xacro was used, which is a macro language that allows us to create URDF models using variables and macros.

Once the URDF model was defined, it was converted into the SDF format, which is the simulation description format used by Gazebo. The SDF file includes information about the robot's physical properties, such as mass, inertia, and collision models.

To integrate the URDF model with ROS, the `robot_state_publisher` package was used, which updates the TF tree based on the URDF model. This allows to visualize the robot's movements and orientation using RViz, a 3D visualization tool provided by ROS. RViz allows to view the robot in a simulated environment and visualize its sensor data, such as camera images and point clouds.

The simulation environment allowed to test the drone's software in various scenarios, such as navigating through a cluttered environment and avoiding obstacles. Once the software was validated in the simulation environment, it was deployed on a physical drone for further testing and evaluation.

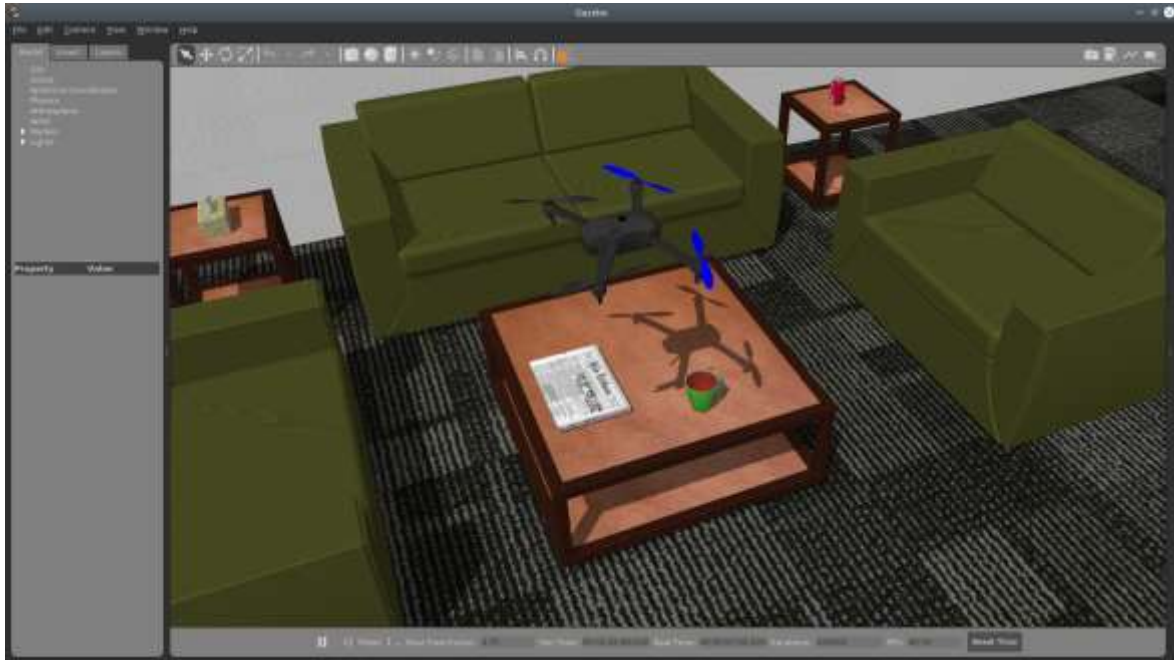


Fig. 44 Drone model in simulated environment.

3.2.2. Complete TF tree

To achieve stable navigation and accurate mapping in a robotic system, a complete and valid TF tree needs to be established. The TF tree provides a consistent representation of the coordinate frames and transformations between them, allowing different nodes in the system to communicate with each other effectively.

ROS provides a range of libraries and tools for working with the TF tree, including a TF broadcaster and listener for publishing, and subscribing to transformation updates and a TF viewer for visualizing the tree. To use the TF tree effectively, all nodes must have a defined edge to their parent. In this case, all edges are defined with static transforms that are produced by the URDF model of a drone. The entire drone is treated as a rigid body object, and the only dynamic transformation relates to the drone's position on the map. This transformation is defined as a transformation between the *odom* (odometry) and *base_link* frame, with *base_link* representing the middle point of the drone. This transformation must be broadcast with every change of position.

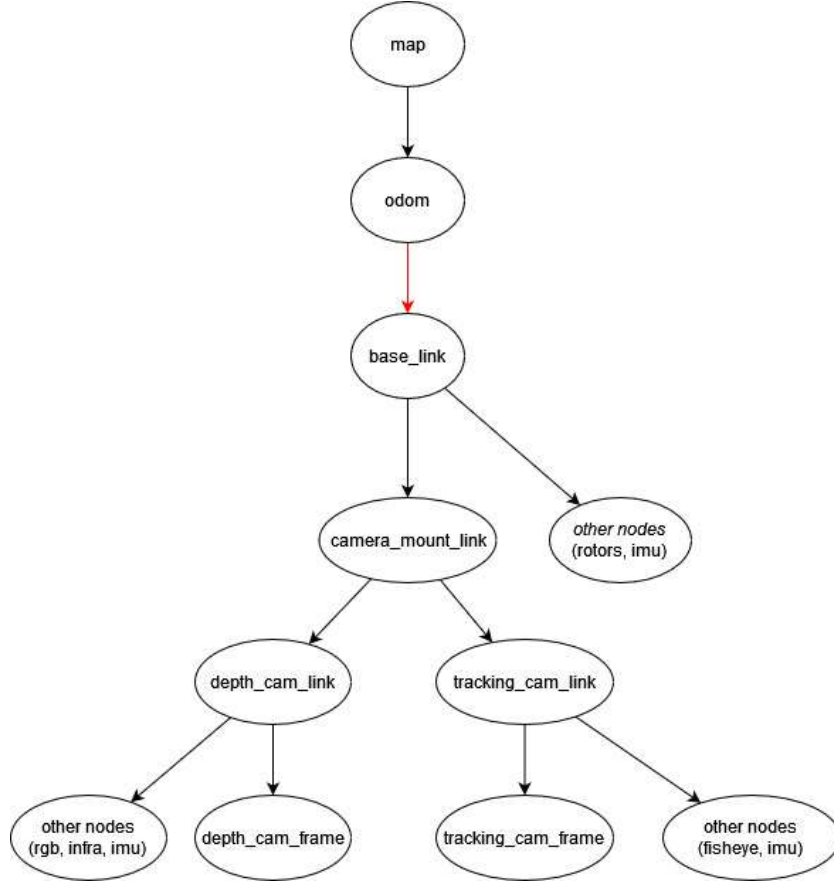


Fig. 45 Part of the TF tree showing relations among transform matrixes

However, the drone's pose is derived from the *tracking_cam_frame* instead of the *base_link* frame. As such, the pose gathered from the *tracking_cam_frame* needs to be recalculated relative to the *odom* frame. The known transformation from *base_link* to *tracking_camera_frame* can be utilized for this transformation to *base_link*.

The calculation of this transformation requires a recalculation of the pose obtained from the *tracking_cam_frame* relative to the *odom* frame. This necessitates the consideration of the static transform between *base_link* and *tracking_cam_frame*, effectively subtracting it from the obtained pose.

Let o_bT define the homogeneous transformation matrix that defines the transformation between the *odom* frame and the drone's *base_link*. Then it can be expressed as:

$${}^o_bT = {}^b_tT^{-1} \cdot P_t \cdot {}^b_tT \quad (3)$$

where b_tT represents the transformation matrix between the *base_link* and *tracking_cam_frame*, ${}^b_tT^{-1}$ represents an inverted matrix from b_tT , and P_t is a current pose represented as a transformation matrix relative to the *tracking_cam_frame*.

By applying the ${}^b_tT^{-1}$ matrix and then the b_tT matrix, we effectively cancel the transformation between the *base_link* and *tracking_cam_frame*, producing a pose as if it came from *base_link*. The resulting transformation can then be applied to the transformation between the *odom* and *base_link* frames to make TF tree complete.

3.2.3. Transforming points from the depth camera

Another transformation which needs to be applied is that of 3D points captured from a depth camera. The depth image produced by the depth camera is relative to the *depth_cam_frame*, the distance of points is measured directly from the camera. Nevertheless, it's required that these points be relative to *base_link*, which presently represents the drone's pose, as derived from the data obtained from the tracking camera.

To recalculate the depth image, the TF package was again used for calculations and obtaining transformation matrices. It is needed to define a homogenous transformation matrix which defines the transformation between the *depth_cam_frame* and the drone's *base_link* denoted as d_bT . It is defined as:

$${}^d_bT = \begin{bmatrix} {}^d_bR & \vec{t} \\ \vec{p} & s \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

where d_bR is the 3x3 rotation matrix that describes the orientation of the camera frame with respect to the *base_link* frame, \vec{t} is the 3x1 translation vector that describes the position of the camera frame origin relative to the *base_link* frame origin, \vec{p} is the 1x3 perspective vector, which is not used and therefore, its value is $\vec{0}$, and s is a scaling factor which is always equal to 1.

To transform a 3D point P_d captured from the depth camera frame to the *base_link* frame, we can apply the transformation matrix as follows:

$$P_b = {}^d_bT \cdot P_d \quad (5)$$

where P_b is the transformed point in the *base_link* frame.

Aligning the tracking camera and depth camera into a common reference frame allowed to obtain a more accurate and complete representation of the environment around the drone. By aligning the two camera frames into the *base_link* frame, a consistent and coherent view of the world can be obtained, which is necessary for localization and mapping.

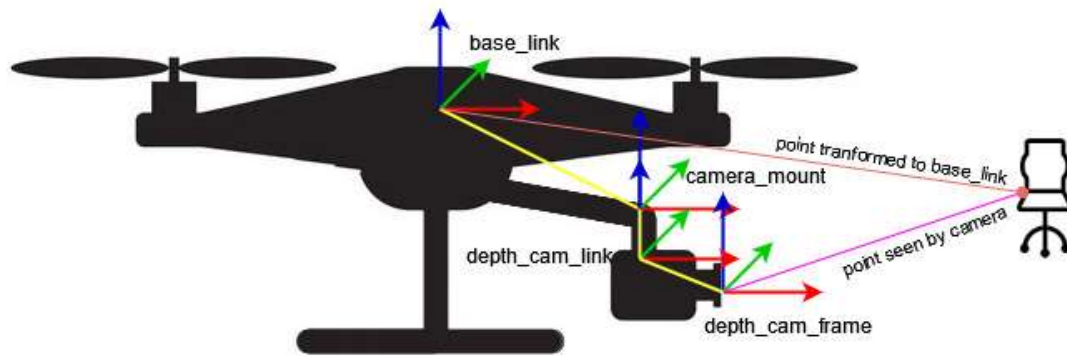


Fig. 46 Transforming the pose of the seen object to the *base_link* frame

3.2.4. Video streaming

In drone piloting, the ability to stream video is an essential aspect. For the drone constructed in this work, the video stream consisted of three types of information: video from the RGB sensor, video from the depth sensor (the depth map), and metadata that included the timestamp and the current position in X, Y, and Z coordinates. However, streaming the generated map in real time was not possible due to the limited speed of the Wi-Fi connection, especially in areas with low signal coverage.

To generate the stream, the GStreamer was used, which provided an RTSP server that ran in a separate thread in the ROS node responsible for processing the sensor data. The frames were read directly from the sensor using the librealSense SDK and then converted to OpenCV type for images. OpenCV was used to combine both frames together and embed metadata, and the final OpenCV frame was fed to the RTSP server, which handled the streaming to the user. The stream could be viewed directly in the ground station or in any video player capable of receiving a network stream, such as VLC.



Fig. 47 Video stream. up) image from RGB sensor, down) image from a depth sensor

The real-time nature of the stream was ensured by achieving a final video stream delay of under 300ms. This delay was measured by re-capturing the video that was being streamed from the monitor, producing an infinity mirror effect with discernable timestamp differences in each sub-image. The Raspberry Pi provided hardware acceleration for H264 encoding, which reduced the load on the main CPU. Streaming took only 7% of one core of the CPU, which demonstrated the system's efficiency.

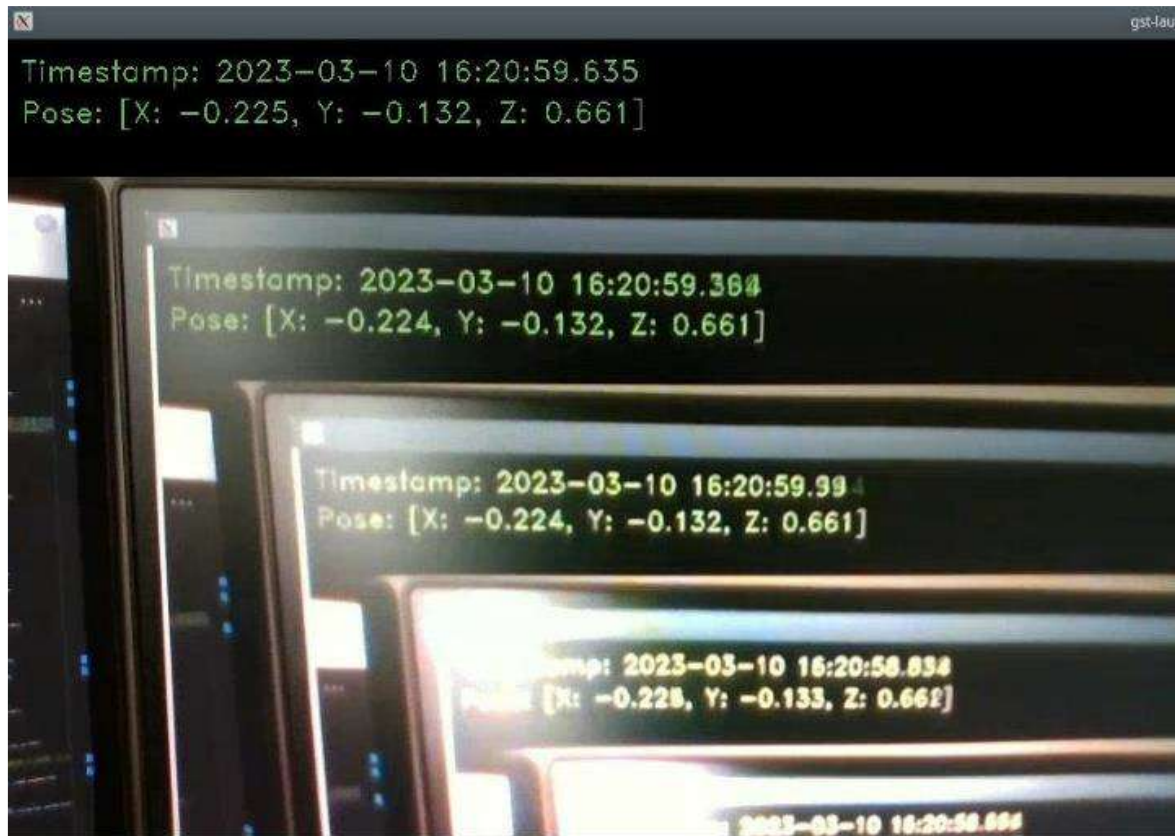


Fig. 48 Measurement of video stream latency

3.2.1. 3D reconstruction from the depth image

The process of creating a 3D map involves the usage of depth images obtained from the depth camera. A depth image is, in essence, a two-dimensional matrix where each pixel corresponds to the distance from the camera to the object. To create a 3D map from these depth images, the depth data needs to be converted into a pointcloud. This is done with RTAB-Map software packages, which can read the depth image and compute the 3D coordinates of each point in the scene [8]. Once the pointcloud is created, the software can use it to generate a 3D reconstruction of the environment. In order to track the motion of the depth camera and align each pointcloud, odometry is taken from a tracking camera. This way, the software can accurately estimate the position and orientation of the depth camera for each captured depth image and combine multiple pointclouds into a single 3D map.

In addition to mapping, RTAB-Map also provides localization capabilities. It uses odometry data from the drone's tracking camera to estimate its position and orientation within the map. This makes it possible for the drone to navigate through the environment and perform tasks based on its current location.

RTAB-Map also incorporates loop closure detection, which helps to correct errors that may have accumulated in the map over time. By identifying areas of the map that the drone had previously visited, RTAB-Map can refine its estimate of the drone's location and improve the accuracy of its map.

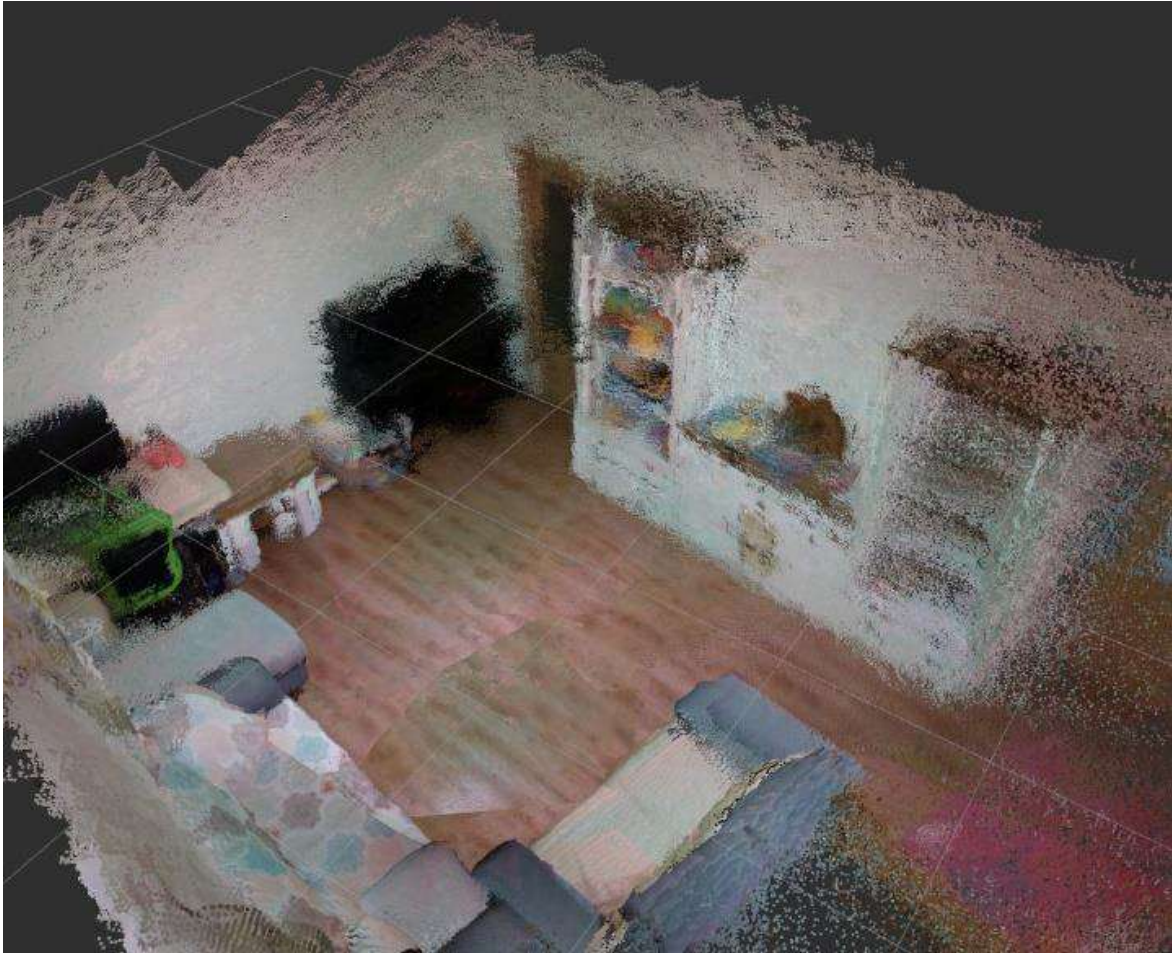


Fig. 49 Point cloud of a room created by RtabMap

3.2.2. Localization and navigation

Localization and navigation are the main components of an autonomous drone system. Prior to localization, a map of the environment must be created. This is done with the help of RTAB-Map, which creates a 3D point cloud map of the environment. The map is built by flying the drone manually over the environment using an RC transmitter or by uploading a pre-existing map to the drone's RTAB-Map database.

The localization is achieved through the use of the tracking camera mounted on the drone. The tracking camera provides real-time visual odometry data that is used to estimate the position and orientation of the drone relative to the environment. This data is fed into the localization component of the navigation stack, which uses it to determine the drone's location in the map.

The tracking camera works by analyzing the images it captures in real-time, using features in the environment to estimate the drone's position and orientation. These features can include edges, corners, or other distinctive patterns in the environment. By tracking these features over time, the camera is able to estimate the drone's movement and adjust its position and orientation accordingly. The IMU provides additional data on the drone's motion, including acceleration and rotation rate, which is used to improve the accuracy of the localization process.

For navigation, the Nav2 ROS2 navigation stack is utilized, which includes the global and local planners, the costmap generator, and the path follower. The costmap generator is responsible for generating a 2D costmap from a specific altitude of the drone, providing information about the free and occupied space in the environment. This costmap is used by the global and local planners to generate a path from the drone's current location to the desired goal location, while avoiding obstacles in the environment.

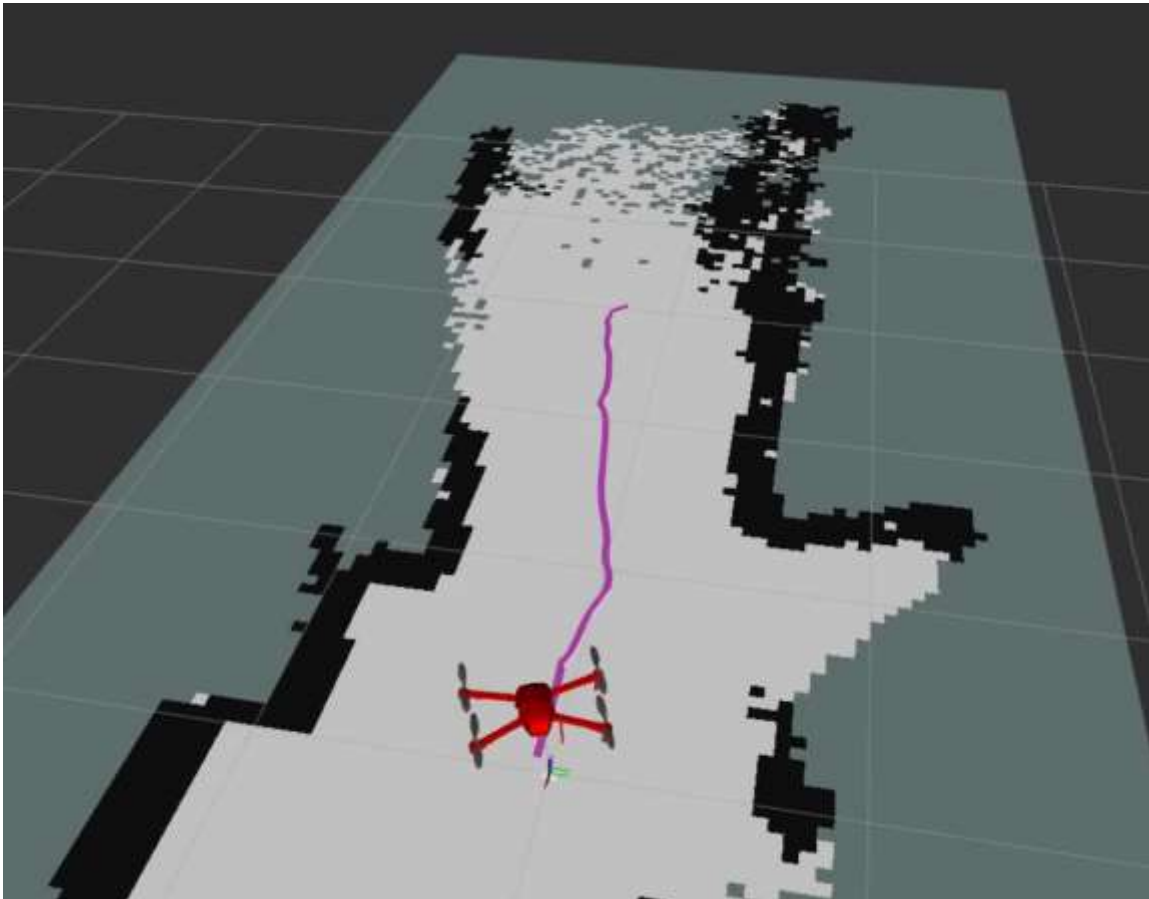


Fig. 50 Navigation and path following using 2D costmap

The global planner, utilizing algorithms such as A* or Dijkstra, takes inputs such as the destination, map, and costmap, and generates a series of waypoints that form a global plan to navigate the drone. This global plan provides a desired path for the drone to follow while taking into account

the environment's obstacles and cost constraints. On the other hand, the local planner uses the global plan and the drone's current pose to generate a local plan that adapts to the surroundings and avoids any obstacles in real-time. The Dynamic Window Approach (DWA) algorithm is used for the local planner to ensure the drone's safety while still maintaining its desired path. Finally, the controller takes the generated local plan and produces control signals that guide the drone along the planned trajectory while maintaining stable flight dynamics.

To allow for this kind of control, the flight controller is switched to offboard mode, which allows for direct control of velocity and direction from mission computer. The Nav2 stack subscribes to the drone's odometry data and transforms it into the global frame of the costmap. The path follower then uses this transformed data to follow the generated path. The entire process is implemented using ROS2 nodes, which communicate with each other using ROS2 messages and services.

3.2.3. Exploration

Exploration is essentially the process of mapping and navigation performed together in order to discover and map new areas while minimizing the time and energy required to do so. Frontier-based exploration is a popular

approach to autonomous exploration in which the robot explores new areas by following the frontier, or the boundary between known and unknown areas. To achieve autonomous exploration, the ROS package `explore_lite` was used, which provides a framework for frontier-based exploration.

The exploration process begins by building a map of the environment using RTAB-Map, which generates a 3D point cloud map of the environment. This map is then converted to a 2D costmap that is used for path planning. The costmap is generated from a specific altitude of the drone to ensure that obstacles are accurately represented.

Once the map and costmap are created, the `explore_lite` package is used to perform frontier-based exploration. This technique works by identifying the frontiers of the known and unknown areas in the environment. A frontier is a boundary between explored and unexplored areas that can be used as a target for the drone to explore further. The package selects a frontier that is closest to the current drone position and plans a path to reach it. The path planning is performed also using the Nav2 ROS2 navigation stack which was already mentioned in chapter 3.2.2.

To make the exploration process more efficient, a stop criteria was also implemented, which is based on the percentage of the map that has been explored. Once the drone has explored a

predefined percentage of the map, it returns to the starting point and the exploration task is completed.

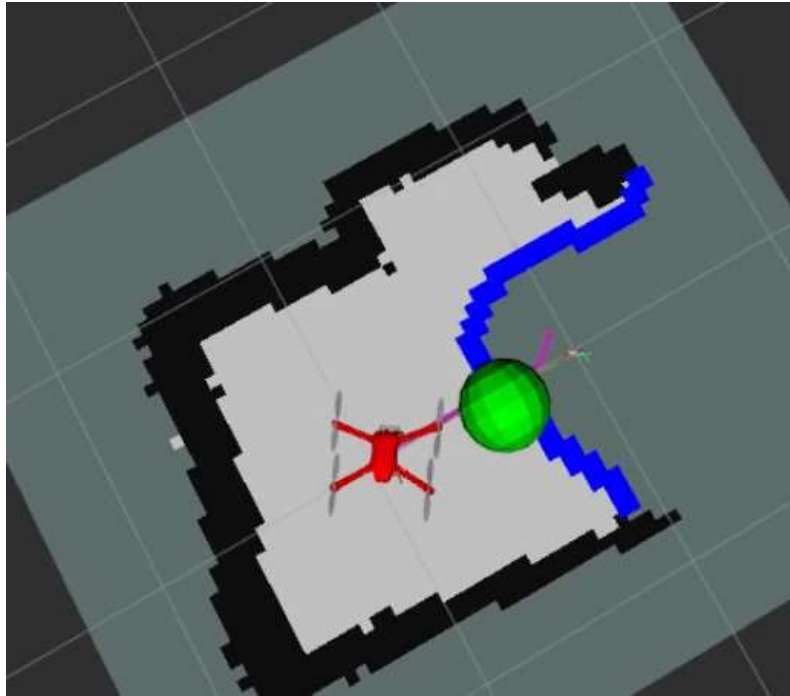


Fig. 51 Start of exploration. Blue line defines frontier which needs to be explored.



Fig. 52 Exploration progress

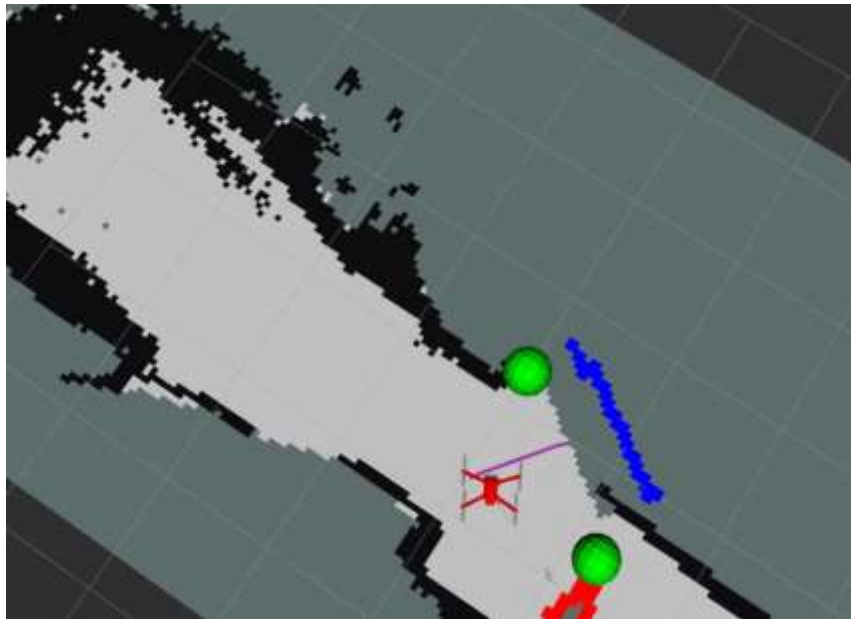


Fig. 53 Exploration progress

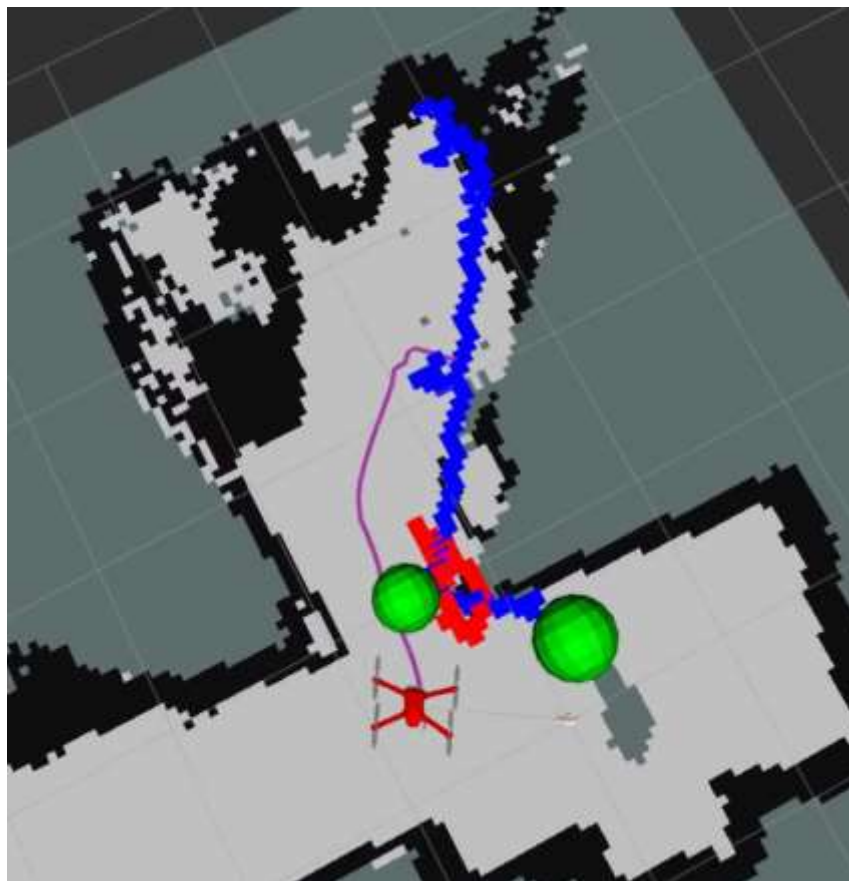


Fig. 54 Exploration progress

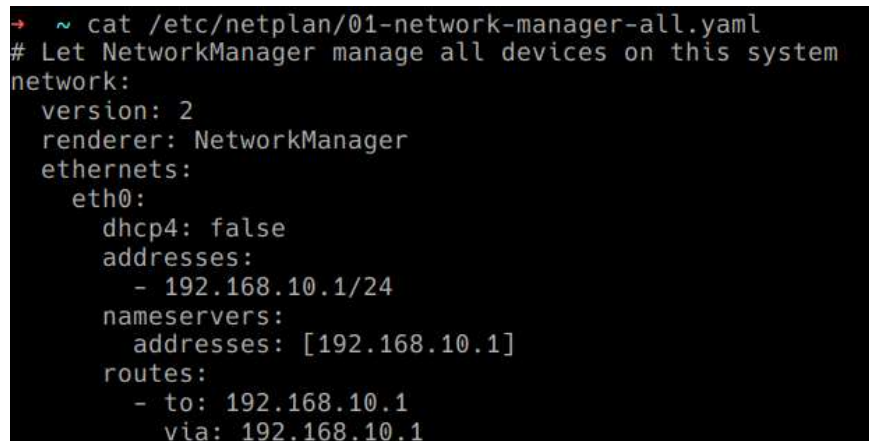
3.3. Flight controller software

3.3.1. Communication with mission computer

One of the crucial aspects of drone control is the ability to directly set its velocity from the mission computer, where the navigation stack is running. This is made possible by switching the flight controller to offboard mode, which enables direct control of the drone's velocity and direction.

To use offboard mode, a connection between the mission computer and flight controller must be established through an Ethernet protocol, which is available on both devices. To enable such communication, a networking configuration must be set up by assigning static IP addresses from the same subnet to both devices.

The mission computer runs on Ubuntu, which allows for network configuration through Network Manager. This can be achieved by configuring the `/etc/netplan/01-network-manager-all.yaml` file and assigning an IP address to the ethernet interface (see Fig. 55). Once the changes are made, the configuration file can be applied by executing the command `"sudo netplan apply"`.



```
→ ~ cat /etc/netplan/01-network-manager-all.yaml
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    eth0:
      dhcp4: false
      addresses:
        - 192.168.10.1/24
      nameservers:
        addresses: [192.168.10.1]
      routes:
        - to: 192.168.10.1
          via: 192.168.10.1
```

Fig. 55 Network configuration file of mission computer

To set IP addresses on the mission computer, a different file needs to be configured because PX4 is based on NuttX, a real-time operating system. The network configuration file is located at `/fs/microsd/net.cfg` on the SD card. This file contains key-value pairs that configure specific network settings (see Fig. 56). Once the file is properly configured, the flight controller must be rebooted for the changes to take effect.


```
[0] % cat net.cfg
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.10.2
NETMASK=255.255.255.0
ROUTER=192.168.10.1
DNS=192.168.10.1
```

Fig. 56 Content of net.cfg file.

Once the network settings and IP addresses for both the mission computer and the flight controller have been configured, it is important to test the communication between them to ensure that they can communicate properly. This can be done using the ping command, which is available on most operating systems. To test the connection, the ping command followed by the IP address of the flight controller was used from the mission computer (see Fig. 57). If the connection is successful, the ping command will send and receive test packets and show the round-trip time, which indicates that the mission computer can communicate with the flight controller.

```
→ ~ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.243 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.314 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=0.209 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.159 ms
^C
--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
```

Fig. 57 Ping from mission computer to flight controller.

3.3.2. Communication middleware

The last requirement for successful communication between the mission computer and flight controller is to establish the application network layer. This was achieved by utilizing the micro-ROS framework, an extension of ROS 2 designed to bring the benefits of ROS 2 to microcontrollers commonly used in embedded systems with limited resources. The framework provides a communication layer similar to ROS 2, which enables nodes to communicate with each other through topics, services, and actions.

The micro-ROS framework uses a middleware layer called the eProsima Micro XRCE-DDS (Data Distribution Service) to handle the communication between the microcontroller and the ROS 2 nodes running on the server. This middleware layer is designed to be lightweight and efficient, making it ideal for use in embedded systems with limited resources. This middleware layer includes microRTPS agent and microRTPS client which are responsible for communication. The

communication between the agent and client is established through a UDP protocol, which allows for the exchange of data packets between the two devices. The agent sends data packets in microRTPS format, which contain information about the ROS 2 topics, services, and actions. The client receives these packets and translates them into uORB messages, which can be processed by the PX4 firmware (see Fig. 58).

To establish communication between the mission computer and the flight controller, the microRTPS agent is configured to use the IP address and port of the microRTPS client running on the flight controller. Once the connection is established, the mission computer and the flight controller can communicate through the topics, services, and actions provided by the ROS2 middleware.

The microRTPS agent enables the mission computer to send and receive data from the flight controller, such as sensor data, commands, and telemetry information. The agent provides a communication layer similar to that of ROS2, which allows the mission computer to publish and subscribe to topics, call services, and use actions to interact with the flight controller.

On the flight controller side, the microRTPS client is responsible for translating the communication from microROS to uORB, which is the messaging and data management system used in PX4. The client receives the data from the microRTPS agent, translates it to uORB messages, and publishes it to the uORB topics. Similarly, it subscribes to the uORB topics, translates the messages to microROS, and sends them back to the microRTPS agent.

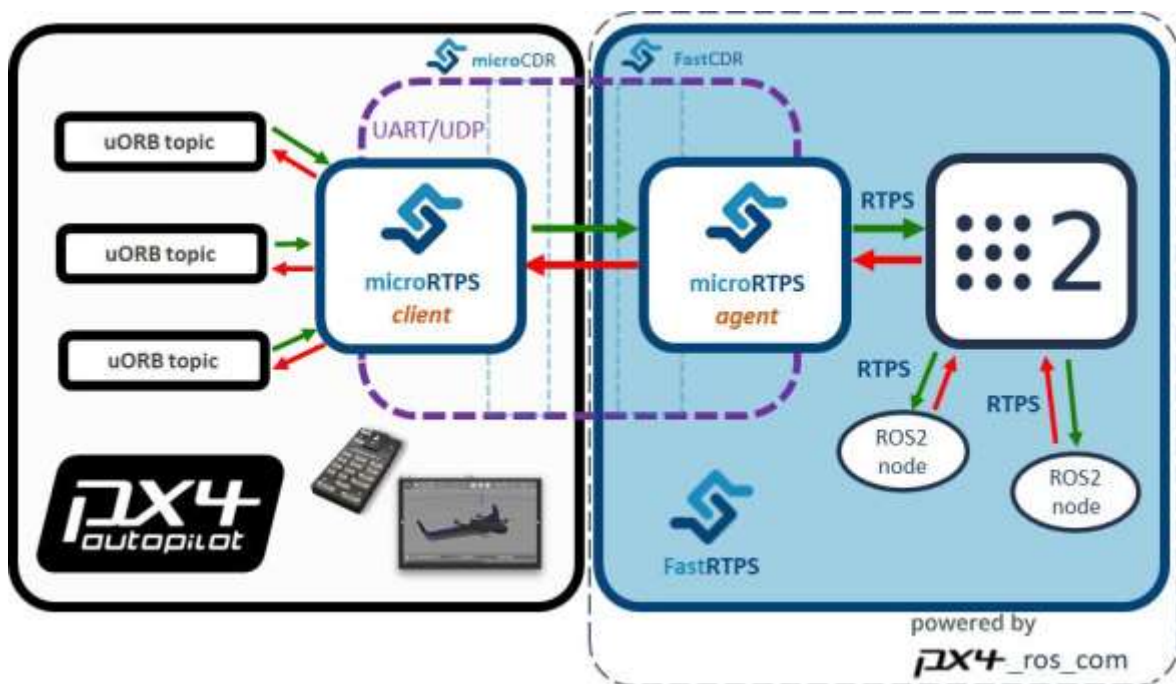


Fig. 58 Relation between the microRTPS client and microRTPS agent.

3.3.3. Offboard mode

Finally, with the possibility of communication with the flight controller using standard ROS topics and services, the offboard mode can be fully utilized. This mode enables more advanced maneuvers, such as following a predefined trajectory, hovering in place, or even executing complex flight patterns.

To fully utilize offboard mode for drone control, a set of steps needs to be followed. The mode can be activated manually from the ground station or directly from the mission computer. Generally, the former is used if the drone is controlled over an RC transmitter, while the latter is used when the drone is fully controlled by the mission computer. The following steps need to be taken to make the drone fly fully from the mission computer:

1. Subscribe to the topic `/fmu/out/vehicle_status` to wait for the current status of the drone.
2. Send an ARM command to `/fmu/in/vehicle_command` to arm the drone.
3. Wait until the vehicle status reports that the drone is armed.
4. Send an offboard mode command to `/fmu/in/vehicle_command` to switch to offboard mode.
5. Wait until the vehicle status reports that the drone is in offboard mode.
6. Continuously publish a heartbeat message to `/fmu/in/offboard_control_mode` to keep offboard mode active.
7. Continuously publish a trajectory setpoint to `/fmu/in/trajectory_setpoint` to make the drone fly.

Because these steps have an asynchronous character, they are governed by a state machine that triggers the next command only when the necessary conditions are met. Additionally, certain values must be checked and published to the flight controller continuously. To ensure precise and responsive control of the drone, the ROS node responsible for communicating with the flight controller operates at a frequency of 50 Hz.

An important aspect of flying a drone is to determine its current position, which is provided by the tracking camera. This position is published to the `/fmu/in/vehicle_position` topic, which is reported back by the vehicle status, allowing the navigation system to determine if the drone has reached the requested position. Additionally, the current pose is crucial for calculating the error between the required position and desired position.

3.3.4. Pose stabilization

Despite the drone operating mainly indoors, external factors such as turbulence caused by the drone's own propellers can still affect the drone's position in a significant way. A stable pose is essential for precise 3D mapping, where drone stability can affect the resulting point cloud quality. Drone position stabilization can be achieved by utilizing a tracking camera and a PID controller located in the Pixhawk flight controller. PID controller is defined as the sum of the proportional component, integral component, and derivative component.

$$u(t) = P(t) + I(t) + D(t) \quad (6)$$

Where $u(t)$ is the correction signal that is sent to the drone's motors at time t .

Each component can be defined as follows:

$$P(t) = K_p e(t) \quad (7)$$

Where $P(t)$ is the output of the proportional component at time t , K_p is the proportional gain, and $e(t)$ is the error at time t .

$$I(t) = K_i \int_0^t e(t) dt \quad (8)$$

Where $I(t)$ is the output of the integral component at time t , K_i is the integral gain, $e(t)$ is the error at time t , and the integral is taken from 0 to t .

$$D(t) = K_d \frac{de(t)}{dt} \quad (9)$$

Where $D(t)$ is the output of the derivative component at time t , K_d is the derivative gain, and $\frac{de(t)}{dt}$ is the rate of change of the error at time t .

The final equation will look like this:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (10)$$

For a 3D pose stabilisation, we need to define the vector for desired position $\vec{p}_{des} = [x_{des} \ y_{des} \ z_{des}]$ and the vector for the current position $\vec{p}_{cur} = [x_{cur} \ y_{cur} \ z_{cur}]$. The current position is provided by the tracking camera of the drone. This position is compared to the

desired position, which produces error vector $\vec{e} = \vec{p}_{des} - \vec{p}_{cur}$. The error is then used by the PID controller to calculate an output vector that is used to adjust the drone's velocity.

To calculate the next correction value, we need to take into account the current error and time which elapsed since the last correction. This will produce the following equation:

$$\vec{u} = K_p \vec{e} + K_i I + K_d D \quad (11)$$

where I and D are the integrated and differentiated error terms.

The integrated error term is calculated as the sum of all the previous errors over time:

$$I = I + \vec{e} dt \quad (12)$$

where dt is the time elapsed since the last calculation. This helps to eliminate steady-state error in the system.

The differentiated error term is calculated as the rate of change of the error:

$$D = \frac{(\vec{e} - \vec{e}_{prev})}{dt} \quad (13)$$

where \vec{e}_{prev} is the error from the previous calculation, this helps to account for sudden changes in the error term and prevent overshooting.

Finally, the output vector \vec{u} is used to adjust the drone's velocity in the x, y and z directions:

$$\vec{u} = \vec{v} = [v_x \quad v_y \quad v_z] \quad (14)$$

Where v_x , v_y and v_z are the velocities in the x , y and z directions, respectively. The drone's position is then updated based on these velocities over a short time interval, and the PID controller is recalculated with the new error vector. By continuously repeating this process, the drone is able to maintain its position and adjust for any changes in the environment or external factors such as wind or turbulence.

3.4. Ground station

The term "ground station" usually refers to a computer or other device used to control a drone or observe various flight values, such as telemetry data, battery levels, GPS coordinates, and sensor readings. Ground stations are typically used in conjunction with flight control software, such as

QGroundControl, to remotely control the drone's flight path and execute various flight commands, such as takeoff, landing, and waypoint navigation. Additionally, custom MAVLink commands can be used to send specific instructions to the drone, such as changing the drone's altitude or sending it to a specific position.

In this project, the mentioned QGroundControl application was used, which is a powerful and widely used ground control station software that allows users to control and monitor their drones. QGroundControl is open source and available for Windows, macOS, and Linux operating systems. It supports a wide range of drone platforms and autopilots, including PX4, ArduPilot, and DJI, making it a versatile option for drone enthusiasts and professionals alike.

However, QGroundControl is intended for outdoor drone flights that use GPS for localization and show the exact position on a map according to the drone's position. Therefore, it is not perfectly suitable for the needs of this project, but it is still an acceptable solution because it provides a variety of useful tools for drone calibration, diagnostics logging, and setting configuration values. It also provides a video viewer that was used to see the video stream from the drone.

The main disadvantage of QGroundControl is that it is unable to show captured point cloud data in real-time. However, this would be possible with RViz tool, but due to the enormous amounts of point cloud data, it is impossible to do real-time. Point cloud data was examined only offline, after the drone was landed, and all data was downloaded from the drone.

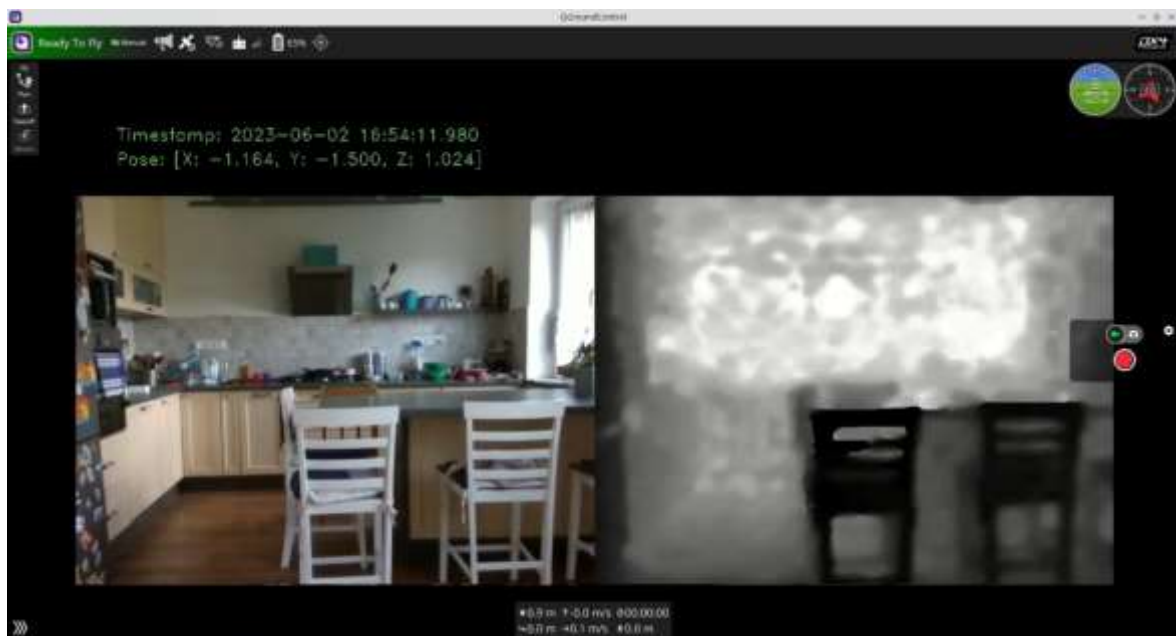


Fig. 59 QGroundControl application with video from a drone

Another tool used from the ground control computer was a simple SSH client. SSH stands for Secure SHell and provides the possibility to execute commands on a remote computer. In this case, the mission computer was connected over a Wi-Fi network, enabling the execution of commands directly on it and triggering various drone commands, such as sending the drone to a specific position or starting the exploration of an unknown environment. This approach was chosen over MAVLink communication because the mission computer should be responsible for the higher logic of drone navigation. Therefore, the mission computer should initiate commands for the flight controller that were requested by the user from the ground station. For example, starting exploration was triggered by calling the ROS2 service `/drone/start_exploration` with a trigger message. On the other hand, stopping exploration can be executed by calling the service `/drone/stop_exploration` with a trigger message.

4. Results

The final solution was tested using a drone, which allowed for scanning in a similar manner to a handheld 3D scanner. The test was conducted indoors on a single floor. Initially, the drone was positioned in the middle of the room and a final check was performed to ensure that all propellers and components were properly fastened. Once confirmed, the drone was connected to the battery and powered on. The ground station was also connected, indicating that the drone was ready for flight. The RC transmitter was also connected, primarily for safety purposes and to maintain control over the drone. With all systems in place, the drone was armed and received a command from the ground station to lift off to a height of one meter above the floor.

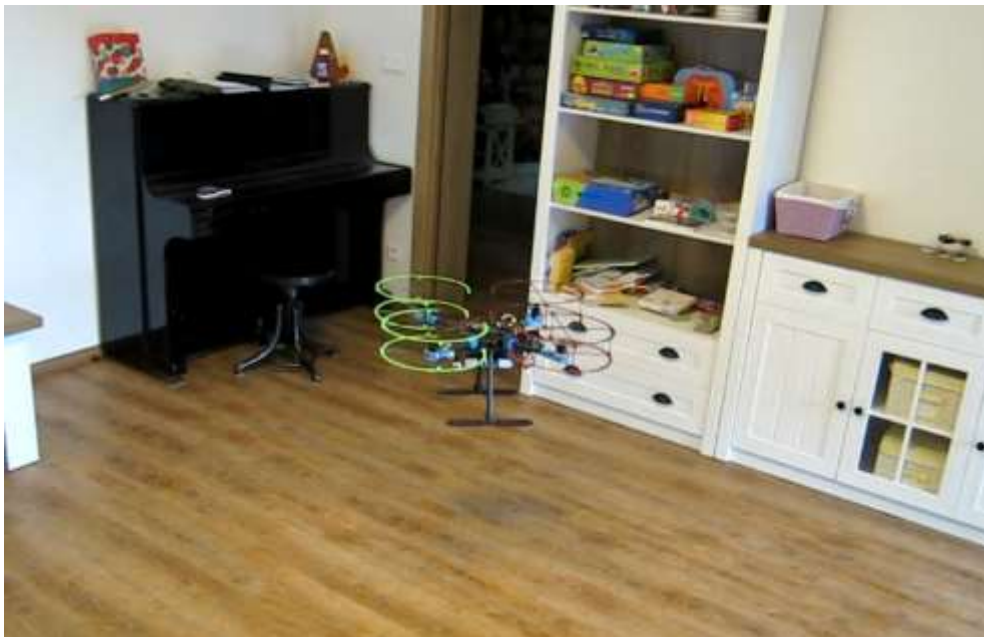


Fig. 60 Flying drone

Once the drone stabilized its position, a full turnaround was performed using the RC transmitter to capture a 3D scan of its immediate surroundings. This enabled the drone to establish its precise localization. Additionally, a connection was established to the mission computer via SSH, allowing for the direct triggering of commands by calling ROS services. Next step involved changing flight mode to offboard mode by calling service `"/drone/start_offboard_mode"`. The exploration was then initiated by triggering the service `"/drone/start_exploration"`, prompting the drone to start flying to new positions. Throughout the exploration, the drone's movements were closely monitored by observing the live video stream and tracking its position within the room. The RC transmitter remained readily available to override any commands and assume manual control if necessary.

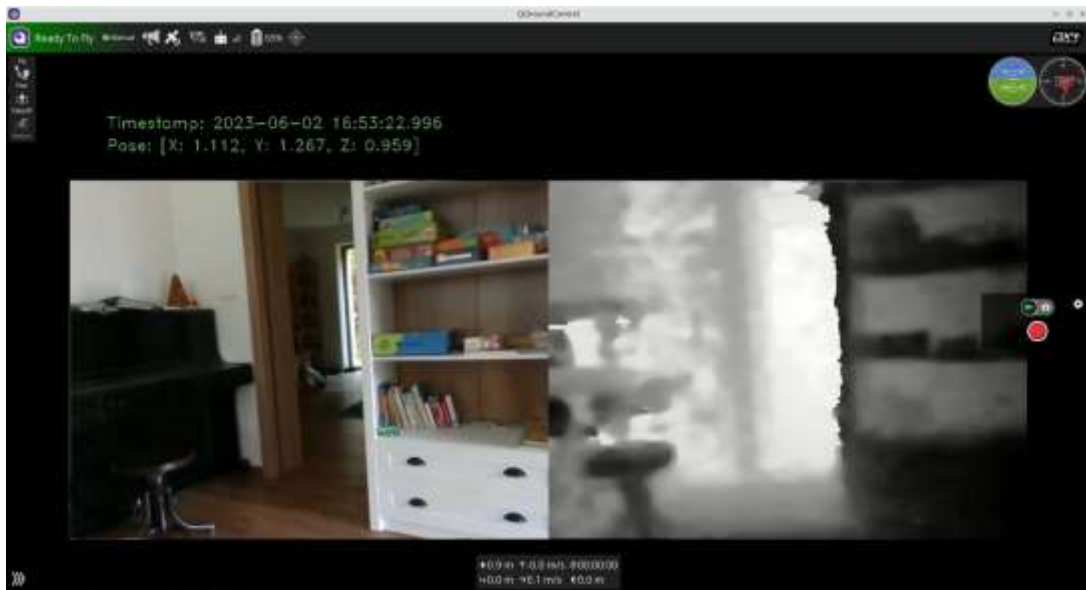


Fig. 61 Stream from drone shown in QGroundControl application

The drone flew slowly to explore new areas and uncover previously unseen environments. After thoroughly exploring all reachable places, the drone returned to its initial position. To initiate the landing process, the drone was switched from offboard mode to landing mode, which triggered a slow descent and a gentle landing on the floor. Upon reaching the ground, the drone automatically disarmed itself. While the drone remained connected to the ground control and the pilot's computer via SSH, the final point cloud data could be downloaded from the drone's flash memory. The testing showcased the capabilities of the drone in conducting efficient and accurate 3D scans of indoor environments.

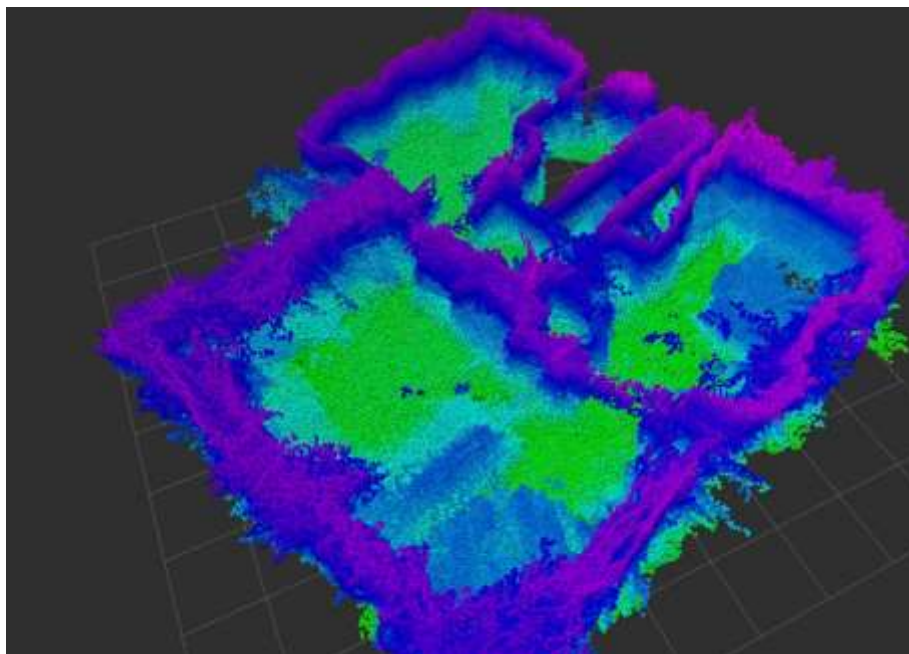


Fig. 62 A point cloud showing the elevation of an indoor environment.

Conclusion

The primary objective of this work was to develop a drone capable of conducting 3D scanning of indoor environments using a depth camera. To accomplish this goal, the project was divided into four distinct theses, each defining specific milestones. The initial phase involved constructing the drone itself. The Holybro X500V2 kit was chosen as the foundation, providing essential components such as the frame, motors, Electronic Speed Controllers (ESC), and propellers. Additional components, including the flight controller, mission computer, battery, Power Distribution Board (PDB), RC transmitter and receiver, and telemetry radio, were integrated to complete the drone assembly. Through meticulous wiring and connection of these components, a functional drone was created, capable of being operated by a pilot and serving as the groundwork for subsequent project phases.

The first thesis focused on creating a 3D reconstruction of an indoor environment using a depth camera mounted on a manually controlled drone. To validate the feasibility of the concept, a handheld 3D scanner was constructed prior to the drone, serving as a testbed for the main sensors. The chosen depth sensing sensor was the Intel RealSense D435i camera, which captures depth images that are subsequently processed by 3D reconstruction algorithms. Additionally, the Intel RealSense T265 tracking camera was employed in conjunction with the depth camera to determine the drone's position. The tracking camera utilizes Visual Simultaneous Localization and Mapping (V-SLAM) algorithms to track its movement within the environment. During this phase, the drone remained under the pilot's control, allowing them to navigate the drone through space while the data from the depth camera and tracking camera was processed in real-time using the RTAB-Map program. This program generated a complete 3D reconstruction of the surrounding environment. All data processing was carried out on an overclocked Raspberry Pi 4B with 8 GB of RAM running Linux Ubuntu Mate 20.04. The Robot Operating System version 2 (ROS2) and the RTAB-Map package were employed for data processing, resulting in a comprehensive 3D reconstruction of the environment.

The second thesis was accomplished by implementing the Nav2 navigation stack, a key component of ROS2. This stack comprises a global and local path planner, which work together to generate a path within a 2D costmap. The costmap is derived from the captured point cloud data acquired at a specific altitude and serves as the basis for path planning. To execute the planned path, a controller algorithm from Nav2 was employed to send velocity commands to the flight controller. The flight controller was connected to the mission computer via an Ethernet cable,

utilizing the microROS agent and client for bidirectional message exchange between the two. The desired goal position was directly sent to the mission computer as X, Y, and Z coordinates.

The third thesis combines the concepts explored in the previous two. To achieve this, the `explore_lite` ROS2 package was utilized. This package implements frontier-based exploration, where it identifies frontiers on a 2D costmap and plans destinations to reach them. As the frontiers are explored, new regions of the map are uncovered, leading to the complete exploration of the reachable space. However, it should be noted that the exploration process did not always function as expected. In some cases, the `explore_lite` package planned trajectories that traversed unexplored areas, resulting in movement into regions that were later identified as enclosed. To address this issue, the drone remained connected to an RC transmitter, allowing the pilot to override the drone's flight trajectory when necessary. Potential solutions to this challenge include improving the `explore_lite` package, exploring alternative exploration algorithms, or developing a custom exploration program tailored to the specific requirements of the project.

The final thesis aimed to develop a ground station desktop program for controlling the drone. Initially, the plan was to use the Unity 3D engine to create an application that would establish a connection with the mission computer over Wi-Fi and exchange data and commands with the drone. However, this approach was not feasible due to poor Unity support for ROS. Unity primarily communicates using the TCP protocol, which is not suitable for handling large amounts of data. An alternative solution was to utilize RViz, a standard robot visualization tool used in ROS. RViz was utilized during development; however, using it over Wi-Fi and for real-time visualization of the point cloud proved challenging due to the large size of the data. Consequently, a third solution was implemented, utilizing a standard ground station application designed for use with the PX4 firmware. This ground station application facilitated communication with the drone via a telemetry radio, while also providing the capability to visualize the video stream received over Wi-Fi. To enable the pilot to have a real-time visual perspective, the drone was programmed to stream video from the RGB sensor of the depth camera, as well as the depth image itself. Additionally, the pilot's PC, connected to the same Wi-Fi network as the drone, also served as an ground station, allowing for direct communication with the mission computer via SSH. This proved to be highly useful during development and flight operations, enabling the pilot to send various commands to ROS directly from the terminal.

The author considers the main objectives of this work to have been achieved, although certain challenges still remain. These unresolved issues can be addressed in future research endeavors. One primary concern is the limitation of 2D navigation in a 3D operating space. To overcome this

limitation, the exploration of 3D navigation packages for ROS2 is essential. A potential solution is to explore the capabilities of the MoveIt package, primarily used for 3D navigation of robotic arms, which may also facilitate trajectory planning in larger spaces. Another noteworthy issue pertains to the ground station and real-time visualization of captured point cloud data. To address this, it is necessary to develop a more efficient communication protocol to facilitate seamless communication with the ROS framework over Wi-Fi, while also minimizing data transmission. Lastly, the design and construction of the drone itself should be re-evaluated to reduce its size and weight, thereby enhancing its performance in indoor environments. Addressing these challenges will pave the way for future advancements in the field of indoor 3D scanning using drones.

In summary, the integration of various components and technologies in this work brings forth a novel solution for indoor 3D scanning using drones. By combining affordable hardware and software, the developed system provides a versatile and efficient approach to capturing detailed 3D representations of indoor environments. Autonomous navigation, scanning, and reconstruction capabilities enable a wide range of applications, including mapping, surveillance, inspection, and more.

Benefit for the further development of science and technology

The contributions of this work to the field of UAV are as follows:

1. Designing a quadrotor drone using affordable components, including sensors, actuators, power sources, and control units.
2. Developing software for drone navigation in GPS-denied environments, utilizing the ROS framework.
3. Creating simulations for drone navigation based solely on input from the depth camera.
4. Exploring the capabilities of 2D exploration algorithms in a 3D space.
5. Processing depth images to generate 3D reconstructions of indoor environments using the RtabMap algorithm.
6. Processing depth and RGB images from the depth camera to create a video stream for the ground station.
7. Establishing communication between the mission computer, responsible for higher-level navigation logic, and the flight controller, the embedded system for drone control.

References

- [1] H. Chen, X. -m. Wang and Y. Li, "A Survey of Autonomous Control for UAV," 2009 International Conference on Artificial Intelligence and Computational Intelligence, Shanghai, China, 2009, pp. 267-271, doi: 10.1109/AICI.2009.147.
- [2] Yuncheng Lu, Zhucun Xue, Gui-Song Xia & Liangpei Zhang (2018) A survey on vision-based UAV navigation, *Geo-spatial Information Science*, 21:1, 21-32, DOI: 10.1080/10095020.2017.1420509
- [3] Hadi, Ghazali S., et al. "Autonomous UAV system development for payload dropping mission." *The Journal of Instrumentation, Automation and Systems* 1.2 (2014): 72-22.
- [4] T. Tomic et al., "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," in *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 46-56, Sept. 2012, doi: 10.1109/MRA.2012.2206473.
- [5] S. Chen, H. Chen, C. -W. Chang and C. -Y. Wen, "Multilayer Mapping Kit for Autonomous UAV Navigation," in *IEEE Access*, vol. 9, pp. 31493-31503, 2021, doi: 10.1109/ACCESS.2021.3055066.
- [6] X. Wang, L. Ma, D. Wang and B. Su, "Autonomous UAV-Based Position and 3D Reconstruction of Structured Indoor Scene," *2021 China Automation Congress (CAC)*, Beijing, China, 2021, pp. 3914-3919, doi: 10.1109/CAC53003.2021.9727567.
- [7] Ayoub, N.; Schneider-Kamp, P. "Real-Time On-Board Deep Learning Fault Detection for Autonomous UAV Inspections." *Electronics* 2021, 10, 1091. Doi: 10.3390/electronics10091091
- [8] D. Kang, Y. J. Cha, "Autonomous UAVs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging." *Computer-Aided Civil and Infrastructure Engineering* 33.10 (2018): 885-902.
- [9] C. Koparan, A. B. Koc, C. V. Privette, C. B. Sawyer, "Autonomous In Situ Measurements of Noncontaminant Water Quality Indicators and Sample Collection with a UAV". *Water* 2019, 11, 604. doi: 10.3390/w11030604
- [10] S. Zhao et al., "A Robust Real-Time Vision System for Autonomous Cargo Transfer by an Unmanned Helicopter," in *IEEE Transactions on Industrial Electronics*, vol. 62, no. 2, pp. 1210-1219, Feb. 2015, doi: 10.1109/TIE.2014.2345348.
- [11] H. Qin et al., "Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-Denied Environments," in *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1339-1350, Feb. 2019, doi: 10.1109/TVT.2018.2890416.

-
- [12] X. Chen, Q. Wu and S. Wang, "Research on 3D Reconstruction Based on Multiple Views," 2018 13th International Conference on Computer Science & Education (ICCSE), 2018, pp. 1-5, doi: 10.1109/ICCSE.2018.8468705.
- [13] W. -H. A. Wang and Man-Ching Lin, "A fast method in reconstruction 3D computed tomography medical images," 2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management, 2010, pp. 1877-1881, doi: 10.1109/ICIEEM.2010.5645895.
- [14] X. Yang et al., "Towards Automated Semantic Segmentation in Prenatal Volumetric Ultrasound," in IEEE Transactions on Medical Imaging, vol. 38, no. 1, pp. 180-193, Jan. 2019, doi: 10.1109/TMI.2018.2858779.
- [15] S. Riehemann, M. Palme, P. Kuehmstedt, C. Grossmann, G. Notni and J. Hintersehr, "Microdisplay-Based Intraoral 3D Scanner for Dentistry," in Journal of Display Technology, vol. 7, no. 3, pp. 151-155, March 2011, doi: 10.1109/JDT.2010.2096799.
- [16] S. Yang and Y. Fan, "3D Building Scene Reconstruction Based on 3D LiDAR Point Cloud," 2017 IEEE International Conference on Consumer Electronics – Taiwan (ICCE-TW), 2017, pp. 127-128, doi: 10.1109/ICCE-China.2017.7991028.
- [17] J. P. Lavelle, S. R. Schuet and D. J. Schuet, "High speed 3D scanner with real-time 3D processing," ISA/IEEE Sensors for Industry Conference, 2004. Proceedings the, 2004, pp. 102-108, doi: 10.1109/SFICON.2004.1287140.
- [18] K. Kim, J. Kim, S. Kang, J. Kim and J. Lee, "Vision-based bin picking system for industrial robotics applications," 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2012, pp. 515-516, doi: 10.1109/URAI.2012.6463057.
- [19] H. B. Suay and S. Chernova, "Humanoid robot control using depth camera," 2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2011, pp. 401-401, doi: 10.1145/1957656.1957802.
- [20] N. A. Zainuddin, Y. M. Mustafah, Y. A. M. Shawgi and N. K. A. M. Rashid, "Autonomous Navigation of Mobile Robot Using Kinect Sensor," 2014 International Conference on Computer and Communication Engineering, 2014, pp. 28-31, doi: 10.1109/ICCCE.2014.21.
- [21] Hitomi E., Silva J., Ruppert G. (2015) 3D Scanning Using RGBD Imaging Devices: A Survey. In: Tavares J., Natal Jorge R. (eds) Developments in Medical Image Processing and Computational Vision. Lecture Notes in Computational Vision and Biomechanics, vol 19. Springer, Cham. https://doi.org/10.1007/978-3-319-13407-9_22
- [22] C. Lee, H. Song, B. Choi and Y. -S. Ho, "3D scene capturing using stereoscopic cameras and a time-of-flight camera," in IEEE Transactions on Consumer Electronics, vol. 57, no. 3, pp. 1370-1376, August 2011, doi: 10.1109/TCE.2011.6018896.
-

- [23] M. Shibata and T. Honma, "3D object tracking on active stereo vision robot," 7th International Workshop on Advanced Motion Control. Proceedings (Cat. No.02TH8623), 2002, pp. 567-572, doi: 10.1109/AMC.2002.1026983.
- [24] R. Noll and M. Krauhausen, "Multibeam laser triangulation for the measurement of geometric features of moving objects in production lines," 2003 Conference on Lasers and Electro-Optics Europe (CLEO/Europe 2003) (IEEE Cat. No.03TH8666), 2003, pp. 468-, doi: 10.1109/CLEOE.2003.1313531.
- [25] I. Anwar and S. Lee, "High performance stand-alone structured light 3D camera for smart manipulators," 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2017, pp. 192-195, doi: 10.1109/URAI.2017.7992709.
- [26] M. Atif and S. Lee, "Adaptive Pattern Resolution for Structured Light 3D Camera System," 2018 IEEE SENSORS, 2018, pp. 1-4, doi: 10.1109/ICSENS.2018.8589640.
- [27] O. Choi et al., "Range unfolding for Time-of-Flight depth cameras," 2010 IEEE International Conference on Image Processing, 2010, pp. 4189-4192, doi: 10.1109/ICIP.2010.5651383.
- [28] C. Anand, M. Sarkar and K. Jainwal, "A Three-Phase, One-Tap High Background Light Subtraction Time-of-Flight Camera," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1-1, doi: 10.1109/ISCAS45731.2020.9180523.
- [29] O. Choi and S. Lee, "Wide range stereo time-of-flight camera," 2012 19th IEEE International Conference on Image Processing, 2012, pp. 557-560, doi: 10.1109/ICIP.2012.6466920.
- [30] D. Bastos, P. P. Monteiro, A. S. R. Oliveira and M. V. Drummond, "An Overview of LiDAR Requirements and Techniques for Autonomous Driving," 2021 Telecoms Conference (ConfTELE), 2021, pp. 1-6, doi: 10.1109/ConfTELE50222.2021.9435580.
- [31] D. V. Nam and K. Gon-Woo, "Solid-State LiDAR based-SLAM: A Concise Review and Application," 2021 IEEE International Conference on Big Data and Smart Computing (BigComp), 2021, pp. 302-305, doi: 10.1109/BigComp51126.2021.00064.
- [32] Karel Van Acoleyen, Hendrik Rogier, and Roel Baets, "Two-dimensional optical phased array antenna on silicon-on-insulator," Opt. Express 18, 13655-13660 (2010)
- [33] Rostami, R.; Bashiri, F.S.; Rostami, B.; Yu, Z. A Survey on Data-Driven 3D Shape Descriptors. Computer Graphics Forum 2018, 00, 1–38.
- [34] Kazmi, I.K.; You, L.; Zhang, J.J. A survey of 2D and 3D shape descriptors. In Proceedings of the 10th International Conference Computer Graphics, Imaging, and Visualization (CGIV); IEEE, 2013; pp. 1–10.
- [35] R.B Rusu and S. Cousins "3D is here: Point Cloud Library (PCL)" Proc. IEEE Int. Conf. Robot. Autom. pp. 1-4 May 2011.

-
- [36] Houshiar, H. Documentation and mapping with 3D point cloud processing, University of Würzburg, 2012.
 - [37] K. Mamou and F. Ghorbel, "A simple and efficient approach for 3D mesh approximate convex decomposition," 2009 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 3501-3504, doi: 10.1109/ICIP.2009.5414068.
 - [38] Shin, D.; Fowlkes, C.C.; Hoiem, D. Pixels, voxels, and views: A study of shape representations for single-view 3D object shape prediction. 2018.
 - [39] I. Ideses, L. Yaroslavsky and B. Fishbain, "Depth Map Manipulation for 3D Visualization," 2008 3DTV Conference: The True Vision – Capture, Transmission and Display of 3D Video, 2008, pp. 337-340, doi: 10.1109/3DTV.2008.4547877.
 - [40] Hitomi E., Silva J., Ruppert G. (2015) 3D Scanning Using RGBD Imaging Devices: A Survey. In: Tavares J., Natal Jorge R. (eds) Developments in Medical Image Processing and Computational Vision. Lecture Notes in Computational Vision and Biomechanics, vol 19. Springer, Cham. https://doi.org/10.1007/978-3-319-13407-9_22
 - [41] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira and V. Teichrieb, "Analysis and Comparison of Robotics 3D Simulators," *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, 2019, pp. 242-251, doi: 10.1109/SVR.2019.00049.
 - [42] OGRE 3D engine, available: <https://www.ogre3d.org/>
 - [43] Gazebo simulation tool, available: <http://gazebo-sim.org/>
 - [44] 3D visualization tool for ROS, available: <http://wiki.ros.org/rviz>
 - [45] Unity – Robotics simulation, available: <https://unity.com/solutions/automotive-transportation-manufacturing/robotics>
 - [46] P. Mirowski, T. K. Ho, Saehoon Yi and M. MacDonald, "SignalSLAM: Simultaneous localization and mapping with mixed WiFi, Bluetooth, LTE and magnetic signals," International Conference on Indoor Positioning and Indoor Navigation, 2013, pp. 1-10, doi: 10.1109/IPIN.2013.6817853.
 - [47] S. Quan and J. Chen, "AGV Localization Based on Odometry and LiDAR," 2019 2nd World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM), Shanghai, China, 2019, pp. 483-486, doi: 10.1109/WCMEIM48965.2019.00102.
 - [48] Y. Chen, Y. Wu and H. Xing, "A complete solution for AGV SLAM integrated with navigation in modern warehouse environment," 2017 Chinese Automation Congress (CAC), Jinan, 2017, pp. 6418-6423, doi: 10.1109/CAC.2017.8243934.
-

- [49] M. Labbé and F. Michaud, "RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation," in *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. (Wiley)
- [50] S. Pütz T. Wiemann J. Sprickerhof and J. Hertzberg "3d Navigation Mesh Generation for Path Planning in Uneven Terrain" *IFAC-PapersOnLine* vol. 49 no. 15 pp. 212-217 2016.
- [51] Kang-Min Lee Dong-Kyun Lim Kyung-Geun Kim and Byung-suhl Suh "A Wall-Following Method of Mobile Robot for Mapping" *KIEE Information and Control Systems Symposium* pp. 102-105 May. 2005.
- [52] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997, pp. 146–151.
- [53] L. Freda, G. Oriolo, and F. Vecchioli, "Sensor-based exploration for general robotic systems," 2008, pp. 2157–2164.
- [54] G. A. Lopes, E. Najafi, S. P. Nagesh Rao, and R. Babuska, "Learning complex behaviors via sequential composition and passivity-based control," in *Handling Uncertainty and Networked Structure in Robot Control*, L. Busoniu and L. Tamas Eds. Springer, 2015, pp. 53–74.
- [55] C. Zhu, R. Ding, M. Lin and Y. Wu, "A 3D Frontier-Based Exploration Tool for MAVs," 2015 *IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2015, pp. 348-352, doi: 10.1109/ICTAI.2015.60.
- [56] L. Quan, L. Han, B. Zhou, S. Shen, F. Gao, "Survey of UAV motion planning." *IET Cyber-systems and Robotics*, 2020, 2(1), 14-21.
- [57] M. Torres, D. A. Pelta, J. L. Verdegay, J. C. Torres, "Coverage path planning with unmanned aerial vehicles for 3D terrain reconstruction." *Expert Systems with Applications*, 2016, 55, 441-451.
- [58] Intel Realsense Depth Camera D435i, available: <https://www.intelrealsense.com/depth-camera-d435i/>
- [59] Intel Realsense Tracking Camera T265, available: <https://www.intelrealsense.com/tracking-camera-t265/>
- [60] Raspberry Pi 4, available: Raspberry Pi 4
- [61] Robot Operating System, available: <http://wiki.ros.org/>
- [62] X500V2, available: <https://holystone.com/products/px4-development-kit-x500-v2>
- [63] Pixhawk 6x, available: <http://www.holystone.com/product/pixhawk-6x/>

List of the author's publications

- [1] Phollower—The Universal Autonomous Mobile Robot for Industry and Civil Environments with COVID-19 Germicide Addon Meeting Safety Requirements: Ján Bačík, Stanislav Alexovič ... [et al.] Spôsob prístupu: <https://doi.org/10.3390/app10217682...> - 2020. In: Applied sciences. - Basel (Švajčiarsko) : Multidisciplinary Digital Publishing Institute Roč. 10, č. 21 (2020), s. [1-16] [online]. - ISSN 2076-3417 (online)
- [2] Introduction into Autonomous Mobile Robot Research and Multi Cooperation: Stanislav Alexovič ... [et al.] Spôsob prístupu: https://doi.org/10.1007/978-3-030-77445-5_30... - 2021. In: Artificial Intelligence in Intelligent Systems : proceedings of 10th Computer Science On-line Conference 2021, Vol. 2. - Cham (Švajčiarsko) : Springer s. 326-336 . - ISBN 978-3-030-77444-8 - ISSN 2367-3370
- [3] An overview of Autonomous Mobile Robot research and multi cooperation: Stanislav Alexovič - 2021. In: 21st Scientific Conference of Young Researchers. - Košice (Slovensko) : Technická univerzita v Košiciach s. 75-78 [online, CD-ROM]. - ISBN 978-80-553-3904-7
- [4] 3D Scanning of the Indoor Environment: Stanislav Alexovič, Milan Lacko Spôsob prístupu: http://scyr.kpi.fei.tuke.sk/wp-content/scyr-files/proceedings/SCYR_2022_Proceedings.pdf... - 2022. In: 22nd Scientific Conference of Young Researchers : proceedings from conference. - Košice (Slovensko) : Technická univerzita v Košiciach s. 98-99 [CD-ROM, print]. - ISBN 978-80-553-4061-6
- [5] Handheld 3D Scanner Based on Intel RealSense Depth and Tracking Cameras / Stanislav Alexovič ... [et al.] Spôsob prístupu: http://dx.doi.org/10.1007/978-3-031-09076-9_22... - 2022. In: Artificial Intelligence Trends in Systems. - Cham (Švajčiarsko) : Springer Nature s. 226-235 [online]. - ISBN 978-3-031-09075-2 - ISSN 2367-3370
- [6] Simulation of multiple autonomous mobile robots using a gazebo simulator: Stanislav Alexovič, Milan Lacko, Ján Bačík Spôsob prístupu: http://dx.doi.org/10.1007/978-3-031-21435-6_30... - 2023. In: Software Engineering Application in Systems Design : Proceedings of 6th Computational Methods in Systems and Software 2022, Volume 1. - Cham (Švajčiarsko) : Springer International Publishing AG s. 339-351 . - ISBN 978-3-031-21434-9 - ISSN 2367-3370
- [7] 3D Scanning of the Indoor Environment using a quadrotor drone: Stanislav Alexovič Spôsob prístupu: http://scyr.kpi.fei.tuke.sk/wp-content/scyr-files/history/SCYR_2023_Proceedings.pdf... - 2023. In: 23rd Scientific Conference of Young Researchers : proceedings from conference. - Košice (Slovensko) : Technická univerzita v Košiciach s. 31-32 . - ISBN 978-80-553-4377-8

- [8] 3D mapping with a drone equipped with a depth camera in indoor environment: Stanislav Alexovič. Accepted to the journal Acta Electrotechnica et Informatica. 2023
- [9] 3D mapovanie dronom vybaveným hĺbkovou kamerou v prostredí bez prístupu GPS navigácie: Stanislav Alexovič. Prijaté do časopisu QuoVadis

The author participated in the APVV-18-0436 project granted by Slovak Research and Development Agency.

Appendix

Appendix A CD medium – dissertation in electronic form, sources codes in electronic form