Universidade de Aveiro
2024

Jorge Miguel Ferreira da Silva

Integration of GNSS and LiDAR to perform the georeferenced measurement of outdoor volumes using ATLASCAR2

Integração de GNSS e LiDAR para medição georreferenciada de volumes exteriores usando o ATLASCAR2

Universidade de Aveiro
2024

Jorge Miguel Ferreira da Silva

Integration of GNSS and LiDAR to perform the georeferenced measurement of outdoor volumes using ATLASCAR2

Integração de GNSS e LiDAR para medição georreferenciada de volumes exteriores usando o ATLASCAR2

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado com Agregação.

o júri / the jury

presidente / president	**Prof. Doutor César Miguel de Almeida Vasques** Professor Auxiliar em Regime Laboral da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Paulo Miguel de Jesus Dias** Professor Auxiliar com Agregação da Universidade de Aveiro
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado com Agregação da Universidade de Aveiro

agradecimentos / acknowledgements

Um especial agradecimento ao professor Vítor Santos pelo apoio, rigor e exigência colocados na orientação deste trabalho.
Ao Engenheiro Rui Heitor que se mostrou sempre disponível para me ajudar.
Agradeço aos meus pais e amigos por todo o apoio.

keywords ROS; LIDAR; GNSS; ATLASCAR2; Volume Calculation; Point Cloud

abstract The industrial task of monitoring and measuring stockpile volumes is traditionally complex, relying on manual measurements that pose safety risks, lead to financial misestimations, and disrupt supply chain management. This dissertation addresses these challenges by developing a robust application within the Robot Operating System (ROS) framework. The primary objective is to automate the presentation, representation, and processing of LIDAR and GNSS detection data. By leveraging these advanced technologies, the proposed system enhances accuracy and efficiency in volume assessment, thus improving safety and operational efficiency in industrial environments. The implementation of this system is detailed, demonstrating significant advancements in automating stockpile volume monitoring and measurement.

palavras-chave ROS; LIDAR; GNSS; ATLASCAR2; Cálculo de Volumes; Nuvem de pontos

resumo A tarefa industrial de monitorar e medir volumes de pilhas de materiais é tradicionalmente complexa, dependendo de medições manuais que representam riscos à segurança, levam a estimativas financeiras imprecisas e interrompem a gestão da cadeia de suprimentos. Esta dissertação aborda esses desafios desenvolvendo uma aplicação robusta no framework Robot Operating System (ROS). O objetivo principal é automatizar a apresentação, representação e processamento de dados de detecção de LIDAR e GNSS. Ao aproveitar essas tecnologias avançadas, o sistema proposto melhora a precisão e eficiência na avaliação de volumes, aprimorando assim a segurança e a eficiência operacional em ambientes industriais. A implementação deste sistema é detalhada, demonstrando avanços significativos na automação do monitoramento e medição de volumes de pilhas de materiais.

List of Acronyms

CAD	Computer Aided Design
CAN	Controller Area Network
COLLADA	collaborative Design Activity
DAE	Digital Asset Exchange
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
JPEG	Joint Photographic Experts Group
LIDAR	Light Detection and Ranging
PCD	Point Cloud Data
PCL	Point Cloud Library
PCM	Point Cloud Measurements
PNG	Portable Network Graphics
RANSAC	Random Sample Consensus
ROR	Radius Outlier Removal
ROS	Robot Operating System
SDF	Simulation Description Format
SFM	Structure From Motion
SOR	Statistical Outlier Removal
SPAN	Synchronous Position, Attitude and Navigation
TLS	Terrestrial Laser Scanning

UAV	Unmanned Aerial Vehicle
URDF	Unified Robot Description Format
VGF	Voxel Grid Filtering
PTF	Passthrough Filtering

Contents

List of Tables

List of Figures

Intentionally blank page.

Listings

Intentionally blank page.

Chapter 1

Introduction

The measurement of volumes in temporary storage areas such as harbors, squares, logistics parks or other similar contexts in outdoor environments is currently performed mostly by manual human operations and besides being tiring and slow, it is also an imprecise process and implies labor costs dedicated to the task. These problems happen because measurements are most often carried out by manual measurement, using a measuring tape, and in some situations the surfaces are not physically delimited, suffering frequent variations. One of the most challenging situations, like the one represented in figure 1.1, is the storage of inert materials in bulk (sand, stone, etc.) which are arranged in piles or heaps, whose volume varies daily, and their measurement is complex.

In this dissertation, it is intended to find an alternative solution for the problems mentioned before, and that can be integrated in ATLASCAR2.

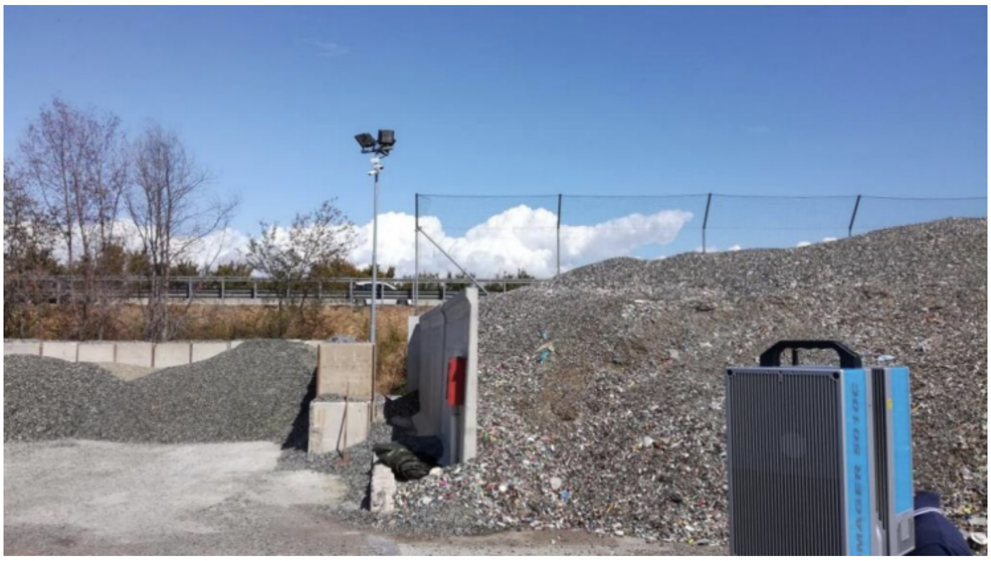


Figure 1.1: Example of a logistic park. [14]

1.1 ATLAS Project

The ATLAS Project was created by the Group of Automation and Robotics at the Department of Mechanical Engineering of the University of Aveiro. The main objective of this project is to develop and enable the application of advanced sensors and active systems designed for implementation in automobiles and platforms.

ATLASCAR2 [2] is part of this project, and it is equipped with several Light Detection and Ranging (LIDAR) sensors, Cameras, Global Navigation Satellite System (GNSS) and Inertial Navigation System (INS) receiver. It is also an all-electric vehicle, unlike the ATLASCAR1 [1], which means it is easier to modify, test and control unlike the previous version. In figure 1.2 it is possible to see the vehicle with several sensors mounted such as LIDAR sensors, cameras and Global Positioning System (GPS).

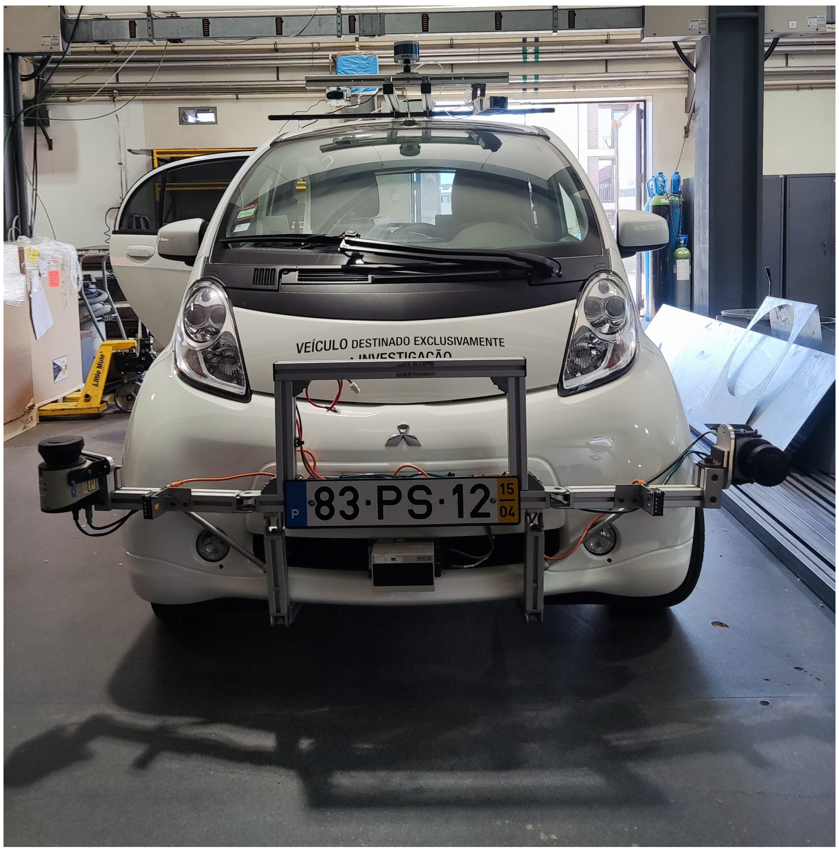


Figure 1.2: Image of ATLSCAR2 vehicle. [2]

1.2 Problem Description

Accuracy and speed are important characteristics for a successful process in a company. We can guarantee these characteristics in a process of outdoor volume calculation by exploring a georeferenced measurement solution that is easier to implement in the field. The approach involves the use of a GPS receiver travelling through the regions to be delimited, recording the paths, developing software to analyze and segment closed regions and evaluate their area. By having one or more sensors for measuring distances, such as a LIDAR, it is possible to collect the profile of the external dimensions of the material stored along these paths during the execution process, and additionally estimate the volume occupied inside the regions to be defined. These sensors, placed on board a vehicle, provide a quick survey and potentially with great precision.

1.3 Objectives

The main objective of this dissertation is to develop a system, on ATLASCAR2, that allows the representation and automatic processing of LIDAR and GNSS detection results. To achieve this purpose, the following intermediate objectives will have to be achieved:

- Development of a solution for data collection in a simulated environment;
- Development of a technical solution in ATLASCAR2 to collect the data necessary for the georeferenced assessment of surfaces and apparent volumes;
- Development of an application to transform the collected data into a description of georeferenced regions and volumes and display the results of surveys and detections;

1.4 Document Structure

This document is divided in 6 chapters. This first chapter presents the context of the problem and the objectives for this dissertation. Chapter 2 describes the state of the art for this type of work. The hardware and software used during this dissertation are presented in Chapter 3. Chapter 4 presents the solution developed, and the results obtained during the experiments are discussed in Chapter 5. Finally, Chapter 6 presents the conclusions and the work that can be developed in the future.

Intentionally blank page.

Chapter 2

State of the Art

This chapter explores solutions and algorithms for 3D point cloud processing and volume estimation. It also covers a description of the current solutions for performing the measurement of volumes for outdoor formations. As mentioned before, it is needed to find a solution that can georeference and calculate the volume of an object, so it is important to investigate some work done in 3D perception, volume calculation, filtering, and processing point clouds.

2.1 Existing Solutions

During this research, two main articles that use different techniques for volume calculation of stockpiles were found.

The article Monitoring and Computation of the Volumes of Stockpiles of Bulk Material by Means of UAV Photogrammetric Surveying [14] explores the use of Unmanned Aerial Vehicle (UAV)s for measuring stockpile volumes, figure 2.1, and compares this method with Terrestrial Laser Scanning (TLS), figure 2.2. UAVs, equipped with high-resolution cameras, capture images from multiple angles to create 3D models of stockpiles, enabling accurate volume calculations. The study highlights that UAV photogrammetric surveying typically achieves horizontal accuracy within 1-3 cm and vertical accuracy within 2-5 cm, with volume calculations often accurate to within 2-5% of the actual volume. In contrast, TLS, which uses a laser scanner to capture detailed 3D data from a fixed location, provides higher precision with distance measurements accurate to within millimeters. TLS is renowned for its detailed surface mapping and high accuracy but involves higher costs and more complex deployment. Setting up TLS can take several hours to a day, including equipment calibration and site preparation. UAV photogrammetric surveying generally requires a few hours for flight planning and data capture, with processing times for generating 3D models and calculating volumes ranging from a few hours to several days.

Also details the use of specific software for both methods, UAV data is processed using photogrammetric software such as Pix4D or Agisoft Metashape, which converts images into accurate 3D models and volume measurements. TLS data is processed with software like Leica Cyclone or Faro Scene, which helps in creating precise point clouds and detailed surface models. The article highlights the efficiency of UAVs in terms of cost, time, and flexibility, particularly in difficult-to-access areas, while TLS serves as a benchmark for high-precision requirements.

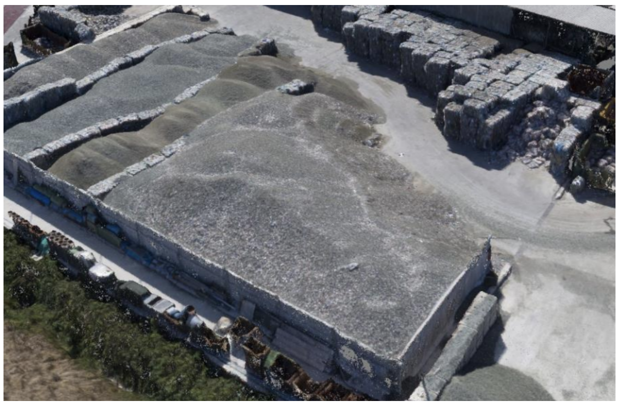


Figure 2.1: Dense cloud generated with Photoscan [14].

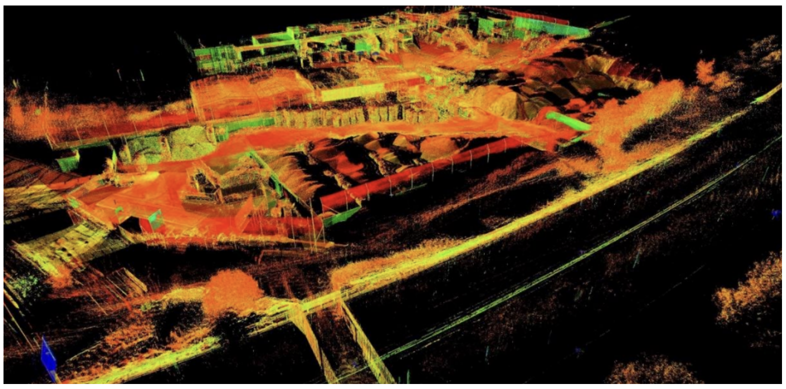


Figure 2.2: 3D point model generated with laser scan [14].

The article, Approximate Volume of Sand Materials Stockpile Based on Structure From Motion (SFM) [9]. focuses on utilizing image processing techniques to estimate the volume of sand stockpiles. Specifically, the study employs the SFM method, which involves capturing images of the stockpile from various angles to create a detailed 3D model. This 3D model is then analyzed to determine the stockpile's volume.

The volume calculation is carried out by determining the centroid and radius distance using the Euclidean distance algorithm, as illustrated in figure 2.3. The process involves capturing the overall three-dimensional structure of the object with cameras positioned around it, as shown in figure 2.4.

To evaluate the effectiveness of this method, the study measured 25 sand stockpiles. The average accuracy of the volume estimations achieved with this technique is approximately 74.21%, indicating a high level of precision in the volume measurements using the SFM approach.

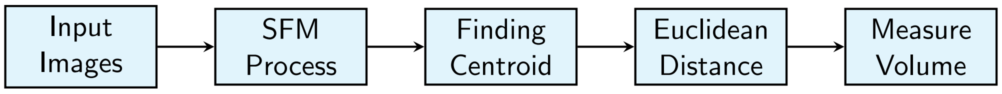


Figure 2.3: Step by step, measuring object volume [9].

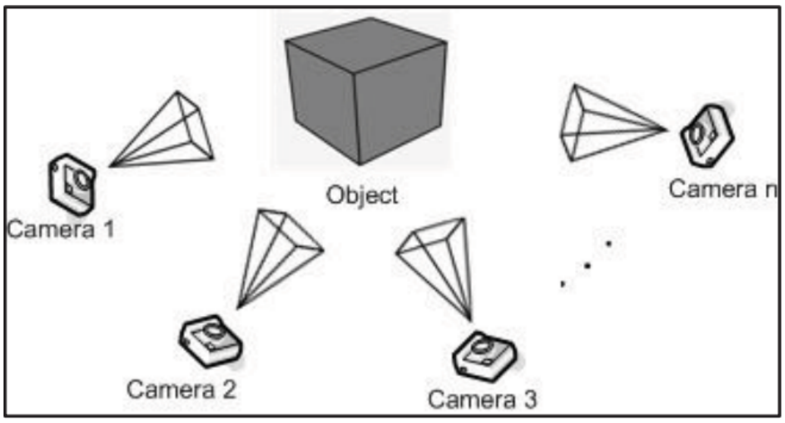


Figure 2.4: Structure from motion [9].

2.2 3D Perception

In recent years, 3D modeling from point-cloud data has been a popular investigation topic by many researchers. An increasing number of applications require the use of accurate 3D models. Examples of popular applications include virtual reality, autonomous mobile mapping, scanning of historical artifacts, gaming, 3D printing, and many others.

The primary data source for various mapping applications has been point clouds from LIDAR sensors, figure 2.5. Processing large-scale geospatial data, especially point clouds, can be very computationally and time-demanding because point clouds can contain millions of points. Therefore, it is crucial to implement fast and accurate processing algorithms. If applications handle unorganized point clouds, the complexity of the algorithms grows due to the scanning from multiple viewpoints and subsequent merging. The difficulty also increases with the amount of noise, outliers, and other inaccuracies.

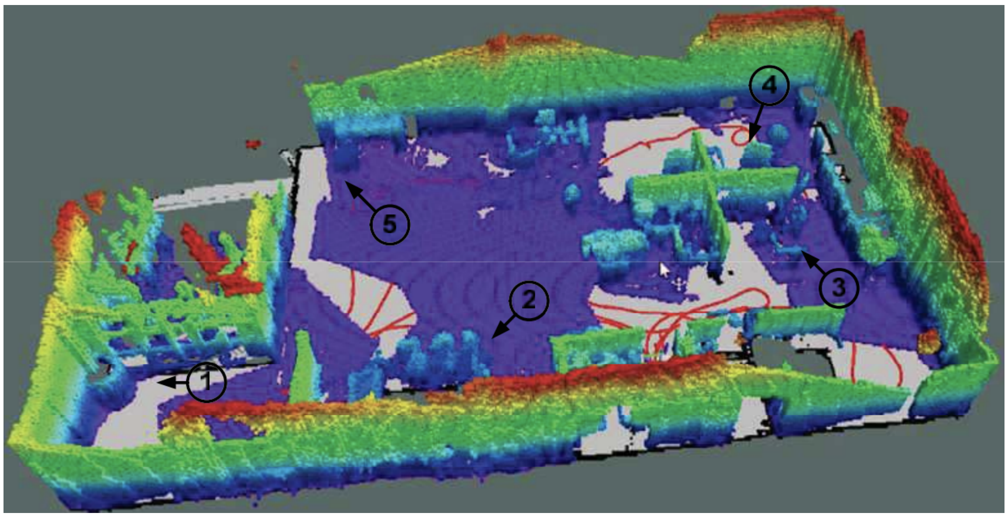


Figure 2.5: Example of a Point Cloud from LIDAR data [22].

Point Clouds

Point clouds are data sets defined by points in a given coordinate system. In a 3D Cartesian coordinate system, a point is identified by three coordinates that, taken together, correlate to a precise point in space relative to the point of origin. The point cloud is a vector-based structure, where each point has XYZ coordinates and other optional attributes that can represent time, intensity, color reflectivity, etc. It is an accurate digital record that covers the external surfaces of an object.

Point clouds can be obtained by accumulation of several laser scans. Robot Operating System (ROS) has a package, `laser_assembler` [5], represented in figure 2.6, which is used to collect point cloud data from laser scans. It essentially accumulates individual laser scans taken at different times and from different positions into a single point cloud representation. This package offers significant advantages for creating point clouds, including improved accuracy, enhanced environmental mapping, seamless ROS integration, flexibility, real-time operation, and resource efficiency. These benefits make it a valuable tool for robotics applications that require detailed and accurate 3D representations of the environment.

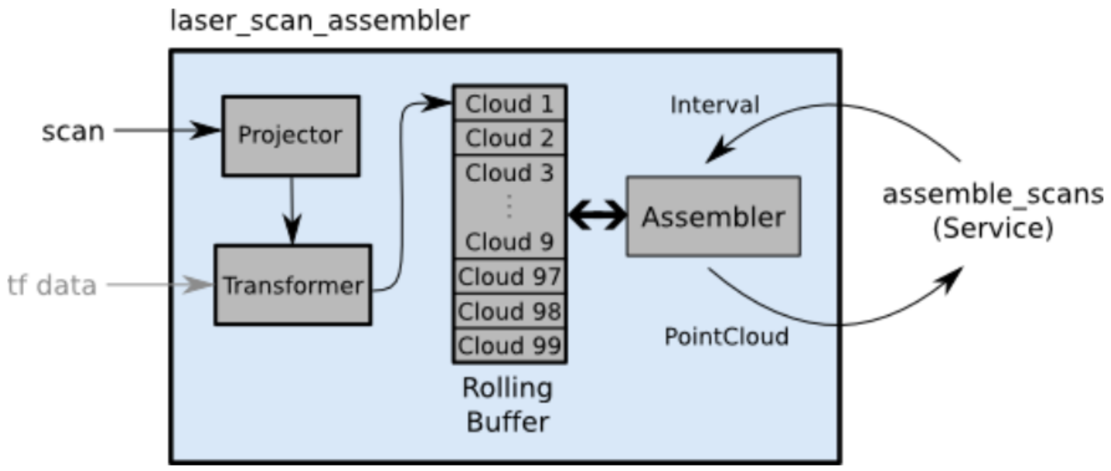


Figure 2.6: Transforming laser scans into point clouds [5].

2.3 Filtering Methods

Filtering a point cloud is essential for ensuring data quality, reducing computational load, improving accuracy, and preparing the data for further processing. Each filtering method addresses specific issues such as noise, outliers, and data size, contributing to more effective and efficient point cloud processing.

2.3.1 Statistical Outlier Removal

Statistical Outlier Removal [23], figure 2.7, is a technique designed to filter out outliers from a point cloud based on statistical analysis of the distances between neighboring points. The method calculates the mean distance of each point to its k nearest neighbors.

If a point's distance to these neighbors exceeds a specified threshold, it is considered an outlier and is removed from the point cloud. This approach is particularly effective for eliminating sparse outliers that deviate significantly from the local point density. The simplicity of the method makes it a popular choice for cleaning up noisy point clouds.

However, Statistical Outlier Removal (SOR) has its limitations. In particular, it can be sensitive to the distribution of points within the dataset. If the point cloud is not uniformly distributed or contains regions with varying densities, SOR might incorrectly classify valid points as outliers. This is because the threshold used to determine outliers is applied uniformly, without accounting for variations in local point density. Consequently, while SOR is effective for removing isolated outliers, careful consideration must be given to its parameters to avoid inadvertently removing useful data.

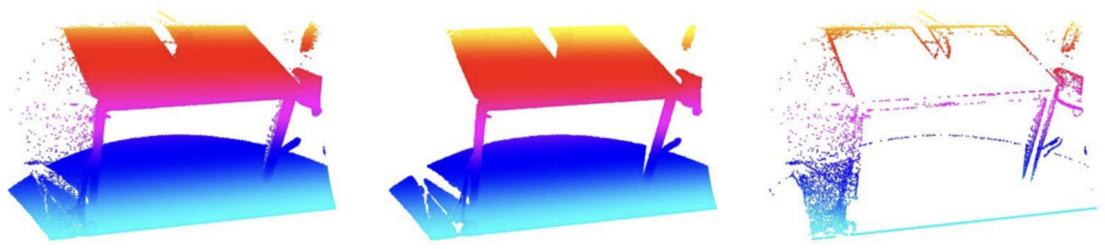


Figure 2.7: Application of Statistical Outlier Removal [23].

2.3.2 Radius Outlier Removal

Radius Outlier Removal [17], figure 2.8, is a technique used to clean up point clouds by identifying and removing outliers based on the density of their local neighborhoods. For each point in the point cloud, Radius Outlier Removal (ROR) counts the number of neighboring points within a specified spherical radius. Points with fewer neighbors than a defined threshold are considered outliers and removed. This method is particularly effective in scenarios where the point cloud is uniformly dense, as it can efficiently eliminate isolated points that deviate from the general density pattern. However, the choice of radius is critical to the method's success. A radius that is too small may fail to include enough neighboring points, potentially removing valid points, while a radius that is too large may capture too many points, leading to incorrect classification of non-outliers. Additionally, ROR may not perform well in cases where the point cloud exhibits varying densities across different regions, as it does not account for these variations when determining outliers.

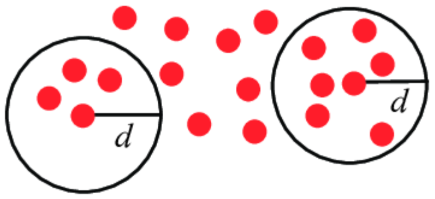


Figure 2.8: Principle of Radius Outlier Removal [17].

2.3.3 Voxel Grid Filtering

Voxel Grid Filtering [19], represented in figure 2.9, is a down-sampling technique that reduces the number of points in a point cloud by dividing the space into a 3D grid of cubic cells, known as voxels. All points within a voxel are replaced by a single representative point, such as the centroid of the points within that voxel. This method significantly decreases the number of points in the point cloud, making it easier to handle and process large datasets. The major advantage of Voxel Grid Filtering (VGF) is its ability to preserve the overall structure and shape of the point cloud while reducing the point count. However, the effectiveness of VGF is highly dependent on the chosen voxel size. A larger voxel size may lead to a loss of important details, while a smaller voxel size might not reduce the number of points sufficiently. VGF also assumes a relatively uniform distribution of points within each voxel, which may not always be appropriate for point clouds with varying densities.

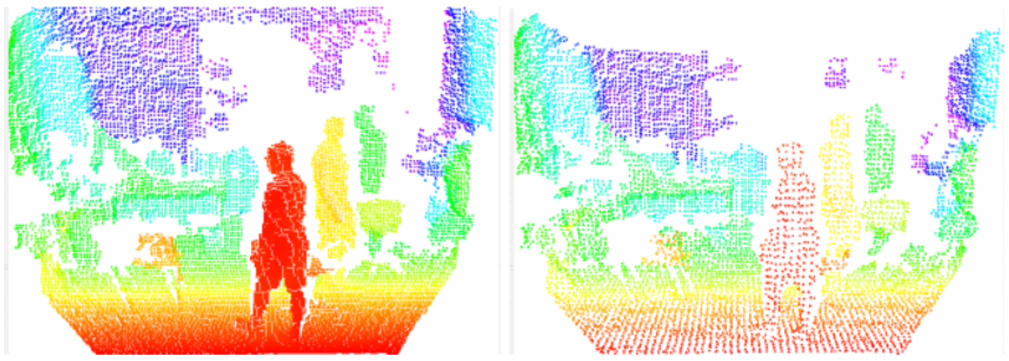


Figure 2.9: The effect of the voxel grid filter on Kinect 3D data [19].

2.3.4 Passthrough Filtering

Passthrough Filtering [18], represented in figure 2.10, is a simple and efficient method for removing points that fall outside a specified range along one or more dimensions (x, y, or z). By defining a range along a given dimension, Passthrough Filtering (PTF) filters out points that are outside this range, effectively focusing on specific regions of interest within the point cloud.

This method is particularly useful for isolating and examining certain areas of the dataset. However, PTF has its limitations; it can only remove points based on their position along a dimension and does not address internal outliers or noise within the defined range. As such, while PTF is effective for defining bounding boxes or focusing on particular spatial regions, it may not handle internal noise or variations within the region of interest.

2.4 Volume Calculation Methods

Calculating the volume enclosed by a point cloud is a fundamental task with numerous applications, including object recognition and terrain modeling. To estimate volume

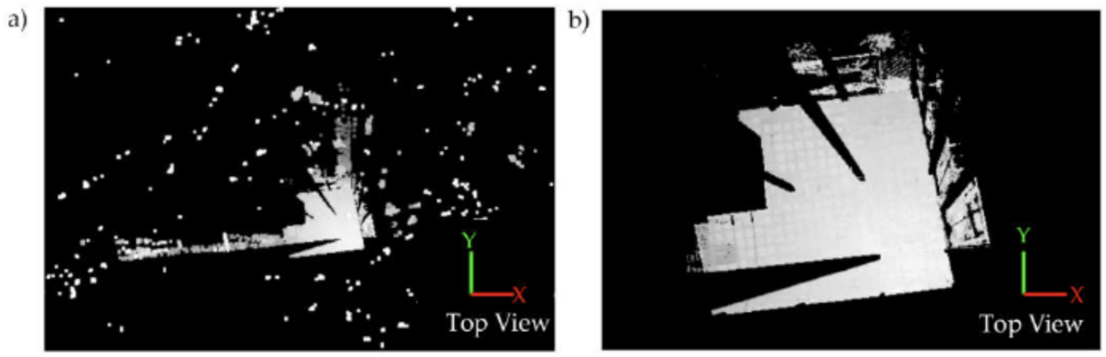


Figure 2.10: Effect of applying the Passthrough Filter [18].

from point clouds, several methods are available, each with its own advantages and disadvantages. In the following subsections, it is presented these methods in detail.

2.4.1 Slice Method

The Slice Method [15] is a volumetric calculation technique that divides a 3D object into a series of parallel slices, calculates the area of each slice, and then sums these areas to estimate the total volume. This method is particularly effective for objects that have a regular shape or can be approximated by a series of 2D cross-sections. Each slice is analyzed to determine its area, and by multiplying this area by the distance between slices, the volume of each section can be calculated and summed. Figure 2.11 illustrates the concept of the Slice Method applied to a 3D point cloud.

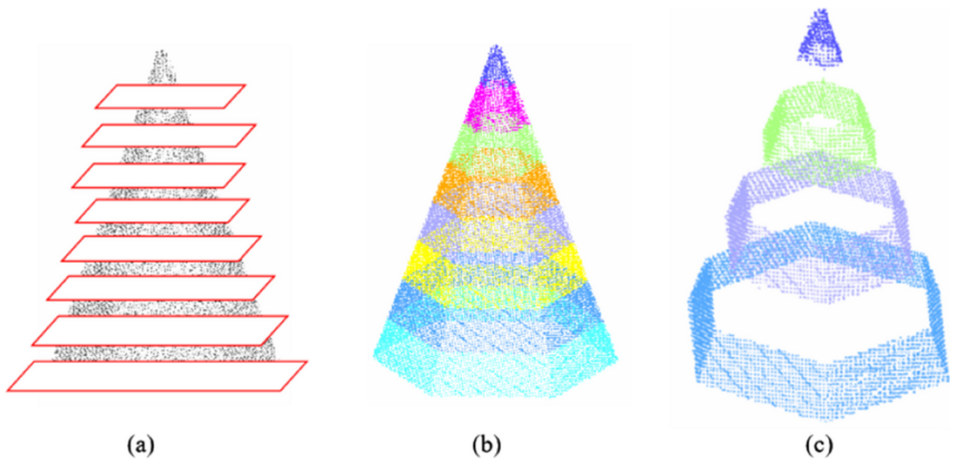


Figure 2.11: Illustration of the Slice Method applied to a 3D point cloud [15].

2.4.2 Delaunay Triangulation

Delaunay triangulation is a geometric approach that constructs a network of triangles from the given points in the point cloud, such that no point is contained within the circumcircle of any triangle. Once the triangulation is formed, the volume can be calculated in several different ways. For example, The triangle can be project onto the xy plane to create a triangular prism, figure 2.12, also it can be used tetrahedrons formed by each triangle and the centroid of the point cloud.

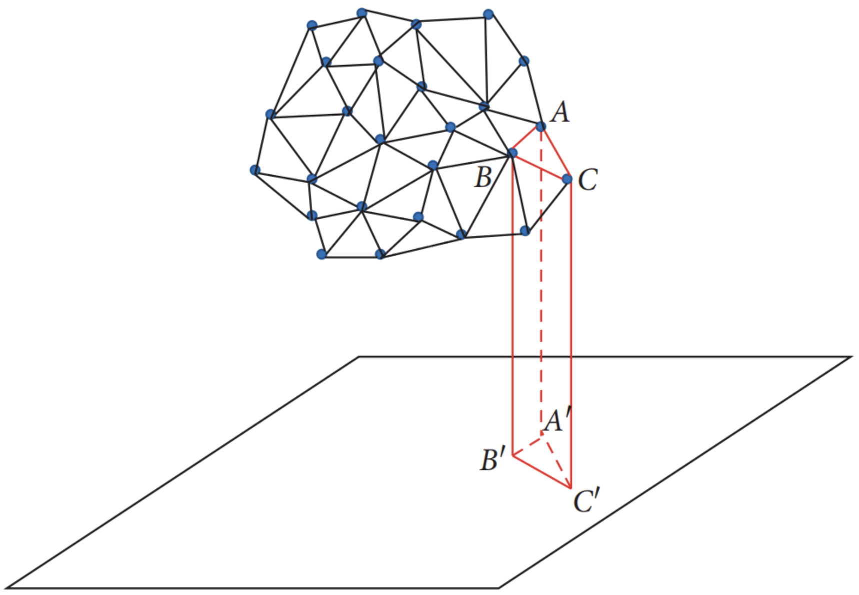


Figure 2.12: Triangle projected into XY plane [16].

Delaunay triangulation offers a more precise volume estimation compared to voxelization, especially for irregular point distributions and complex shapes. However, its computational complexity increases with the number of points, making it less suitable for large-scale point clouds [16].

2.4.3 Voxel-based volume estimation

In this approach [13], the 3D space is divided into a grid of small cubic elements known as voxels. Each voxel represents a discrete volume element, similar to how pixels represent discrete areas in 2D images. This method is efficient and straightforward, particularly for uniformly distributed point clouds. However, the accuracy of the estimated volume can be influenced by the size of the voxels. Smaller voxels can provide a more precise estimate, but at the cost of increased computational resources and processing time. Conversely, larger voxels might simplify computations but can lead to inaccuracies, particularly for irregularly shaped point clouds. Figure 2.13 shows an example of voxelization.

2.5 Conclusion

This chapter provided an overview of the current state-of-the-art solutions and methods for 3D point cloud processing and volume estimation, particularly in the context

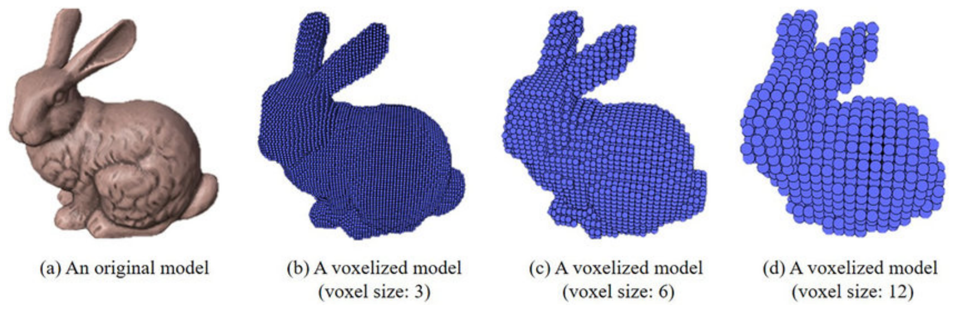

(a) An original model
(b) A voxelized model (voxel size: 3)
(c) A voxelized model (voxel size: 6)
(d) A voxelized model (voxel size: 12)

Figure 2.13: Voxelization of a point cloud [13].

of outdoor formations. The review covered existing solutions that employ UAVs and photogrammetry, as well as various filtering and volume calculation methods.

Most related projects rely on UAVs for data acquisition, leveraging their ability to cover large areas and access hard-to-reach locations. However, the inherent limitations of UAVs, such as sensitivity to weather and limited battery life, suggest that ground-based vehicles like ATLASCAR2 could offer a more stable and reliable platform for such tasks.

Furthermore, while photogrammetry offers high accuracy, the complexity and expense of this approach make it less ideal for certain applications. A more streamlined solution might involve using LIDAR with odometry or GPS+Inertial Measurement Unit (IMU) to simplify data collection and processing.

In the study of point cloud processing, several effective filtering methods were identified, including Passthrough Filtering and Voxel Grid Filtering. These techniques are particularly useful for reducing noise and computational load. The Delaunay triangulation method for volume calculation was also noted for its favorable balance between accuracy and computational efficiency.

Given that stockpiles have irregular top surfaces and flat bottoms, Delaunay triangulation with projection is a suitable method for modeling them. However, with a large number of points, this process can become slow. To speed it up, it's crucial to use effective filtering techniques.

Based on the findings in subsection 2.3, Passthrough Filtering should be used to crop the point cloud, isolating the relevant region and removing unwanted points. Voxel Grid Filtering can then be applied to reduce the number of points within that region. These methods were selected for their simplicity and minimal computational demands.

In conclusion, the insights gained from this chapter will guide the development of a robust and efficient system for georeferencing and volume estimation using 3D point cloud data.

Intentionally blank page.

Chapter 3

Experimental Infrastructure

This chapter describes software and hardware tools used in this dissertation. The hardware are the sensors used to gather all necessary data. The software includes all frameworks used to process and visualize data.

3.1 Hardware

The most relevant sensors for this project are the LIDAR 2D, GPS + IMU, and a odometry unit. This section provides specific characteristics of these sensors.

3.1.1 LIDAR SICK LMS151

A LIDAR sensor is a device that measures reflected light properties in order to obtain distance and other information about a given object.

In this case, SICK LMS151 (Figure 3.1) is a 2D laser range finder designed for both indoor and outdoor use, providing reliable data for navigation, detection, and measurement. In Table 3.1 are described as the main features of this sensor [6].

Figure 3.1: LIDAR sensor LMS151 [6].

The current sensor setup in ATLASCAR 2 (Figure 3.7) does not allow obtaining a 3D point cloud as required for this project. Also, it is necessary to make changes to this setup. These changes will be presented in Chapter 4.

Feature	Value
Light Source	Infrared (905 nm)
Laser class	1 (IEC 60825-1:2014, EN 60825-1:2014)
Horizontal aperture angle	270°
Scanning frequency	25 Hz ... 50Hz
Angular resolution	0.25°, 0.5°
Working range	0.5 m ... 50 m
Amount of evaluated echoes	2

Table 3.1: Main features of SICK LMS151 LIDAR sensor [6].

3.1.2 Novatel SPAN-IGM-A1 and Novatel GPS-702-GG

The Novatel SPAN-IGM-A1 [8] (Figure 3.2) is an advanced GNSS (Global Navigation Satellite System) receiver combined with an inertial measurement unit IMU. It leverages Synchronous Position, Attitude, and Navigation Synchronous Position, Attitude and Navigation (SPAN) technology, which is a fusion of GNSS and inertial navigation systems INS. This combination offers a highly precise and robust positioning solution, even in challenging environments where GNSS signals might be weak or obstructed, such as in urban canyons or under heavy foliage. Table 3.2 shows the equipment specifications.

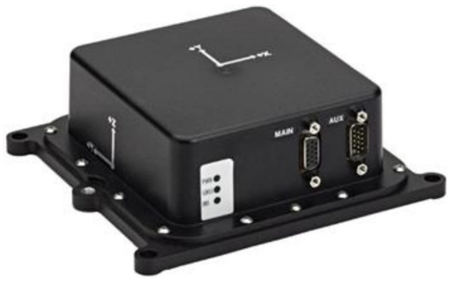


Figure 3.2: Novatel SPAN-IGM-A1 [8].

Feature	Value
Input voltage	10-30 VDC
Time Accuracy	20 ns RMS
Max Velocity	515 m/s
Single point L1/L2 accuracy	1.2 m
IMU measurement data rate	200 Hz
INSS solution data rate	Up to 200 Hz
Dimensions	152 × 142 × 51 mm
Weight	515 g

Table 3.2: Novatel SPAN-IGM-A1 specifications [8].

The Novatel GPS-702-GG [7] (Figure 3.3) is a high-performance GNSS antenna that is designed to work in tandem with the SPAN-IGM-A1. This antenna is capable of receiving signals from multiple GNSS constellations, including GPS, GLONASS, and others, providing enhanced satellite availability and positioning accuracy. Table 3.3 shows the equipment specifications.

Figure 3.3: Novatel GPS-702-GG [7].

Feature	Value
Input voltage	4.5-18.0 VDC
Current (typical)	35 mA
3 dB pass band (typical)	L1: 1588.5 ±23.0 MHz; L2: 1236.0 ±18.3 MHz
Noise figure (typical)	2.5 dB
L1-L2 differential propagation delay	5 ns
Diameter	185 mm
Weight	500 g

Table 3.3: Novatel GPS-702-GG specifications [7].

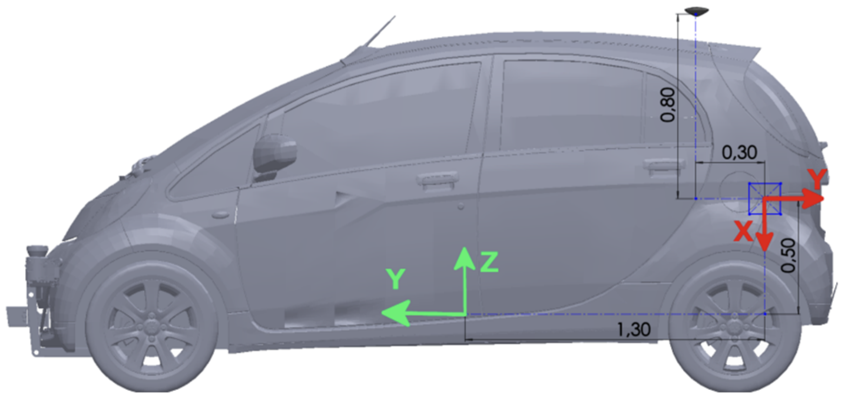
When combined, the Novatel SPAN-IGM-A1 and GPS-702-GG provide a powerful and reliable positioning solution that is particularly well-suited for applications requiring high precision and reliability. For instance, in the context of autonomous vehicles, accurate positioning is critical for navigation and safety. The SPAN-IGM-A1 ensures that the vehicle's position is continuously tracked, even in environments where GNSS signals might be interrupted.

In ATLASCAR2, the current setup is configured to use a GPS signal with a frequency of 20Hz, that goes to the SPAN receiver to be processed along with IMU data, gathered at a frequency of 200Hz. These sensors are connected to the car through RS232. The setup was configured by P. Nova and is explained in more detail in [20].

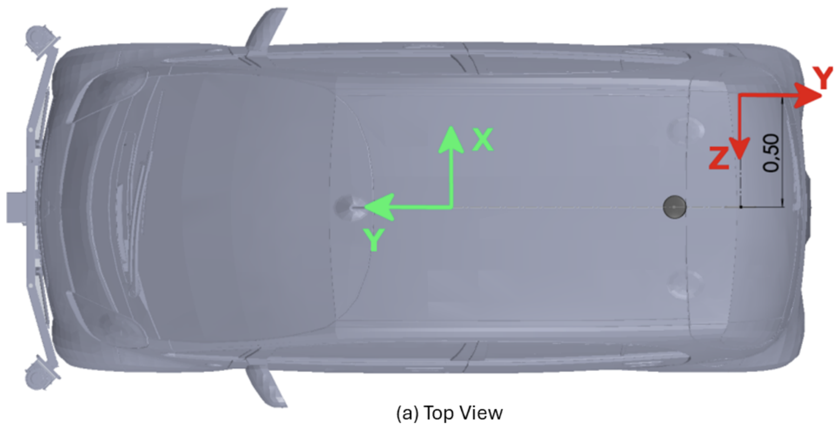
Initial configuration and calibration of GNSS system needs to be performed to ensure synchronization and accuracy before launching the `novatel_bringup`, described in section 4.3.2. Figure 3.4 shows the placement of the device in the car.

To configure the system, a few commands need to be sent to the sensor, Novatel SPAN-IGM-A1. This process can be performed graphically through the NovAtel Connect application or directly from a serial terminal. In this thesis, it was used the `Cutecom` terminal to send the commands.

Figure 3.5 shows which commands were sent to the sensor. The configuration commands are used to set up a NovAtel GNSS+INS system, establishing communication



(a) Side View



(a) Top View

Figure 3.4: GNSS+INS Installation - vehicle's axis system in green and the sensor's axis system in red.

parameters, orientation, and offsets essential for accurate navigation and positioning. The `SERIALCONFIG` commands configure two serial ports (COM1 and COM2) with a baud rate of 230400, no parity, 8 data bits, 1 stop bit, no flow control, and enable these ports. The `SETIMUORIENTATION` command sets the IMU orientation mode to a predefined setting. The `VEHICLEBODYROTATION` and `APPLYVEHICLEBODYROTATION` commands adjust and apply a 180-degree yaw rotation and a slight roll adjustment to the vehicle body relative to a reference frame, so we can obtain the configuration represented in figure 3.4. The `SETIMUTOANTOFFSET` command specifies the positional offset between the IMU and the GNSS antenna, along with its associated uncertainties, while `SETINSOFFSET` defines the INS offset relative to the vehicle body. The `ALIGNMENTMODE` command sets the system to operate in an unaided alignment mode, and finally, `SAVECONFIG` saves these settings to non-volatile memory, ensuring they persist after a restart.

After the completed configuration, it is recommended to perform an alignment without turning off the sensor. This process should be conducted in an open and flat area, allowing the vehicle to travel in a straight line from 20 to 25 km/h around 20 seconds.

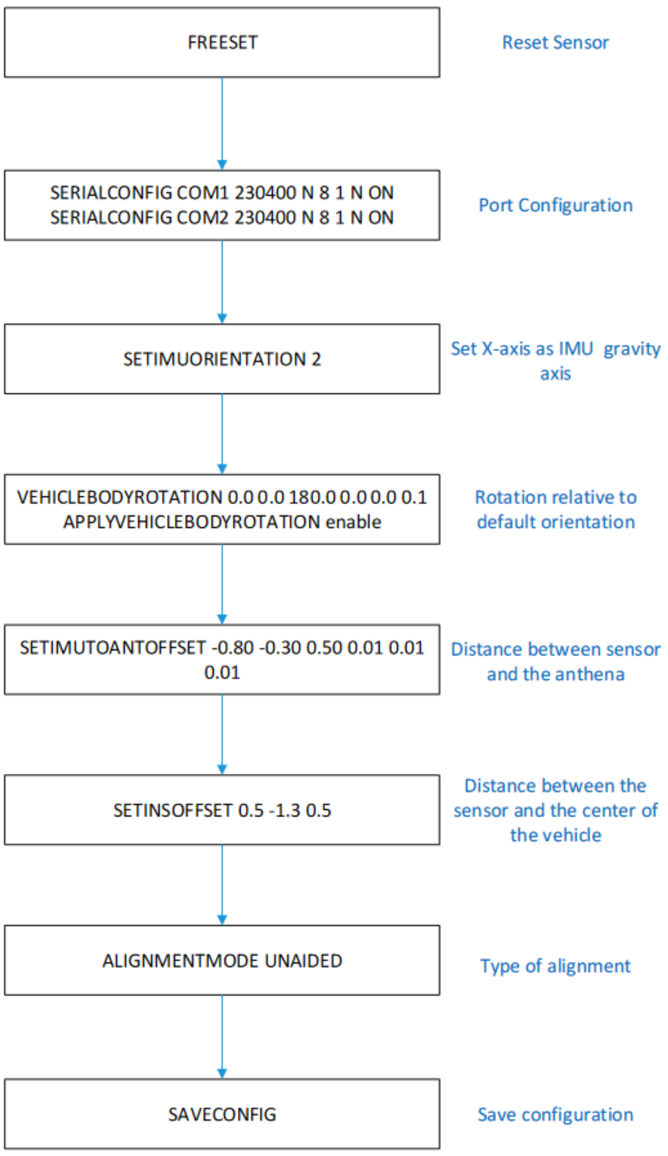

FREESET
Reset Sensor
SERIALCONFIG COM1 230400 N 8 1 N ON
SERIALCONFIG COM2 230400 N 8 1 N ON
Port Configuration
SETIMUORIENTATION 2
Set X-axis as IMU gravity axis
VEHICLEBODYROTATION 0.0 0.0 180.0 0.0 0.0 0.1
APPLYVEHICLEBODYROTATION enable
Rotation relative to default orientation
SETIMUTOANTOFFSET -0.80 -0.30 0.50 0.01 0.01 0.01
Distance between sensor and the anthena
SETINSOFFSET 0.5 -1.3 0.5
Distance between the sensor and the center of the vehicle
ALIGNMENTMODE UNAIDED
Type of alignment
SAVECONFIG
Save configuration

Figure 3.5: Configuration commands to set up a NovAtel GNSS+INS system.

3.1.3 Odometry

The odometry solution developed for the ATLASCAR2 involves several stages, from hardware installation to software computation. This solution utilizes the Controller Area Network (CAN) to acquire necessary data from the vehicle. It uses an encoder RI32-0/1000ER.14KB [3] (Figure 3.6), is an incremental rotary encoder, commonly used in industrial automation and control systems to measure rotational position and speed. Table 3.4 shows the encoder specifications.

In ATLASCAR2, this encoder is used to obtain odometry information. Odometry is

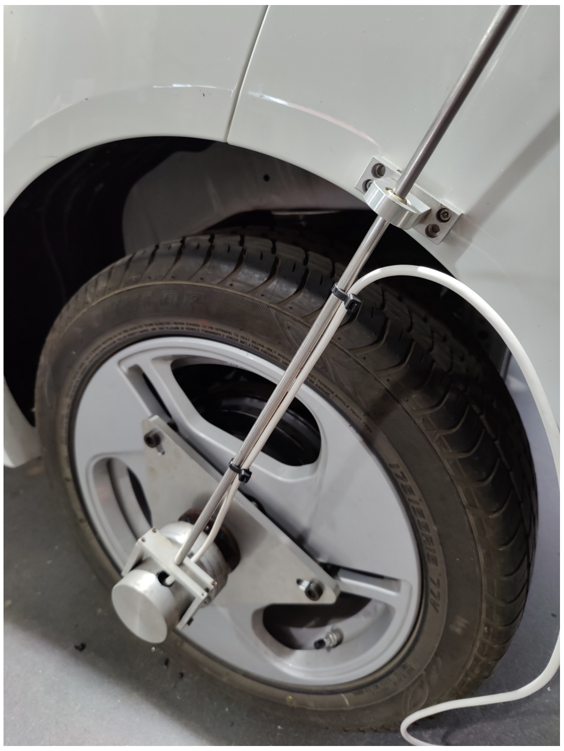


Figure 3.6: Encoder location in ATLASCAR2 [21].

Feature	Value
Type	Incremental
IP rating	IP40, IP50
Diameter	30 mm
Shaft Length	10 mm
Shaft Diameter	5 mm
Max rotational speed	6000 rpm
No. of Channels	3

Table 3.4: RI32-0/1000ER.14KB specifications [3].

useful to accurately track the position and speed of the vehicle. The setup was configured by Sara Pombinho and is explained in [21].

3.1.4 Sensors Placement

As mentioned in Chapter 1, the ATLASCAR2 vehicle is equipped with multiple devices. Besides the sensors mentioned before, it also has four optoelectronics sensors Sick DT20 Hi that assist in the inclinometry module. There is also a PointGrey cameras to perform road perception with artificial vision, a velodyne LIDAR and a SICK 3D LIDAR.

The Sick DT20 Hi are currently configured with 2.5 ms response time (400Hz) and a measuring range between 200 mm and 700 mm and are connected to an Arduino that internally calculates the inclinometry of the car.

The placement of each device in ATLASCAR2 is represented in Figure 3.7.

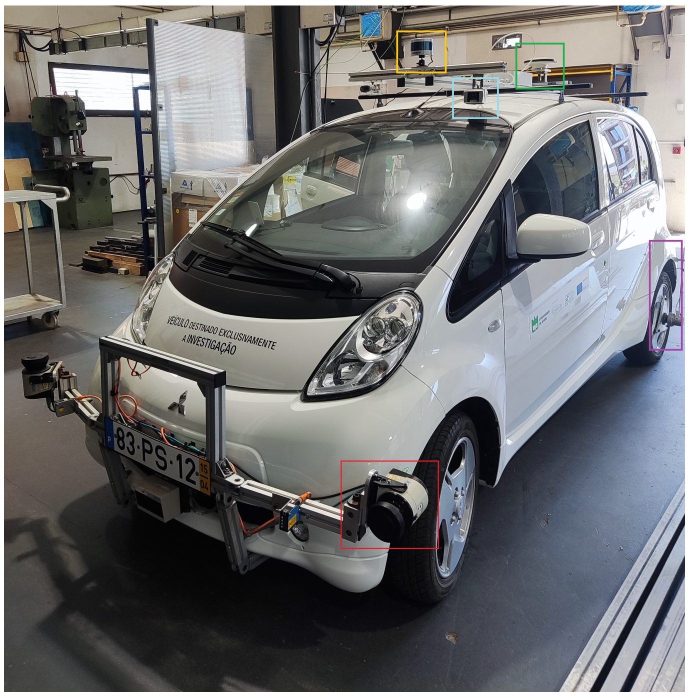


Figure 3.7: Sensors location in ATLASCAR2 [21].

3.2 Software

This section provides information about ROS and its packages, as well as the python libraries used.

3.2.1 ROS - Robot Operating System

ROS, created in 2007 at Stanford University, is an environment designed to help build large scale robotics projects, by focusing on modularity of different systems and having tools to interconnect them [11].

These modules are called nodes in ROS nomenclature, and they communicate with each other using predefined data structures called messages, published and received through the topics that connect the nodes of the program. Nodes are also grouped in packages, that can contain several nodes, or just one, and are defined by a single `CMakeList.txt` file, where the specifications necessary to compile the package are defined.

An example of the architecture for program developed in ROS is represented in Figure 3.8.

The ATLASCAR2 is currently developed using ROS platform, so in order for this work to be integrated in the ATLAS project, all the code that will be developed in the context of this dissertation will also follow the ROS framework.

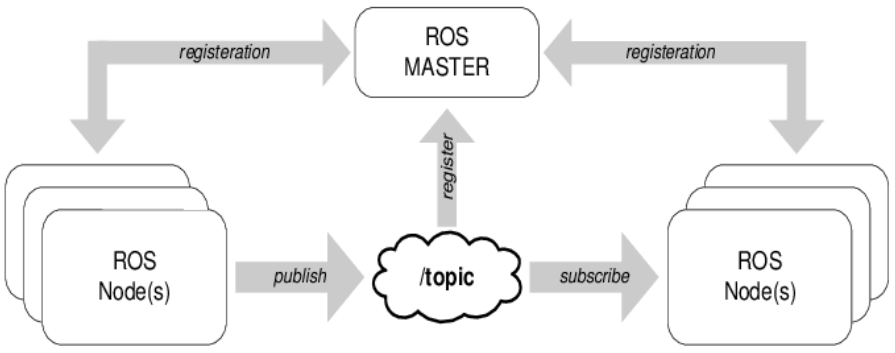
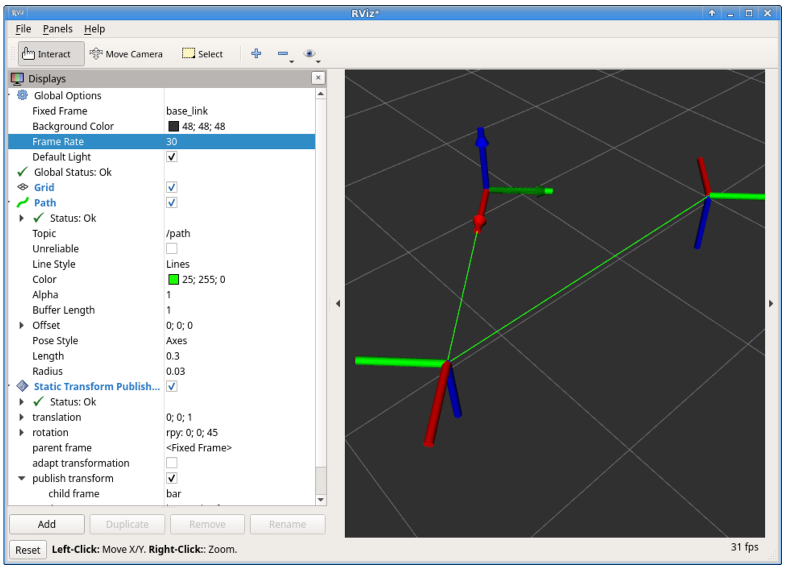


Figure 3.8: ROS Diagram [22].

3.2.2 Rviz

Rviz (Figure 3.9) is a 3D visualization tool for ROS. It provides a Graphical User Interface (GUI) for the visualization of robotic platforms and of a wide range of sensor's data types. Rviz also includes a library that allows most of its functionalities to be used by other applications [12].

In this work, Rviz will be used to display the processed data from the LIDARs.



Global Options	
Fixed Frame	base_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
Path	✓
Status: Ok	
Topic	/path
Unreliable	
Line Style	Lines
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	Axes
Length	0.3
Radius	0.03
Static Transform Publish...	✓
Status: Ok	
translation	0; 0; 1
rotation	rpy: 0; 0; 45
parent frame	<Fixed Frame>
adapt transformation	
publish transform	✓
child frame	bar

Left-Click: Move X/Y. **Right-Click:**: Zoom.

Figure 3.9: Rviz GUI [16].

3.2.3 Gazebo

Gazebo (Figure 3.10) is an open source 3D robotics simulator. It uses a physical engine for illumination, gravity, inertia, etc. It is possible to evaluate and test robots in different scenarios and model sensors, such as laser range finders, cameras, Kinect style sensors, etc. [4].

Gazebo will be used to simulate the data collection.

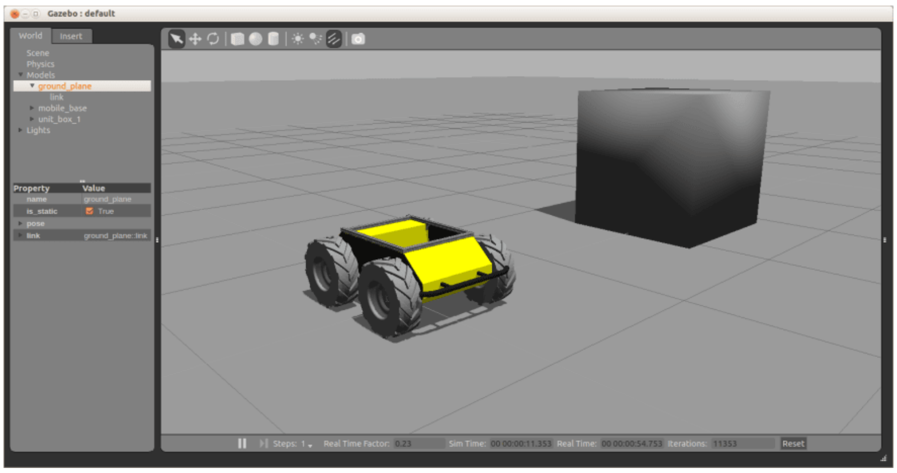


Figure 3.10: Gazebo GUI [4].

3.2.4 Rqt

The Rqt is a software framework of ROS that implements various GUI tools. The tools used in this project were `rqt_graph` to visualize topics and nodes, and the `robot_steering`, which publishes a `Twist` message in a topic.

3.2.5 ROS Packages Used in This Project

The ROS packages listed below are detailed in [11].

LMS1xx

The LMS1xx package supports every Sick LMS1xx laser scanner. This package includes the LIDARs Unified Robot Description Format (URDF).

novatel_gps_driver

The novatel_gps_driver package is designed for Novatel GPSs and includes features such as choosing the device type (Ethernet or USB) and the messages the user wants to publish. Using USB, the user can choose the baudrate, sample rate, and device port of the sensor, without needing to use the ”Novatel Connect” application.

laser_assembler

The laser_assembler package listens to streams of scans and uses them to assemble them into Cartesian coordinates point cloud. This package is used in this project to accumulate laser scan messages, GPS, and odometry.

laser_to_pcl

The laser_to_pcl package converts the point cloud assembled into Point Cloud Library (PCL) format.

The ROS packages outlined in this section play crucial roles in enabling the experimental infrastructure for the ATLASCAR2 project. The `LMS1xx` package provides support for the SICK `LMS1xx` laser scanners, facilitating the integration and use of LIDAR data within the ROS ecosystem.

swri_transform_util

This package offers essential utilities for handling transformations between different coordinate frames, including translation, rotation, and conversion between various representations such as matrices, quaternions, and Euler angles. This package plays a crucial role in tasks such as sensor fusion, motion planning, and robot navigation by providing robust tools for computing and applying transforms.

static_transform_publisher

It is designed to publish fixed, unchanging transformations between coordinate frames. It allows users to define and broadcast static transforms, which represent a constant spatial relationship between two frames, such as the position and orientation of sensors relative to a robot's base. This package is typically configured via command-line parameters to specify translation and rotation values.

3.2.6 Python Libraries

To develop the Point Cloud Measures application discussed in section 4.5, the following Python libraries were utilized. Detailed information about these libraries can be found in [10].

PyQt5

PyQt5 is a set of Python bindings for the Qt application framework developed by the Qt Company. It allows Python programmers to create cross-platform applications with a native look and feel. In this project, PyQt5 is used to design the GUI components of the point cloud visualization and analysis tool.

matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface for generating plots and graphs, including support for 2D and 3D plotting. Here, matplotlib is employed to create 3D scatter plots for visualizing point cloud data.

numpy

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. In this project, numpy is used for handling and manipulating point cloud data represented as arrays.

Open3D

Open3D is a modern library for 3D data processing. It offers functions for 3D visualization, including point cloud visualization, geometry processing, surface reconstruction, and registration. Open3D is utilized in this project to read point cloud files and perform operations such as voxel downsampling and filtering.

scipy

SciPy is a library used for scientific and technical computing in Python. It provides modules for optimization, linear algebra, integration, interpolation, signal and image processing, and other tasks. In this project, scipy's Delaunay module is used to compute the Delaunay triangulation of the point cloud, which is essential for volume and area calculations.

mpl_toolkits

mpl_toolkits is Matplotlib's toolkit for 3D plotting. It extends the capabilities of matplotlib to support three-dimensional plotting, including functions and classes for creating 3D axes, plots, and visualizations. In this project, `mpl_toolkits` is utilized to create 3D scatter plots and visualize the point cloud data.

3.3 Conclusion

This chapter outlines the experimental infrastructure established for this dissertation, detailing the essential hardware and software components for data acquisition, processing, and visualization.

The hardware configuration includes the SICK LMS151 2D LIDAR, the Novatel SPAN-IGM-A1 with GPS-702-GG, and an odometry system. Each component is crucial for ensuring accurate data collection and positioning, which are vital for the effective operation of autonomous vehicles. The limitations of the current sensor setup and the necessary modifications are also examined.

On the software side, ROS is employed for system integration, Rviz for data visualization, Gazebo for simulation, and various Python libraries for data processing. These tools collectively enable effective data handling and visualization, supporting the research objectives of the dissertation.

The integration of these hardware and software tools establishes a robust experimental infrastructure, providing a solid foundation for the development and evaluation of solutions presented in the subsequent chapters.

Intentionally blank page.

Chapter 4

Proposed Solution

As stated earlier, the main objective of this dissertation is to develop a system, installed on ATLASCAR2, that allows the representation and automatic processing of LIDAR and GNSS detection results. To achieve this goal, the proposed solution involves the use of a vertical orientation LIDAR sensor to capture a detailed 3D point cloud of the surrounding environment. By moving the vehicle around the object of interest, the LIDAR continuously scans and collects data from different angles, which are then integrated to form a comprehensive three-dimensional model. The vehicle's precise localization is achieved through a combination of GPS and IMU data, or alternatively, through odometry, which tracks the movement and position based on previous positions and motion measurements. This integration of LIDAR scanning and robust localization techniques ensures accurate mapping and modeling of the environment. Figure 4.1 illustrates this proposed solution.

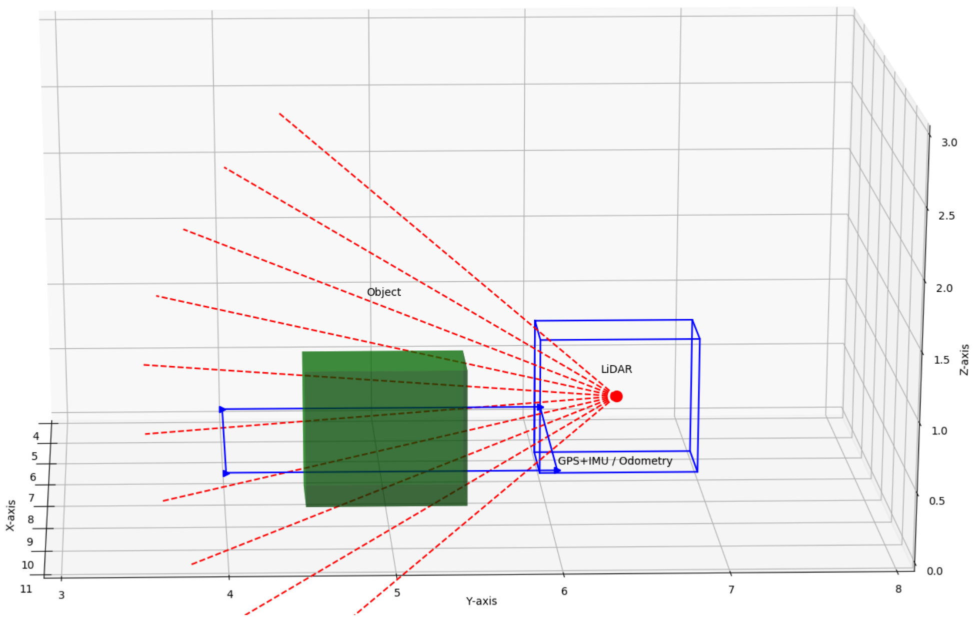


Figure 4.1: Diagram of the proposed solution.

Also, an interactive application was created that allows the user to visualize, filter and calculate the volume and area of a chosen point cloud.

4.1 LIDAR Installation

The current 2D LIDAR sensor setup in ATLASCAR2 does not generate a 3D point cloud in the required format for the primary objective of this work. To address this limitation, modifications to the LIDAR sensor's orientation and mounting are necessary.

The optimal solution involves rotating the LIDAR sensor by 90 degrees, enabling it to capture laser scan data from different segments of an object's surface. As ATLASCAR2 moves, these laser scans can be accumulated to form a comprehensive 3D point cloud.

To implement this solution, a new support structure was designed, as shown in Figure 4.2, allowing the reorientation of the LIDAR sensor.

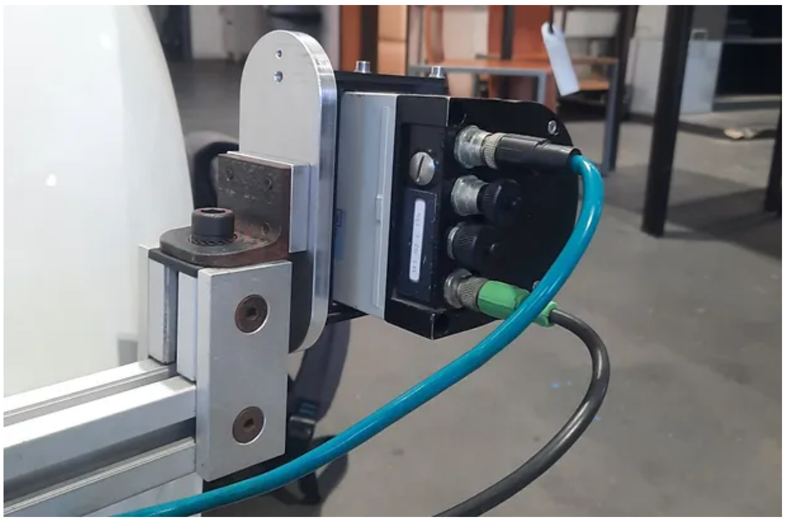


Figure 4.2: LIDAR with new supports applied.

Additionally, modifications were required in the ATLASCAR2 URDF file to accurately reflect the sensor's new orientation on the physical vehicle. These adjustments ensure that the simulation mirrors the real-world setup, thereby enhancing the accuracy of sensor data used during testing and development. By aligning the virtual sensor's orientation with its physical counterpart, the simulated laser scans will more accurately represent the actual environment. This alignment is crucial for improving the reliability and validity of tasks such as mapping, obstacle detection, and navigation. The necessary changes in the ATLASCAR2 URDF file are shown in Listing 4.1, where the sensor's orientation parameters are updated to reflect the 90-degree rotation.

```
...
<!-- left lms151 laser -->
<xacro:sick_lms1xx_joints frame="left_laser" topic="/left_laser/
    scan" parent="base_link"
     x="${length/2␣+␣length_offset␣-␣0.08}" y="0.76" z="${0.27+
         height_offset}"
     rol="${1.5708}" pit="0" yaw="${1.5708}"/>
...
```

Listing 4.1: Modifications in ATLASCAR2 URDF file to reflect the sensor relocation.

4.2 Architecture

The architecture of this solution is depicted in Figure 4.3. It is built upon the new ATLASCAR2 architecture developed by Sara Pombinho in [21], incorporating specific modifications to the GNSS and 2D LIDAR components. This architecture is organized into three main categories: sensor initialization, point cloud creation, and information management and processing for result generation. The subsequent sections will provide a detailed explanation of each component.

Although the primary focus of this thesis is on utilizing GNSS for displacement measurement, the architecture also accommodates the use of odometry for displacement purposes. This flexibility allows for the integration of different measurement techniques depending on the application requirements and available sensors.

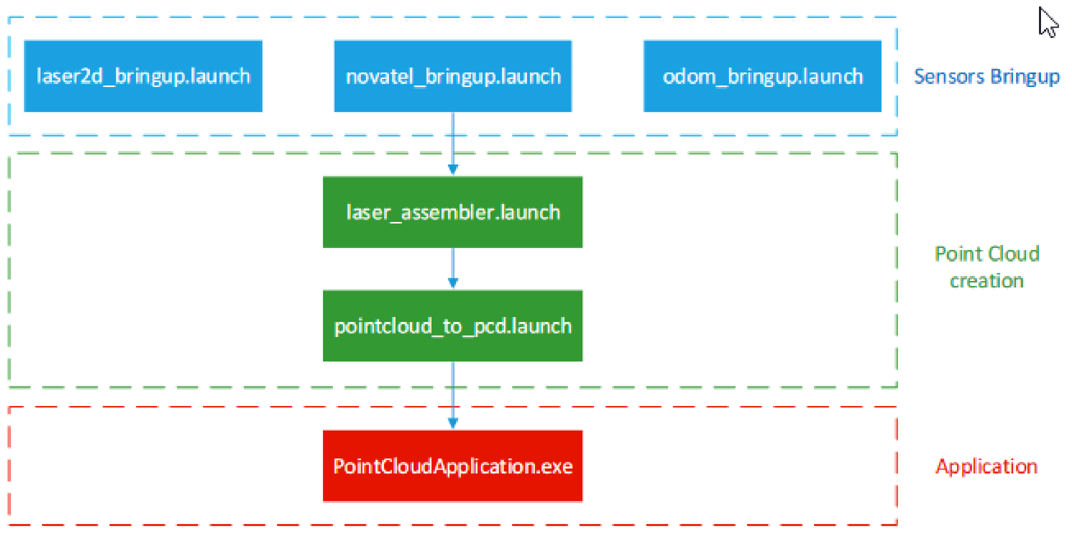


Figure 4.3: Overview of software architecture for the proposed solution.

4.3 Sensors Bring up

The "Sensors Bring Up" procedure acquires all data from the sensors GPS+IMU, LIDAR 2D and odometry.

4.3.1 LIDAR2D bringup

The Laser 2D bringup launch file is designed to initialize and configure one of the 2D LIDAR sensors mounted on the bumper of the ATLASCAR2. This file is versatile, allowing users to specify which LIDAR (left or right) to launch via an argument (name). By default, it is set to initialize the left LIDAR. Within the configuration, the file includes another launch file (`lms1xx.launch`), figure 4.2, setting the IP address and frame ID specifically for the left LIDAR. The inclusion is wrapped in a conditional group that checks if the name argument is set to 'left', ensuring the correct sensor configuration. This setup, ensures the LIDAR data is accurately captured and correctly referenced within the vehicle's coordinate frame.

```
<launch>
   <arg name="name" default="left" doc="lidar␣type␣[left,right]"/>
   <group if="$(eval␣arg('name')␣==␣'left')">
       <include file="$(find␣atlascar2_bringup)/launch/include/
          lms1xx.launch">
            <arg name="port" value="192.168.0.4" />
            <arg name="frame_id" value="left_laser" />
       </include>
   </group>
   <group if="$(eval␣arg('name')␣==␣'right')">
       <include file="$(find␣atlascar2_bringup)/launch/include/
          lms1xx.launch">
            <arg name="port" value="192.168.0.5" />
            <arg name="frame_id" value="right_laser" />
       </include>
   </group>
</launch>
```

Listing 4.2: LIDAR 2D launch file

4.3.2 Novatel bringup

The novatel launch file, figure 4.3, was developed by Pedro Bouça Nova [20] and changed to perform in the new ATLASCAR2 architecture. It is used to initialize and configure the Novatel GPS and IMU sensors for accurate positioning and orientation data in the ATLASCAR2. The configuration section sets up two separate nodelets: one for processing GPS data from the device connected to `/dev/ttyUSB0`, and another for handling IMU data from the device on `/dev/ttyUSB1`. Key parameters are specified for each nodelet to control the types of messages published, such as GPS positions, NMEA messages, IMU data, and diagnostics. Additionally, the launch file includes nodes for publishing geometric transformations using GPS and combined GPS+IMU data to enhance accuracy during low-speed or stationary conditions. A static transform publisher establishes a fixed frame relationship, and an initialization node sets the local coordinate frame origin based on the first GPS fix received. This setup ensures robust and accurate localization.

```
<launch>
    <!-- Node novatel position commands-->
    <node name="novatel_position" pkg="nodelet" type="nodelet"
       args="standalone␣novatel_gps_driver/novatel_gps_nodelet">
    <rosparam>
      connection_type: serial
      device: /dev/ttyUSB0
      ...
    </rosparam>
  </node>
    <!-- Node novatel IMU -->
  <node name="novatel_imu" pkg="nodelet" type="nodelet" args="
     standalone␣novatel_gps_driver/novatel_gps_nodelet">
    <rosparam>
```

```
      connection_type: serial
      device: /dev/ttyUSB1
      ...
    </rosparam>
  </node>
  <node pkg="swri_transform_util" type="gps_transform_publisher"
     name="gps_transform_publisher" output="screen"/>

  <node pkg="swri_transform_util" type="imu_transform_publisher"
     name="imu_transform_publisher" output="screen"/>
  <node pkg="tf" type="static_transform_publisher" name="
     swri_transform" args="0␣0␣0␣1.5708␣0␣0␣/world␣/map␣100"/>
  <node pkg="swri_transform_util" type="initialize_origin.py" name
     ="initialize_origin" output="screen">
        <param name="local_xy_frame" value="/world"/>
        <param name="local_xy_origin" value="auto"/>
        <remap from="gps" to="/gps/fix"/>
  </node>
</launch>
```

Listing 4.3: Novatel launch file

4.3.3 Odom bringup

The odom launch file, figure 4.4 developed by Sara Pombinho [21], is designed to initialize the odometry system for the ATLASCAR2, which is essential for tracking the vehicle's position and movement over time. The configuration section sets an argument, `remote_running`, which determines whether the system should use simulated time, crucial for operations involving pre-recorded data (bag files). This parameter is dynamically set based on the argument's value. The execution section launches two key nodes: `can_ros_msgs_to_ackermann` and `ackermann_to_odom`. The first node converts CAN bus messages from the vehicle into Ackermann steering messages, which are a standard format for steering data in automotive applications. The second node processes these Ackermann messages to compute the vehicle's odometry, providing detailed information about its position and orientation. This setup ensures that the vehicle's movement data is accurately captured and represented, facilitating effective navigation and control in both real-time and simulated environments.

```
<launch>
    <arg name="remote_running" default="false"/>
    <param name="/use_sim_time" value="$(arg␣remote_running)"/>

    <node pkg="atlascar2_odom" type="can_ros_msgs_to_ackermann.py"
         name="can_ros_msgs_to_ackermann" output="screen"/>
    <node pkg="atlascar2_odom" type="ackermann_to_odom.py" name="
       ackermann_to_odom" output="screen"/>
</launch>
```

Listing 4.4: Odom launch file

4.4 Point cloud creation

The conversion of laser scan data into a point cloud format and subsequently saving this data to a file involves several key components and configurations. The primary tools utilized in this process are the `laser_assembler` package, a custom Python script, and the `pcl_ros` package. This section provides a comprehensive guide on setting up and executing the necessary nodes to achieve this transformation.

4.4.1 Configuration

The process begins with the configuration of the `laser_assembler`.launch file. This file initiates the `laser_scan_assembler` node, responsible for collecting and assembling laser scans into a point cloud. The key parameters in this configuration are `max_scans`, which defines the maximum number of scans to assemble, and `fixed_frame`, which sets the reference frame for the point cloud.

The next component is the `laser_to_pcl.py` script. This script initializes a node named `assemble_scans_to_cloud` and sets up a service proxy to the `assemble_scans2` service, which assembles laser scans into a point cloud. The script then creates a publisher to send PointCloud2 messages to the `/laser_pointcloud` topic. Inside the main loop, which runs at 10 Hz, the script calls the service to assemble scans from the beginning of time to the current time, logs the number of points in the resulting cloud, and publishes the point cloud. If the service call fails, an error is logged. The loop ensures continuous operation until the node is shut down, handling any interruptions.

Finally, we have the `pointcloud_to_pcd.launch` file that is configured to start the `pointcloud_to_pcd` node from the `pcl_ros package`. This node subscribes to the `/laser_pointcloud` topic and saves the incoming point cloud messages to Point Cloud Data (PCD) files.

The key parameters here include prefix, which defines the prefix for the saved PCD files, and folder, which specifies the directory where the PCD files will be saved. As the point cloud data is received, it is automatically saved to the designated folder with filenames starting with the defined prefix.

```
<launch>
<node type="point_cloud2_assembler" pkg="laser_assembler"
    name="pcl_assembler_server">
    <remap from="cloud" to="laserPointCloud"/>
    <param name="max_clouds" type="int" value="15000" />
    <param name="fixed_frame" type="string" value="odom" />
</node>
</launch>
```

Listing 4.5: Laser Assembler launch file

4.4.2 Execution

To execute the entire process, the first step is to launch the `laser_assembler`.launch file. This action initiates the `laser_scan_assembler` and custom `laser_to_pcl.py` nodes. These nodes work to assemble laser scans into a point cloud and publish the data to the `/laser_pointcloud` topic.

The next step is to launch the `pointcloud_to_pcd.launch` file to start the `pointcloud_to_pcd` node. This node subscribes to the `/laser_pointcloud` topic and saves the received point cloud messages to PCD files in the specified directory.

By following these steps in this order, laser scan data is effectively converted into a point cloud, and the results are saved as PCD files.

4.5 The Point Cloud Measurements (PCM) Application

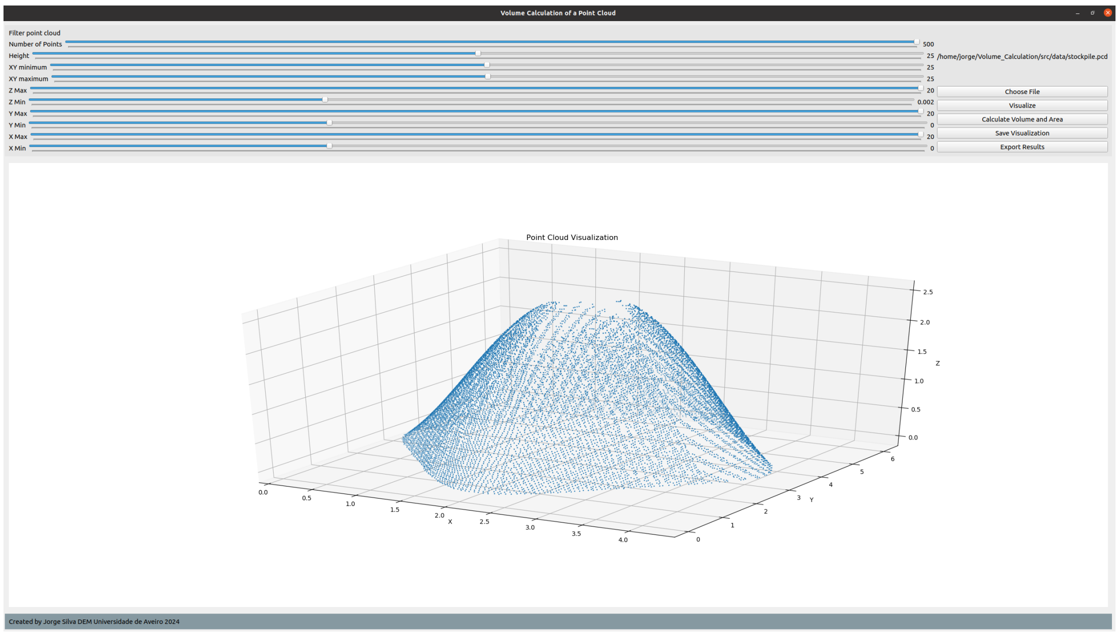
This section presents an interactive application, Point Cloud Measurements (PCM), created for visualization, filtering, volume and area calculation of the point cloud collected.

4.5.1 Architecture

The PCM application (Figure 4.4) was built in python using `PyQt5` to create a GUI with sliders to control the points, windows to visualize the points and the triangulation mesh, buttons to apply visualization and to calculate the volume.

This tool is designed to be platform-independent, run seamlessly on different operating systems, optimize processing performance while handling large-scale point cloud datasets. Efficient algorithms, data structures, and parallel processing techniques are employed to ensure scalability and responsiveness, even with substantial amounts of data. Also, incorporates mechanisms for providing feedback to users, such as warning and information messages using `QMessageBox`.

The architecture adopts a modular design approach, and each component encapsulates specific functionality. This modular structure enhances the codebase's readability, maintainability, and extensibility, allowing for easy integration of new features or improvements in the future.



Volume Calculation of a Point Cloud

Filter point cloud
Number of Points 500
Height 25
XY minimum 25
XY maximum 25
Z Max 20
Z Min 0.002
Y Max 20
Y Min 0
X Max 20
X Min 0

/home/jorge/Volume_Calculation/src/data/stockpile.pcd

Created by Jorge Silva DEM Universidade de Aveiro 2024

Figure 4.4: The PCM Application to visualize the point cloud.

4.5.2 User Interface

The main window of the application consists of three sections. The Control Panel on the left contains buttons, sliders, and labels for file selection and filtering. The Visualization Area in the center displays the 3D plot of the point cloud. The Footer at the bottom displays the creator's information. This layout ensures that all the necessary controls are easily accessible while providing a clear view of the point cloud visualization.

To choose a point cloud file, click the "Choose File" button in the Control Panel. A file dialog will open, allowing you to select a PCD file from your computer. Once selected, the file path will be displayed next to the button, confirming that the file has been loaded successfully.

For visualizing the point cloud, adjust the Number of Points, Height, XY Minimum, and XY Maximum sliders to your desired settings. Then, click the "Visualize" button to display the point cloud in the Visualization Area. The point cloud will be read from the file and filtered based on the slider values, and the filtered point cloud will be displayed as a 3D plot.

To filter the point cloud further, adjust the Z Min/Max, Y Min/Max, and X Min/Max sliders in the Control Panel. These sliders allow you to set the desired range for filtering the point cloud along each axis. After adjusting the sliders, click the "Visualize" button again to update the visualization based on the new filter settings.

To calculate the volume and surface area of the filtered point cloud, click the "Calculate Volume and Area" button. A waiting message will appear while the application performs Delaunay triangulation and calculates the volume and surface area. Once the calculation is complete, the results will be displayed in a message box, and the 3D plot will be updated to show the triangulation.

To save the current visualization, click the "Save Visualization" button. A file dialog will open, allowing you to select the location and format (Portable Network Graphics (PNG) or Joint Photographic Experts Group (JPEG)) to save the plot. Once saved, the plot will be available as an image file in the chosen location. Similarly, to export the results of the volume and area calculations, click the "Export Results" button. A file dialog will open, allowing you to select the location to save the results. The volume and area calculations will be saved as a text file in the chosen location.

Figure 4.5 shows the process for volume and area calculation of the PCM application.

4.5.3 Visualization

The process begins with loading a point cloud file selected by the user, typically in the PCD format. The `open3d` library facilitates this process by parsing the file and extracting the point cloud data, including the spatial coordinates of each point.

Users are provided with options to generate synthetic points based on specific parameters, such as the number of points, height, and XY bounds. `Numpy` integration is leveraged to generate synthetic points, utilizing random values within the specified bounds to simulate additional points.

Two different methods for filtering are used `Passthrough Filtering` and `Voxel Grid Filtering`. `Voxel Grid Filtering` is used to reduce the global number of points with a standard voxel size of 0.05 m, this value was chosen by testing different values and different objects, so the performance can be quick, and the layout stays the same. `Passthrough Filtering` to delete unwanted points. This application offers adjustable

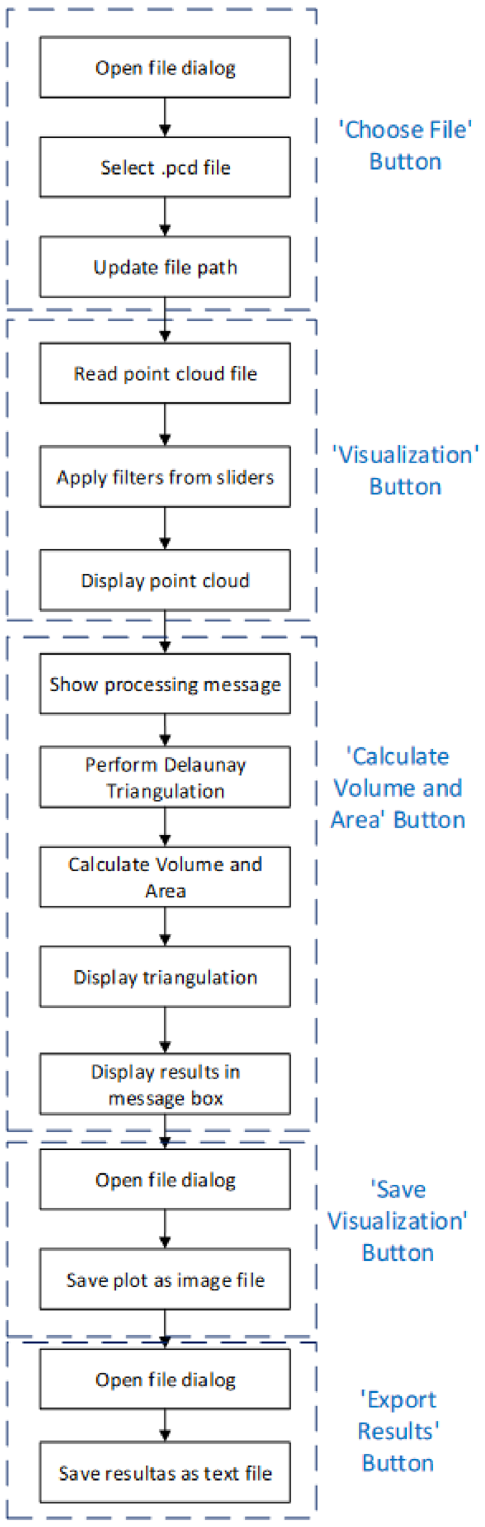


Figure 4.5: Flowchart of the Point Cloud Measurements (PCM) application.

sliders for users to dynamically adjust filtering parameters. These parameters include the minimum and maximum bounds along the XYZ axes for cropping the point cloud data. Figures 4.6 and 4.7 show a point cloud before filtering and after filtering. Additionally, `voxel downsampling` and `passthrough filtering` techniques are applied to refine the point cloud data, reducing noise and computational overhead while preserving essential features.

The visualization process occurs in real-time, allowing users to observe immediate changes resulting from parameter adjustments or filtering operations. Matplotlib's rendering engine efficiently handles the rendering of large point cloud datasets, ensuring smooth performance even with substantial amounts of data.

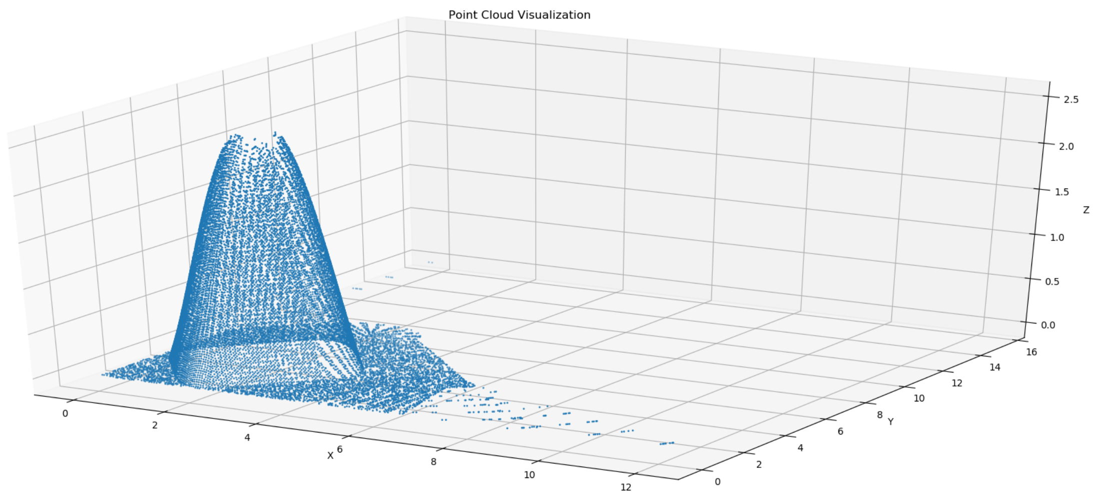


Figure 4.6: Example of a Point Cloud before the filtering process.

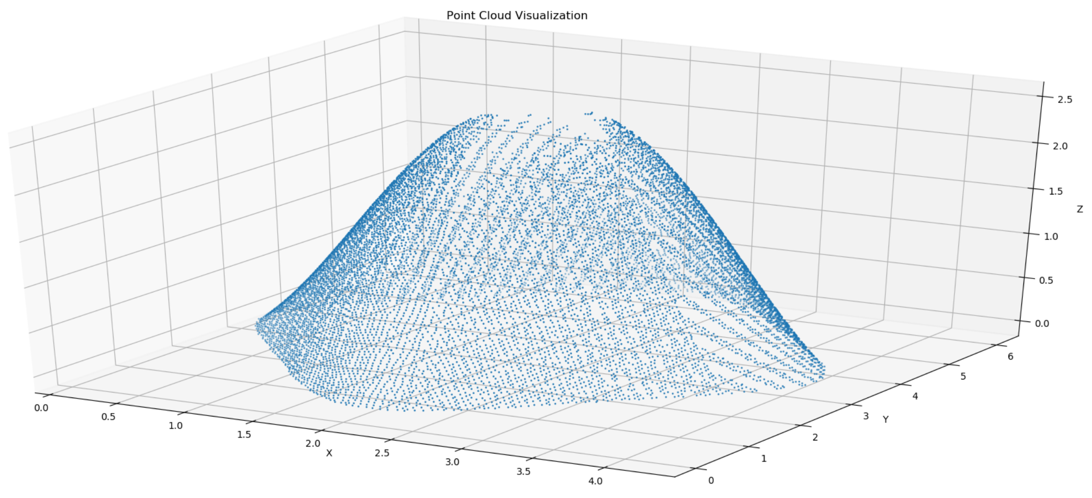


Figure 4.7: Example of a Point Cloud after the filtering process.

4.5.4 Algorithm for Volume Calculation

In Chapter 2 different types of volume calculation algorithms were studied, and it was concluded that Delaunay triangulation combined with projection method is the most suitable for this project. So, this method was chosen to integrate this application.

The process involves several steps. First, the point cloud data is prepared by extracting the XY coordinates of each point. Then, the Delaunay triangulation is computed using the `scipy.spatial.Delaunay` module, which generates a triangulated surface from the point cloud data.

The resulting triangulated surface is visualized using `matplotlib` 3D plot functionality, with each triangle represented as a set of connected vertices. Once the Delaunay triangulation is computed, the calculation layer iterates through the triangles formed by the triangulation to calculate the volume and surface area of the point cloud data.

To calculate the volume under a triangle (Figure 4.8), the vertices are retrieved, and projected in the XY plane, by removing the Z value. Then the area of the triangle is

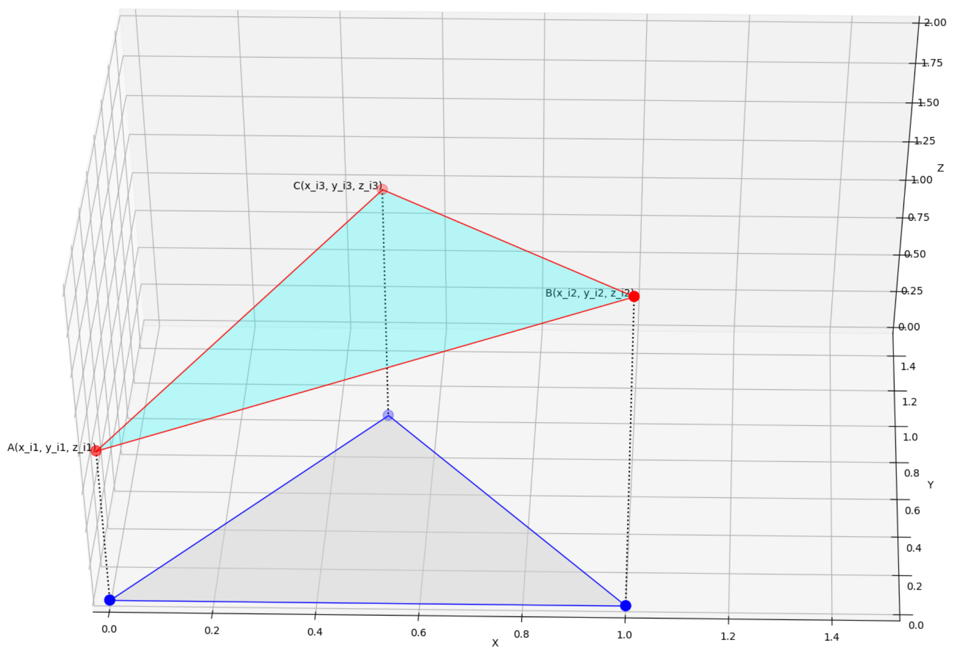


Figure 4.8: Volume under triangle representation.

computed using a suitable method, such as the shoelace formula (4.1). Additionally, the height of the corresponding triangular prism is determined as the average of the z-coordinates of the triangle's vertices:

$$A_i = \frac{1}{2}|(x_{i1}y_{i2} + x_{i2}y_{i3} + x_{i3}y_{i1}) - (x_{i1}y_{i3} + x_{i2}y_{i1} + x_{i3}y_{i2})| \tag{4.1}$$

$$H_i = \frac{1}{3}(z_{i1} + z_{i2} + z_{i3}) \tag{4.2}$$

where $i = 1, 2, \ldots, N$ and N is the total number of triangles in the Delaunay triangulation. $A(x_{i1}, y_{i1}, z_{i1})$, $B(x_{i2}, y_{i2}, z_{i2})$, and $C(x_{i3}, y_{i3}, z_{i3})$ are the vertices for each triangle.

Once the area and height are obtained, the volume of the triangular prism is calculated as the product of its area and height:

$$V_i = A_i \times H_i \tag{4.3}$$

To obtain the total volume V_{total} of the point cloud, it is necessary to sum the volumes under all triangles:

$$V_{\text{total}} = \sum_{i=1}^{N} V_i \tag{4.4}$$

The calculated volume and area are then displayed in a message box, providing users with quantitative information about the point cloud data's characteristics.

4.6 Documentation

All documents are stored in the GitHub repository `https://github.com/jorgemfs06/Thesis.git`.

Chapter 5

Tests and Results

This chapter is divided in two types of results. First, the solution was tested in a simulation environment and then using real data collected from field experiments.

5.1 Simulation Results

The first phase of testing and the acquisition of results was carried out in a simulation environment using Gazebo, an open-source simulation software. In this phase, a world file was created that contains specific objects designed to simulate stockpiles. A point cloud was generated for these objects, and their volumes were subsequently calculated. This preliminary testing provided a controlled environment to validate the system's functionality and refine the algorithms used for data processing and volume calculation.

5.1.1 Create worlds in Gazebo

Three different types of objects were created for the simulation: a cube, a truncated pyramid, and an object with a stockpile format. These objects were initially designed using SolidWorks, a Computer Aided Design (CAD) software. The designs were then exported to a Digital Asset Exchange (DAE) format. The DAE file format, based on the collaborative Design Activity (COLLADA) standard, is widely used for storing and exchanging digital assets, including 3D models, textures, animations, and other related data. These files are essential in Gazebo for defining 3D models of objects, robots, and entire environments.

After the 3D models were created and exported, they were incorporated into world format files in Gazebo. These world files, written in Simulation Description Format (SDF), are crucial as they define the environment for the simulation. The SDF files describe various elements of the simulation environment such as the terrain, lighting, objects, robots, and their initial configurations.

Figures 5.1, 5.2 and 5.3 show the objects that were created in the Gazebo environment.

5.1.2 Create point cloud

As mentioned in Section 4.4, creating a point cloud requires setting a fixed frame of reference. In this simulation, the fixed frame was set to the topic `odom`, as it was already

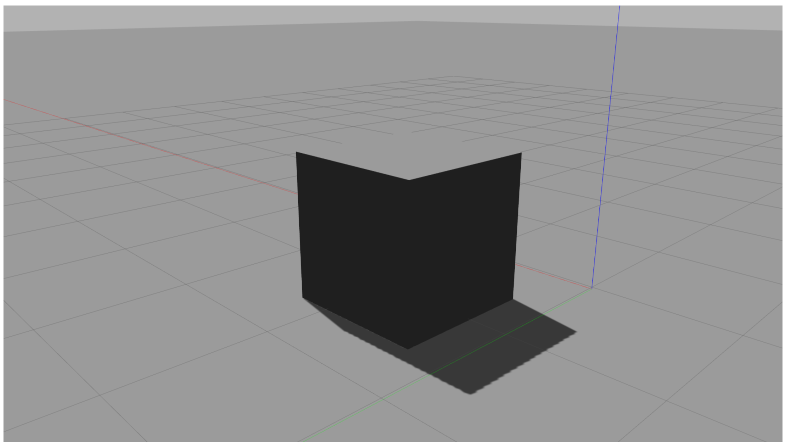


Figure 5.1: Cube in Gazebo.

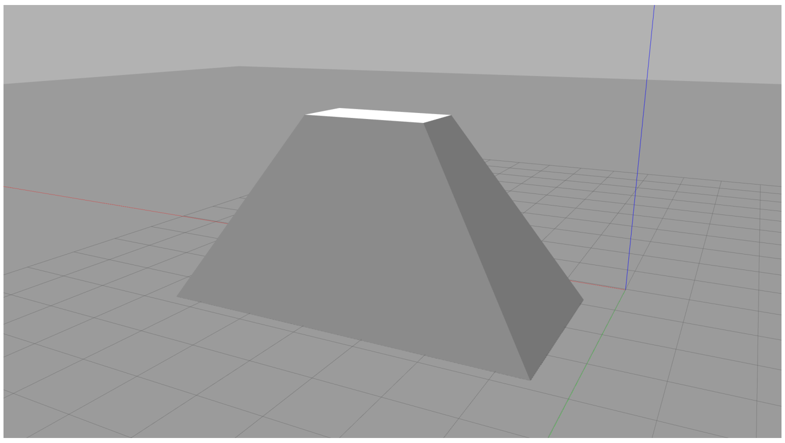


Figure 5.2: Truncated pyramid in Gazebo.

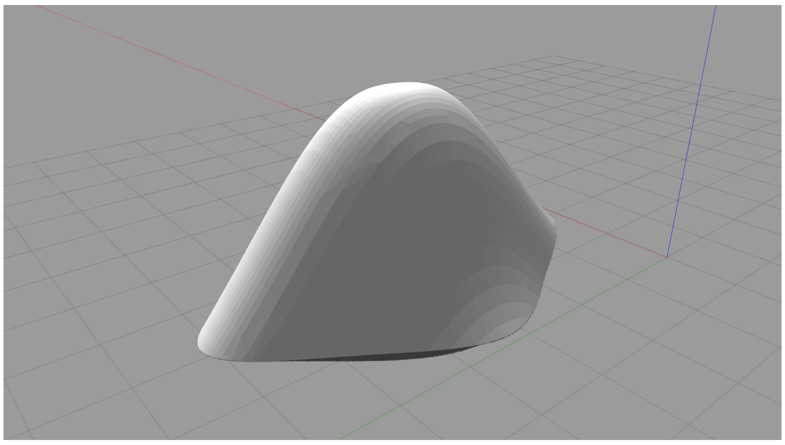


Figure 5.3: Stockpile in Gazebo.

implemented in the ATLASCAR2 repository. The data was gathered by publishing a Twist message on the `ackermann_steering_controller/cmd_vel` topic using the rqt

ROS tool, which allows the user to control the robot. Figure 5.4 shows the created simulation environment with the stockpile object in place.

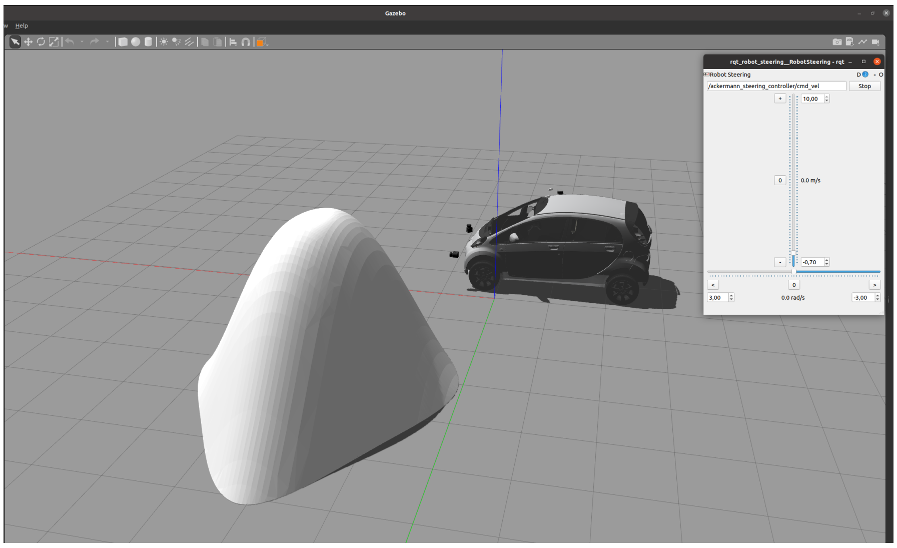

Gazebo
Help
rqt_robot_steering__RobotSteering - rqt
Robot Steering
/ackermann_steering_controller/cmd_vel
Stop
+
10,00
0
0.0 m/s
-
-0,70
<
0
>
3,00
0.0 rad/s
-3,00

Figure 5.4: Simulation in Gazebo.

By using the `laser_assembler` launch mentioned in section 4.4 it is possible to obtain a point cloud. Figure 5.5 shows the point cloud of the stockpile object.

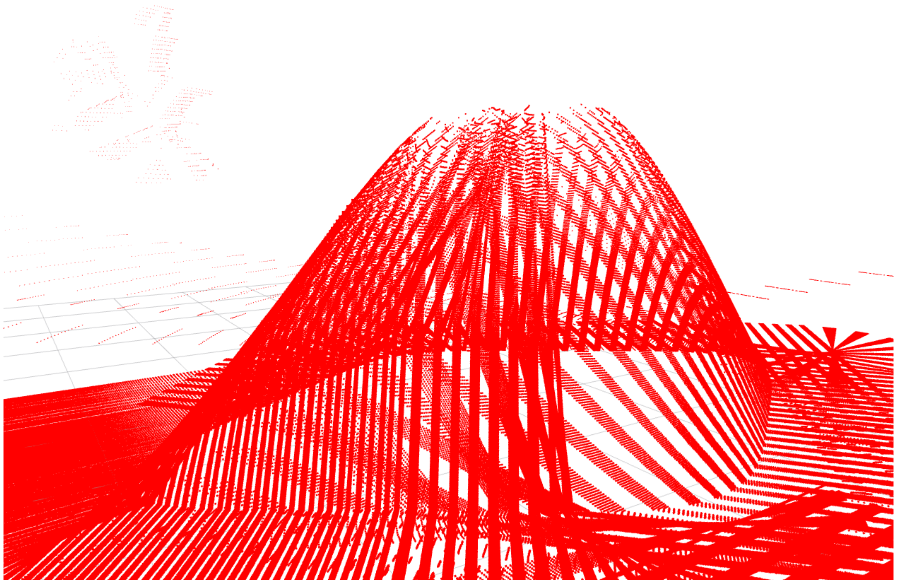
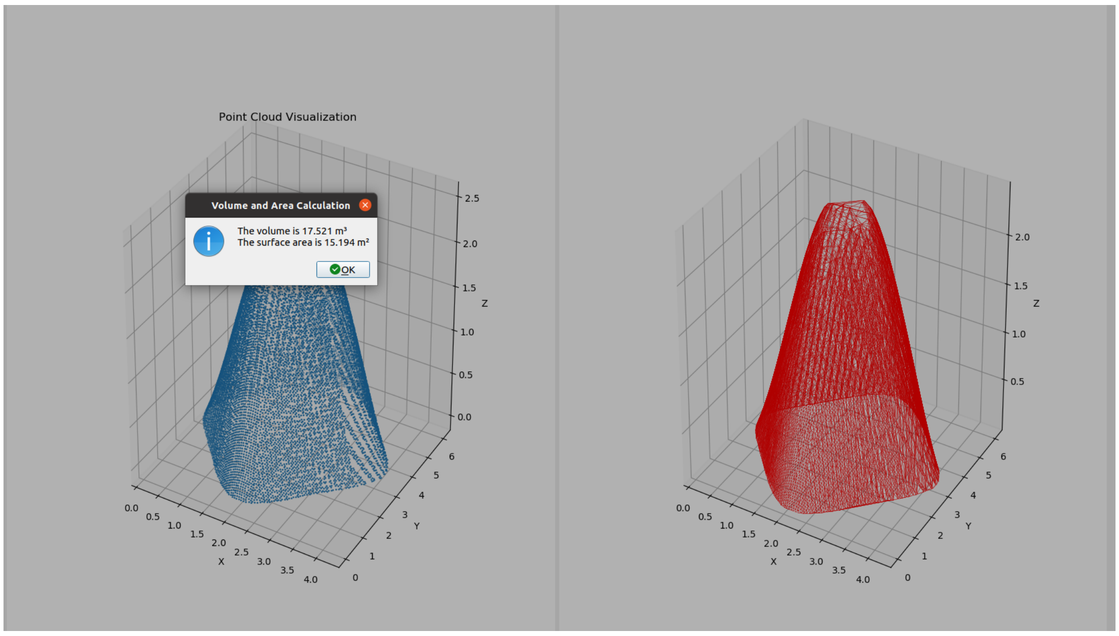


Figure 5.5: Stockpile point cloud.

5.1.3 Area and volume calculation

Upon generating a point cloud, the volume and area can be obtained using the application described in Section 4.5.

Figure 5.6 shows the volume and area of the stockpile object.



The volume is 17.521 m³
The surface area is 15.194 m²

Figure 5.6: Stockpile volume and area for a simulated object.

Object	Solidworks Area (m^2)	Point Cloud Area (m^2)	Absolute Error (m^2)	Relative Error (%)
Cube	1	1.031	0.031	3.1
Truncated pyramid	18	17.999	0.001	0.05
Stockpile	15.068	15.194	0.126	0.84

Table 5.1: Area for each simulated object.

Object	Solidworks Volume (m^3)	Point Cloud Volume (m^3)	Absolute Error (m^3)	Relative Error (%)
Cube	1	0.98	0.020	2
Truncated pyramid	26	25.97	0.030	0.11
Stockpile	17.630	17.521	0.109	0.61

Table 5.2: Volume for each simulated object.

The error analysis presented in table 5.1 and table 5.2 indicates that the point cloud method for calculating area and volume is generally accurate, with most relative errors being quite small.

For the cube, the area and volume calculations have relative errors of 3.1% and 2%, respectively, which are slightly higher compared to the other objects but still within an acceptable range.

The truncated pyramid shows the highest accuracy, with minimal errors: a relative error of 0.05% for the area and 0.11% for the volume. This suggests that the method is highly reliable for geometric shapes with well-defined surfaces.

The stockpile, which has a more complex and irregular shape, shows a relative error of 0.84% for the area and 0.61% for the volume, indicating that while the method remains precise, the complexity of the object's shape can slightly increase the error margins.

Overall, the point cloud method demonstrates robust performance across different objects, with the errors remaining relatively low and within practical limits for most applications.

5.2 Results in Real Scenarios

Following the successful validation in the simulation environment, we proceeded to the second phase of testing in a real-world setting to further substantiate our findings and evaluate the system's performance under actual operating conditions. Due to logistical constraints, it was not possible to test the system on a stockpile as initially planned. Instead, two locations were identified, figure 5.7 and figure 5.8, with clusters of houses that allowed for complete navigation around them, making them suitable substitutes for stockpiles.

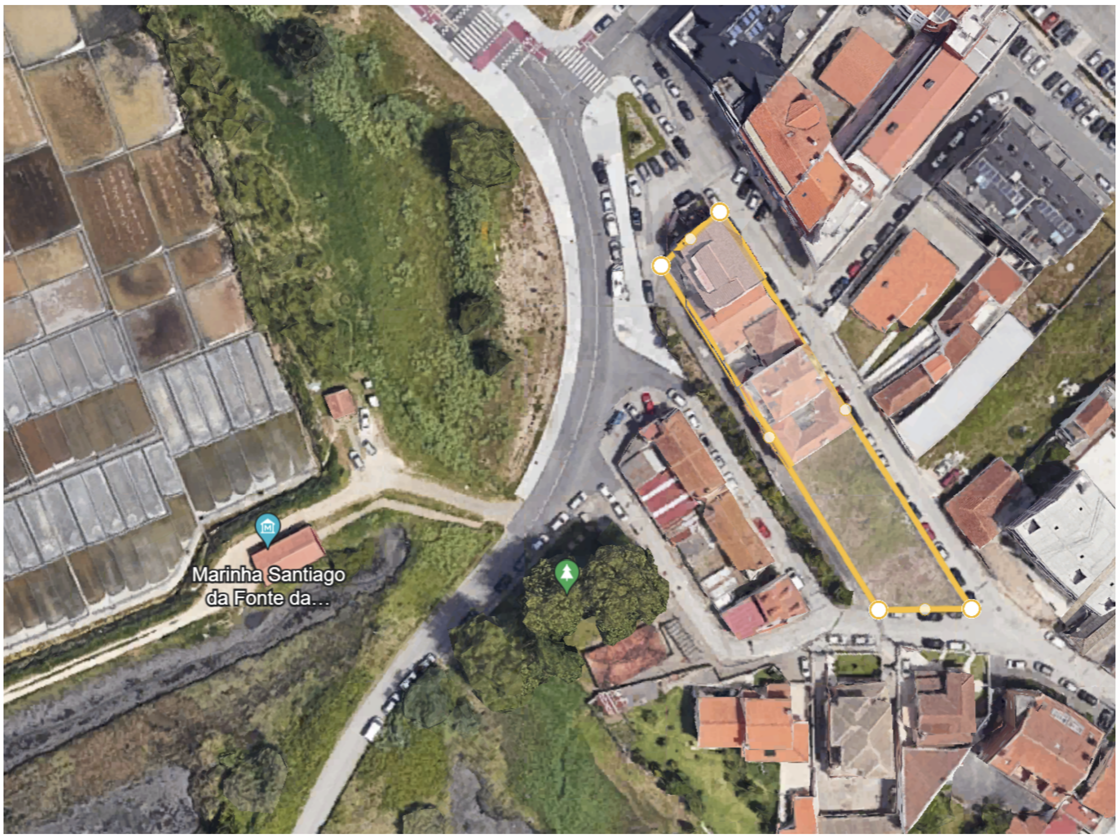


Figure 5.7: First test location.

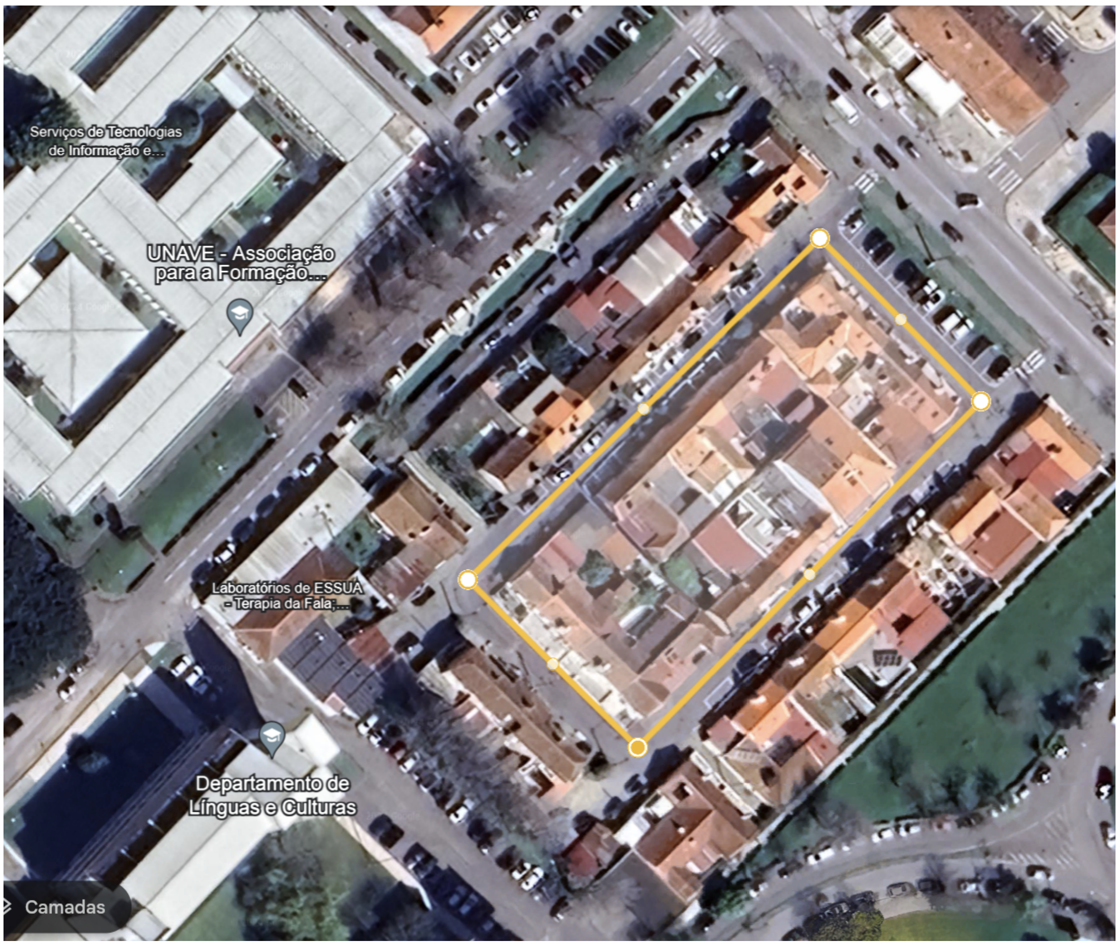

Serviços de Tecnologias de Informação e...
UNAVE - Associação para a Formação...
Laboratórios de ESSUA - Terapia da Fala;...
Departamento de Línguas e Culturas
Camadas

Figure 5.8: Second test location.

5.2.1 Create point cloud

The collected LIDAR and GNSS data is processed with `laser_assembler` launch mentioned in section 4.4. Figure 5.9 and figure 5.10 shows the point clouds for the clusters of houses.

In the first location, a test was conducted on a road with varying inclinations to assess the performance of the LIDAR 2D sensor and GNSS system under different terrain conditions. This site provided a challenging environment to evaluate how changes in elevation and slope affected the accuracy of the point cloud data and subsequent volume calculations. During this test, it was observed that the position of the laser encountered issues, likely because of the uneven terrain that affected the stability and alignment of the sensor. In contrast, the second location featured a flat road with no inclinations, providing a more controlled environment to test the system's performance without the added variable of changing inclinations. The comparison between these two environments highlighted the impact of road inclinations on the data collection process and the overall accuracy of the system.

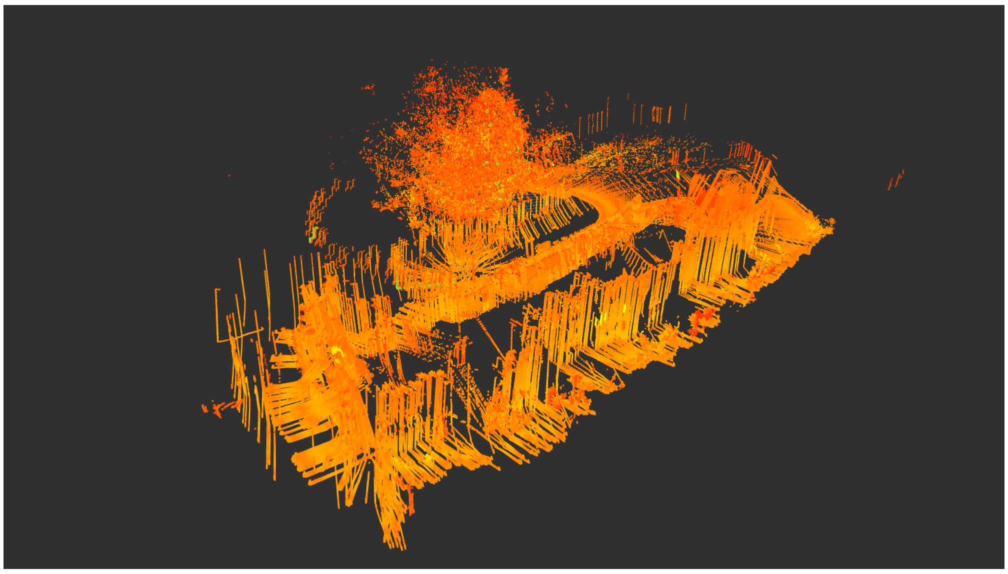


Figure 5.9: First cluster of houses point cloud.

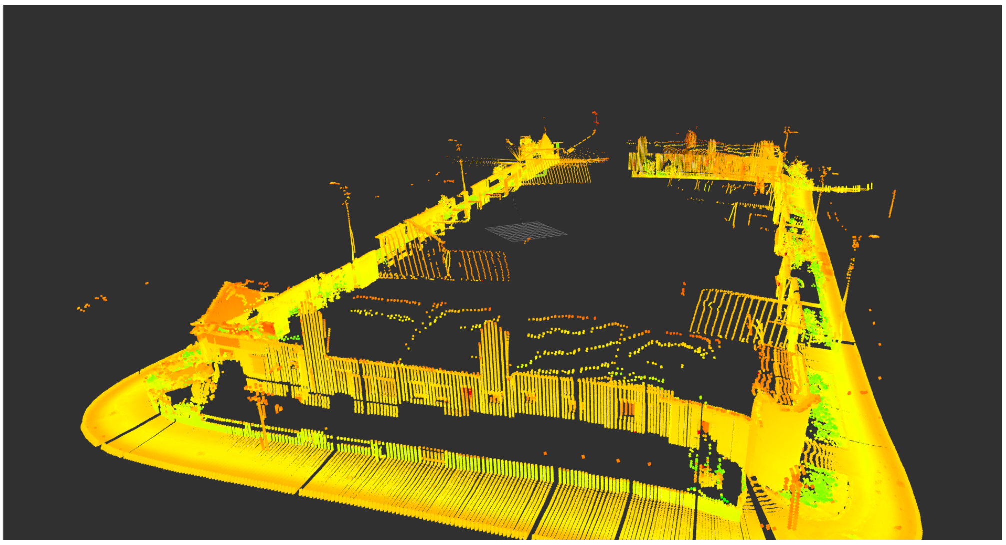


Figure 5.10: Second cluster of houses point cloud.

5.2.2 Area and volume calculation

Once the point cloud is generated, the volume and area can be obtained using the application described in section 4.5.

Figure 5.11 shows the point cloud of the second cluster of houses before cropping and figure 5.12 after applying the crop.

The result for the volume and area calculation is represented in figure 5.13. Also, Table 5.3 summarizes and compares results with reference values for the second experiment, whose point cloud is depicted in Figure 5.10.

According to Google Earth, the cluster has an area of approximately 2749.27 m^2. The PCM application calculated an area of 2760.109 m^2 and a volume of 14 323.622 m^3. The

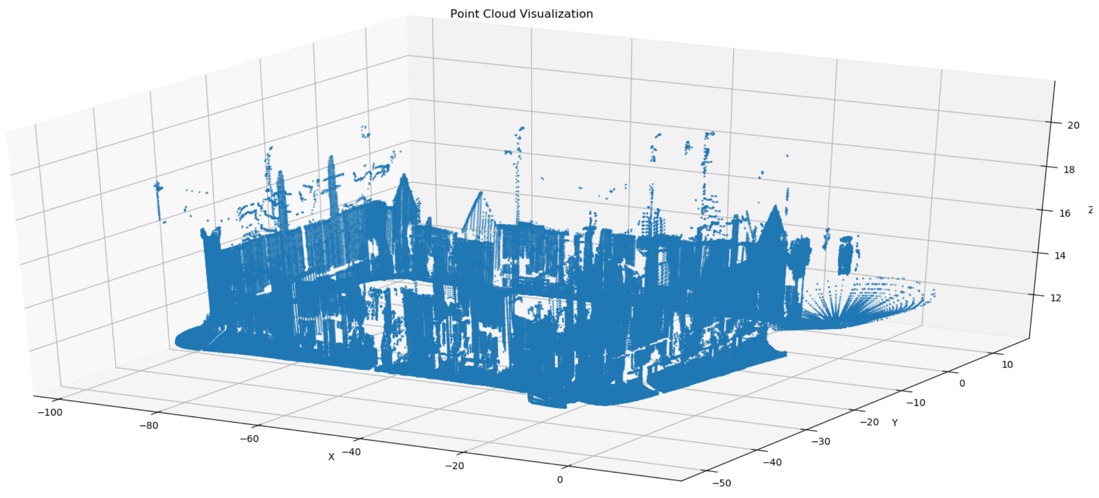


Figure 5.11: Second cluster of houses point cloud before crop.

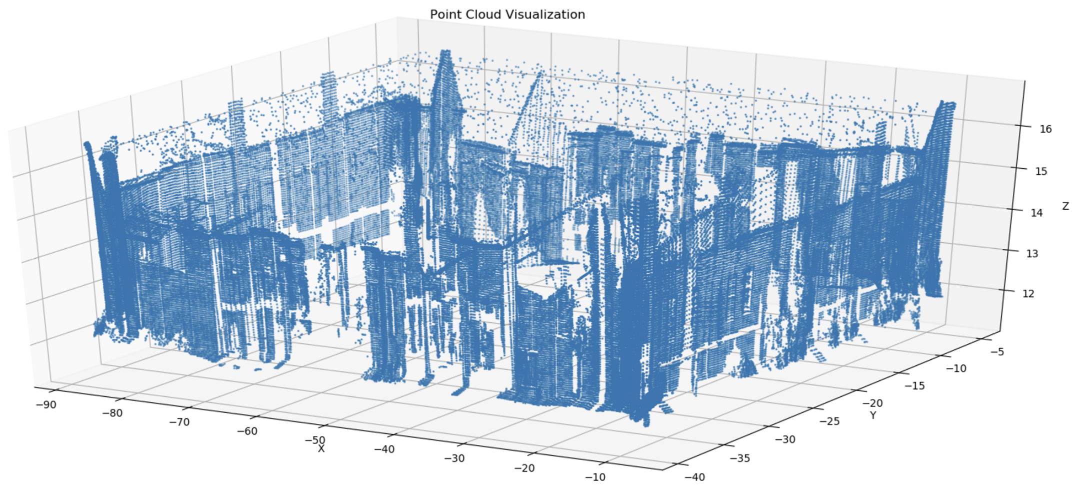
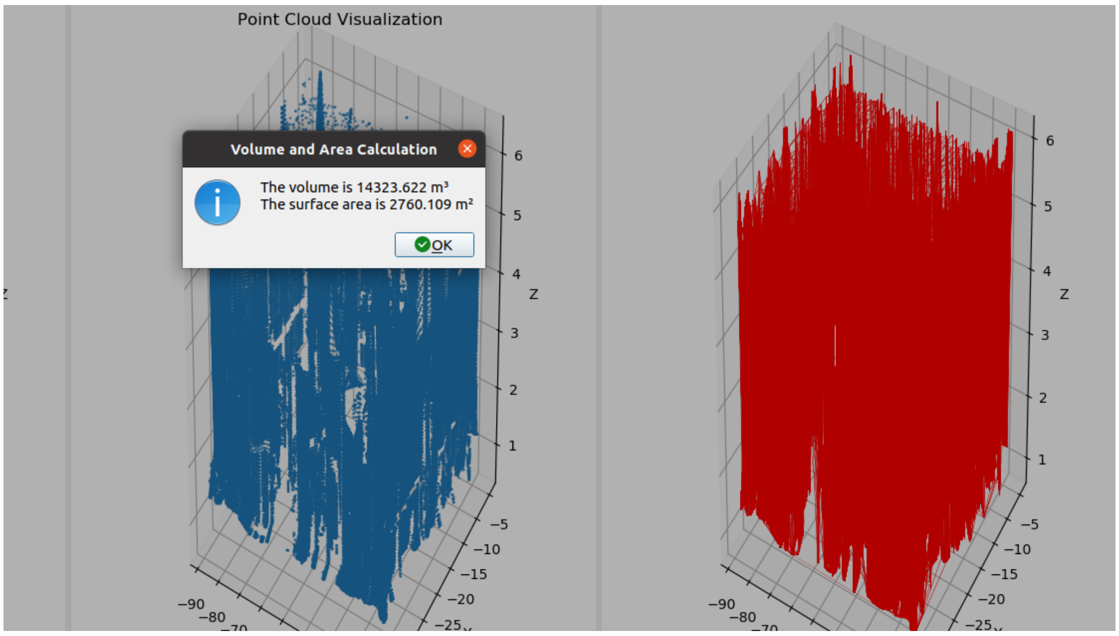


Figure 5.12: Second cluster of houses point cloud after crop.

Parameter	**Reference Value**	**Calculated Value**	**Percentage Error (%)**
Area (m^2)	2749.27	2760.109	0.39
Volume (m^3)	13 746.35	14 323.622	4.20

Table 5.3: Second cluster of houses volume and area results.



The volume is 14323.622 m³
The surface area is 2760.109 m²

Figure 5.13: Second cluster of houses volume and area results.

difference in area is $10.839\,\mathrm{m}^2$, a percentage error of approximately 0.39%, indicating high accuracy in area calculation. For volume calculation, assuming an average height of 5 meters for single-story houses (considering floor heights between 2.5 to 3 meters and roof heights between 1.5 to 2.5 meters, resulting in total heights between 4 to 5.5 meters), the expected volume is around $13\,746.35\,\mathrm{m}^3$. The calculated volume results in a percentage error of approximately 4.20%, suggesting reasonable accuracy.

Overall, the PCM application proves to be a reliable tool for calculating the area and volume of clusters of single-story houses using point cloud data, with results consistent with expected values from external measurements. Future enhancements could improve volume accuracy by integrating more detailed height data and refining point cloud processing algorithms.

Intentionally blank page.

Chapter 6

Conclusions and Future Work

This chapter is a summary of the work developed to fulfill the objectives of this thesis. Based on these conclusions, a few proposals for future works are presented next.

6.1 Conclusions

This thesis presents a comprehensive solution for georeferenced outdoor volume measurement using LIDAR and GNSS technologies mounted on the ATLASCAR2 vehicle. The system integrates a SICK LMS151 LIDAR sensor and a Novatel SPAN-IGM-A1 GNSS/INS system to capture high-resolution point cloud data, which is processed through a Python-based application that incorporates PyQt5 for a graphical user interface (GUI) and algorithms for point cloud manipulation and volume estimation.

The methodology developed included both simulation and real-world testing. In the simulation phase, the system was tested on geometrically simple objects (cube, truncated pyramid, and stockpile) using the Gazebo environment. The data was collected through ROS-based control of the ATLASCAR2, with point clouds generated for each object. The subsequent area and volume calculations demonstrated excellent accuracy, particularly with the truncated pyramid, which had an error margin of 0.05% for area and 0.11% for volume. Even for the more irregularly shaped stockpile, the system achieved a relative error of 0.84% for area and 0.61% for volume, showing that the method can handle complex geometries effectively.

In real-world tests, two locations with clusters of houses were used as substitutes for stockpiles due to logistical constraints. Volume calculations were slightly less accurate, with an error margin of 4.20% in real-world settings and between 0.61% and 2% in simulation. The results demonstrated that flat surfaces yielded more consistent and accurate data, while uneven terrain affected sensor stability and data quality. This highlights the importance of terrain analysis in future implementations.

The Delaunay Triangulation algorithm for volume estimation was chosen due to its robustness in handling irregular and complex surfaces. This algorithm connects non-collinear points into triangles, forming a smooth surface reconstruction suitable for volumetric calculations. In simulations, Delaunay Triangulation achieved precision with error margins consistently below 1% for simple geometries and under 2% for complex shapes, making it ideal for stockpile volume estimation. In real-world tests, minor losses in accuracy were noted due to sensor noise and terrain variability, but the algorithm remained effective in moderately complex environments.

For point cloud filtering, two primary algorithms were chosen: Voxel Grid Filtering and Passthrough Filtering. Voxel Grid Filtering was used to reduce the number of points in the point cloud, which optimized the computational load without sacrificing geometrical accuracy. This was essential for handling large datasets efficiently. Passthrough Filtering allowed the system to focus on specific areas of interest in removing irrelevant data outside a predefined range. These two filters worked together to enhance point cloud clarity, ensuring high-quality data was passed to the Delaunay Triangulation algorithm for volume calculation.

The system's modular design allows it to efficiently process large point cloud datasets, making it applicable for various real-world uses, particularly in environmental mapping, stockpile volume estimation, and other large-scale volumetric analysis tasks. Future improvements could enhance accuracy, particularly in real-world environments, by conducting tests with more controlled object geometries rather than clusters of buildings.

In summary, the integration of GNSS and LIDAR on ATLASCAR2, along with the use of the Delaunay Triangulation algorithm and effective filtering methods such as Voxel Grid and Passthrough Filtering, provides a robust and reliable system for volume and area estimation. This system shows promising potential for further development and broader applications in the field of geospatial measurement.

6.2 Future work

To further enhance the system, several areas of improvement and expansion are proposed:

- Incorporating inclinometers to get the inclination of the terrain.
- Further development and refinement of the point cloud processing algorithms could improve both the accuracy and efficiency of the system.
- Improve navigation of ATLASCAR2 by merging GPS+IMU data, odometry and inclinometers.
- Developing real-time processing capabilities would enable the system to be used in dynamic environments, providing immediate feedback and results.
- Conducting more extensive tests in varied environments and conditions would help validate and improve the robustness of the system.
- Improving the user interface to be more intuitive and user-friendly can facilitate broader adoption and easier operation by non-technical users.

Finally, expanding the system's application to other types of structures and environments can open up new use cases and enhance its versatility.

References

[1] ATLASCAR1 Description. https://atlas.web.ua.pt.

[2] ATLASCAR2 Description. https://github.com/lardemua/atlascar2.

[3] Farnell. ri32-0/1000er.14kb. https://pt.farnell.com/hengstler/ri32-0-1000er-14kb/encoder-rotary/dp/615985.

[4] Gazebo. https://gazebosim.org/home.

[5] Laser Assembler Package. http://wiki.ros.org/laser_assembler.

[6] LMS151-10100 — Detection and ranging solutions — SICK. https://www.sick.com/us/en/detection-and-ranging-solutions/2d-lidar-sensors/lms1xx/lms151-10100/p/p141840.

[7] Novatel. Gps-702-gg-n antenna. https://portal.hexagon.com/public/Novatel/assets/Documents/Manuals/om-20000095.

[8] Novatel. Span-igm user manual. https://portal.hexagon.com/public/Novatel/assets/Documents/Manuals/OM-20000141.

[9] Putra, Chrystia Aji and Wahyu Syaifullah, J. S. and Adila, Mar'Atul, Approximate volume of sand materials stockpile based on structure from motion (SFM). *Proceeding - 6th Information Technology International Seminar, ITIS 2020.* https://www.researchgate.net/publication/348673018_Approximate_Volume_of_Sand_Materials_Stockpile_Based_on_Structure_From_Motion_SFM.

[10] The python standard library. https://docs.python.org/3/library/index.html.

[11] Ros/introduction - ros wiki. http://wiki.ros.org/ROS/Introduction.

[12] rviz - ROS Wiki. http://wiki.ros.org/rviz.

[13] Siwen Quan, Jie Ma, Fangyu Hu, Bin Fang and Tao Ma, Local voxelized structure for 3D binary feature representation and robust registration of point clouds from low-cost sensors, 2018. https://www.sciencedirect.com/science/article/abs/pii/S0020025518301646.

[14] Tucci, Grazia and Gebbia, Antonio and Conti, Alessandro and Fiorini, Lidia and Lubello, Claudio, Monitoring and computation of the volumes of stockpiles of bulk material by means of UAV photogrammetric surveying, 2019. https://www.researchgate.net/publication/333937847_Monitoring_and_

Computation_of_the_Volumes_of_Stockpiles_of_Bulk_Material_by_Means_of_UAV_Photogrammetric_Surveying.

[15] Wen-Chung Chang and Chia-Hung Wu, Object Volume Estimation Based on 3D Point Cloud, 2017. https://ieeexplore.ieee.org/document/8284244.

[16] Yong Liu and Yanwei Zheng, Accurate Volume Calculation Driven by Delaunay Triangulation for Coal Measurement, 2021. https://www.researchgate.net/publication/350935996_Accurate_Volume_Calculation_Driven_by_Delaunay_Triangulation_for_Coal_Measurement.

[17] Yusen Geng, Yuankai Zhang, Xincheng Tian, Xiaorui Shi, Xiujing Wang, and Yigang Cui. A method of welding path planning of steel mesh based on point cloud for welding robot. 2021. https://www.researchgate.net/publication/350716828_A_Method_of_Welding_Path_Planning_of_Steel_Mesh_Based_on_Point_Cloud_for_Welding_Robot.

[18] Rocio Mora, Jose A. Jimenez, Susana Lagüela, and Diego González-Aguilera. Automatic point-cloud registration for quality control in building works. 2021. https://www.researchgate.net/publication/349121563_Automatic_Point-Cloud_Registration_for_Quality_Control_in_Building_Works.

[19] Matteo Munaro, Filippo Basso, Stefano Michieletto, Enrico Pagello, and Emanuele Menegatti. A software architecture for rgb-d people tracking based on ros framework for a mobile robot. 2013. https://www.researchgate.net/publication/286624443_A_Software_Architecture_for_RGB-D_People_Tracking_Based_on_ROS_Framework_for_a_Mobile_Robot.

[20] Pedro Miguel Bouça Nova. Localização e navegação global do atlascar2 usando gnss e interface com mapas on-line. Master's thesis, Universidade de Aveiro, 2018. http://hdl.handle.net/10773/29002.

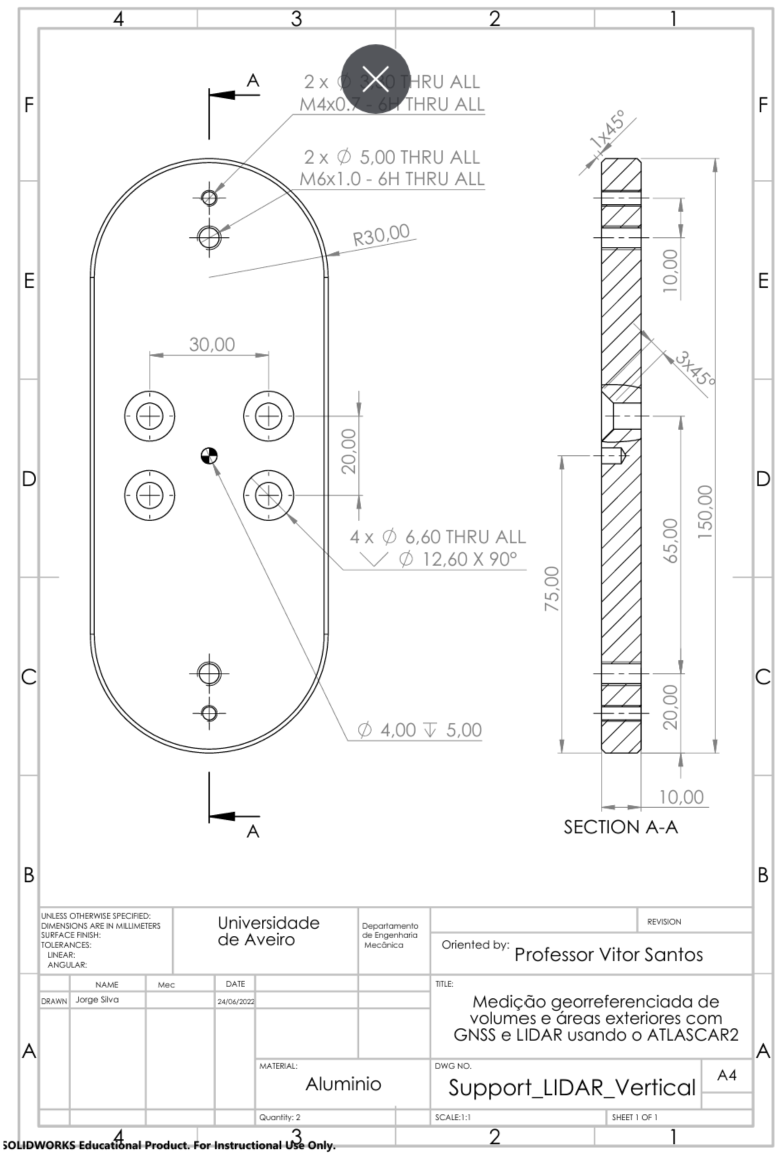
[21] Sara Pombinho. Integrated software architecture in atlascar2. Master's thesis, Universidade de Aveiro, 2022. http://hdl.handle.net/10773/34924.

[22] Gigih Priyandoko, T. Ming, and M.S. Achmad. Mapping of unknown industrial plant using ros-based navigation mobile robot. *IOP Conference Series: Materials Science and Engineering*, 257, 10 2017.

[23] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, 2011.

Appendix A

LIDAR Support



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

Universidade de Aveiro

Departamento de Engenharia Mecânica

REVISION

Oriented by: Professor Vitor Santos

	NAME	Mec	DATE
DRAWN	Jorge Silva		24/06/2022

TITLE: Medição georreferenciada de volumes e áreas exteriores com GNSS e LIDAR usando o ATLASCAR2

MATERIAL: Aluminio

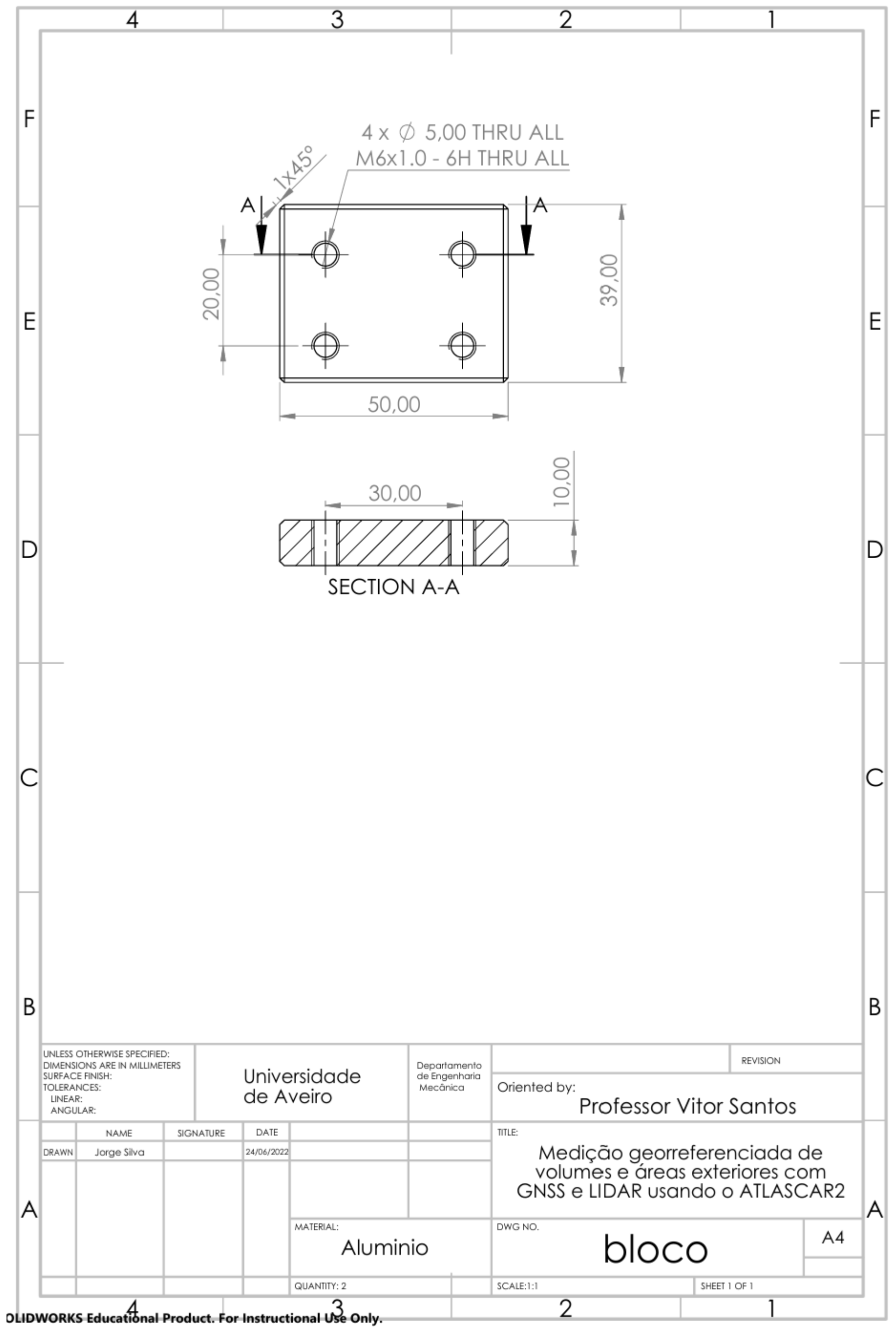
DWG NO. Support_LIDAR_Vertical

A4

Quantity: 2

SCALE:1:1

SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

Universidade de Aveiro

Departamento de Engenharia Mecânica

REVISION

Oriented by: Professor Vitor Santos

	NAME	SIGNATURE	DATE
DRAWN	Jorge Silva		24/06/2022

TITLE: Medição georreferenciada de volumes e áreas exteriores com GNSS e LIDAR usando o ATLASCAR2

MATERIAL: Aluminio

DWG NO. bloco

A4

QUANTITY: 2

SCALE:1:1

SHEET 1 OF 1

Appendix B

Laser Assembler

```
import rospy
from laser_assembler.srv import AssembleScans2
from sensor_msgs.msg import PointCloud2

def assemble_and_publish():
    rospy.init_node("assemble_scans_to_cloud")
    rospy.wait_for_service("assemble_scans2")
    assemble_scans = rospy.ServiceProxy('assemble_scans2',
       AssembleScans2)
    pub = rospy.Publisher ("/laser_pointcloud", PointCloud2,
       queue_size=1)

    rate = rospy.Rate(0.1)  # Adjust the loop rate as needed

    while not rospy.is_shutdown():
        try:
            resp = assemble_scans(rospy.Time(0), rospy.get_rostime
               ())
            rospy.loginfo("Got␣cloud␣with␣%u␣points" % len(resp.
               cloud.data))
            pub.publish(resp.cloud)
        except rospy.ServiceException as e:
            rospy.logerr("Service␣call␣failed:␣%s" % e)

        rate.sleep()

if __name__ == "__main__":
    try:
        assemble_and_publish()
    except rospy.ROSInterruptException:
        pass
```

Intentionally blank page.

Appendix C

The PCM application

```
import sys
import matplotlib.pyplot as plt
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout,
    QHBoxLayout, QLabel, QFileDialog, QPushButton,
    QSlider, QMessageBox
)
import open3d as o3d
import numpy as np
from scipy.spatial import Delaunay
from functools import reduce
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
   as FigureCanvas
from mpl_toolkits.mplot3d import Axes3D

class PointCloudVisualizer(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Volume␣Calculation␣of␣a␣Point␣Cloud")
        self.showMaximized()
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)
        self.main_layout = QVBoxLayout()
        self.central_widget.setLayout(self.main_layout)

        # Create widget for buttons and sliders
        self.control_widget = QWidget()
        self.control_layout = QVBoxLayout()
        self.control_widget.setLayout(self.control_layout)
        self.control_widget.setStyleSheet("background-color:␣#
           e6e6e6;")

        # Left side layout for sliders and buttons
        self.left_layout = QHBoxLayout()
        self.control_layout.addLayout(self.left_layout)

        # Right side layout for plots
        self.right_layout = QVBoxLayout()
        self.control_layout.addLayout(self.right_layout)

        # Layout for buttons
        self.button_layout = QVBoxLayout()

        self.create_points_layout = QVBoxLayout()
        self.create_points_layout.addWidget(QLabel("Filter␣point␣
           cloud"))
        self.number_points_slider = self.create_slider("Number␣of␣
           Points", 0, 500, 500)
        self.height_slider = self.create_slider("Height", 0, 50,
           25)
        self.xy_min_slider = self.create_slider("XY␣minimum",
           -100, 100, 25)
        self.xy_max_slider = self.create_slider("XY␣maximum",
           -100, 100, 25)
        self.left_layout.addLayout(self.create_points_layout)

        self.filter_layout = QVBoxLayout()
        self.z_max_slider = self.create_slider("Z␣Max", -100, 100,
            100)
        self.z_min_slider = self.create_slider("Z␣Min", -100, 100,
            0)
        self.y_max_slider = self.create_slider("Y␣Max", -100, 100,
            100)
        self.y_min_slider = self.create_slider("Y␣Min", -100, 100,
            -100)
        self.x_max_slider = self.create_slider("X␣Max", -100, 100,
            100)
        self.x_min_slider = self.create_slider("X␣Min", -100, 100,
            -100)
        self.left_layout.addLayout(self.filter_layout)

        # Add buttons and sliders to the left layout
        self.point_cloud_file_label = QLabel("")
        self.point_cloud_file_path = ""
        self.point_cloud_file_button = QPushButton("Choose␣File")
        self.point_cloud_file_button.clicked.connect(self.
           choose_point_cloud_file)
        self.button_layout.addWidget(self.point_cloud_file_label)
        self.button_layout.addWidget(self.point_cloud_file_button)

        self.visualize_button = QPushButton("Visualize")
        self.visualize_button.clicked.connect(self.
           visualize_point_cloud)
        self.button_layout.addWidget(self.visualize_button)

        self.calculate_volume_delaunay_button = QPushButton("
           Calculate␣Volume␣and␣Area")
        self.calculate_volume_delaunay_button.clicked.connect(self
           .calculate_volume_delaunay)
        self.button_layout.addWidget(self.
           calculate_volume_delaunay_button)
```

```
        self.save_visualization_button = QPushButton("Save␣
           Visualization")
        self.save_visualization_button.clicked.connect(self.
           save_visualization)
        self.button_layout.addWidget(self.
           save_visualization_button)

        self.export_results_button = QPushButton("Export␣Results")
        self.export_results_button.clicked.connect(self.
           export_results)
        self.button_layout.addWidget(self.export_results_button)

        self.left_layout.addLayout(self.button_layout)

        # Add the control widget to the main layout
        self.main_layout.addWidget(self.control_widget)

        # Placeholder for point cloud visualization widget
        self.point_cloud_widget = QWidget()
        self.point_cloud_layout = QHBoxLayout()
        self.point_cloud_widget.setLayout(self.point_cloud_layout)
        self.main_layout.addWidget(self.point_cloud_widget)

        # Placeholder for footer widget
        self.footer_widget = QWidget()
        self.footer_layout = QHBoxLayout()
        self.footer_widget.setStyleSheet("background-color:␣#8699
           a1")
        self.footer_layout.addWidget(QLabel("Created␣by␣Jorge␣
           Silva␣DEM␣Universidade␣de␣Aveiro␣2024"))
        self.footer_widget.setLayout(self.footer_layout)
        self.main_layout.addWidget(self.footer_widget)

        # Adjusting the stretch factor to ensure proper resizing
        self.main_layout.setStretchFactor(self.control_widget, 0)
        self.main_layout.setStretchFactor(self.point_cloud_widget,
            1)
        self.main_layout.setStretchFactor(self.footer_widget, 0)

    def create_slider(self, label, min_val, max_val, default_val):
        slider_layout = QHBoxLayout()
        label_widget = QLabel(label)
        slider = QSlider()
        slider.setOrientation(1)  # Vertical orientation
        slider.setMinimum(min_val * 1000)  # Adjusted for
           increments of 0.1
        slider.setMaximum(max_val * 1000)  # Adjusted for
           increments of 0.1
        slider.setValue(default_val * 1000)  # Adjusted for
           increments of 0.1
        slider.setSingleStep(1)  # Set single step to 1
        slider.setTickPosition(QSlider.TicksBelow)
        slider.setTickInterval((max_val - min_val) // 1000)
        slider_label = QLabel(str(default_val))  # Display the
            initial value
        slider.valueChanged.connect(lambda val, label=slider_label
            : label.setText(str(val / 1000)))
        slider_layout.addWidget(label_widget)
        slider_layout.addWidget(slider)
        slider_layout.addWidget(slider_label)
        self.create_points_layout.addLayout(slider_layout)
        return slider



    def choose_point_cloud_file(self):
        file_dialog = QFileDialog()
        file_dialog.setNameFilter("Point␣Cloud␣Files␣(*.pcd)")
        if file_dialog.exec_():
            self.point_cloud_file_path = file_dialog.selectedFiles
                ()[0]
            self.point_cloud_file_label.setText(self.
                point_cloud_file_path)

    def visualize_point_cloud(self):
        if not self.point_cloud_file_path:
            QMessageBox.warning(self, "Warning", "Please␣choose␣a␣
                point␣cloud␣file.")
            return None

        # Clear existing plot, if any
        for i in reversed(range(self.point_cloud_layout.count())):
            widget = self.point_cloud_layout.itemAt(i).widget()
            if widget is not None:
                widget.deleteLater()

        pcd = o3d.io.read_point_cloud(self.point_cloud_file_path)
        print("Point␣cloud␣loaded:", pcd)

        if pcd.is_empty():
            QMessageBox.warning(self, "Warning", "The␣point␣cloud␣
                file␣is␣empty␣or␣could␣not␣be␣loaded.")
            return None

        number_points = self.number_points_slider.value()
        p1_load = np.asarray(pcd.points)
        rand_xy = np.random.uniform(self.xy_min_slider.value() /
            1000, self.xy_max_slider.value() / 1000,
                                    (number_points, 2))
        rand_z = np.full((1, number_points), self.height_slider.
            value() / 1000)
        p1 = np.concatenate((rand_xy, rand_z.T), axis=1)
        pointSet = o3d.geometry.PointCloud()
        pointSet.points = o3d.utility.Vector3dVector(p1)
```

```
p3_load = np.concatenate((p1_load, p1), axis=0)
pcd1 = o3d.geometry.PointCloud()
pcd1.points = o3d.utility.Vector3dVector(p3_load)

# Find the minimum Z value
min_z = np.min(np.asarray(pcd1.points)[:, 2])

# Translate points so that the lowest point is at Z = 0
translation = np.array([0, 0, -min_z])
pcd1.translate(translation)

x_min, x_max = self.x_min_slider.value() / 1000, self.
   x_max_slider.value() / 1000
y_min, y_max = self.y_min_slider.value() / 1000, self.
   y_max_slider.value() / 1000
z_min, z_max = self.z_min_slider.value() / 1000, self.
   z_max_slider.value() / 1000

voxel_size = 0.05
pcd1_downsampled = pcd1.voxel_down_sample(voxel_size)

passthrough = o3d.geometry.PointCloud()
passthrough.points = o3d.utility.Vector3dVector([[x, y, z]
    for [x, y, z] in pcd1_downsampled.points if
                                                 x_min <=
                                                    x <=
                                                    x_max
                                                    and
                                                    y_min
                                                    <= y
                                                    <=
                                                    y_max
                                                    and
                                                    z_min
                                                    <= z
                                                    <=
                                                    z_max
                                                    ])

points = np.asarray(passthrough.points)
print("Filtered␣points:", points)

if points.size == 0:
    QMessageBox.warning(self, "Warning", "No␣points␣remain
       ␣after␣filtering.")
    return None

self.figure = plt.figure()
self.ax = self.figure.add_subplot(111, projection='3d')

self.ax.scatter(points[:, 0], points[:, 1], points[:, 2],
   s=1)
self.ax.set_xlabel('X')
```

```
        self.ax.set_ylabel('Y')
        self.ax.set_zlabel('Z')
        self.ax.set_title('Point␣Cloud␣Visualization')

        canvas = FigureCanvas(self.figure)
        self.point_cloud_layout.addWidget(canvas)
        self.latest_fig = self.figure

        return points

    def calculate_volume_delaunay(self):
        # Show a "Waiting" message
        waiting_message = QMessageBox(self)
        waiting_message.setWindowTitle("Processing")
        waiting_message.setText("Please␣wait␣while␣the␣Delaunay␣
           triangulation␣is␣being␣processed...")
        waiting_message.show()
        QApplication.processEvents()  # Ensure the message is
            displayed immediately

        for i in reversed(range(self.point_cloud_layout.count())):
            widget = self.point_cloud_layout.itemAt(i).widget()
            if widget is not None:
                widget.deleteLater()
        extract_xy = []
        points = self.visualize_point_cloud()
        for point in points:
            extract_xy.append([point[0], point[1]])
        index = Delaunay(np.array(extract_xy))

        # Update visualization to show Delaunay triangulation
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        def get_triangles_vertices(triangles, vertices):
            triangles_vertices = []
            for triangle in triangles:
                new_triangles_vertices = [vertices[triangle[0]],
                    vertices[triangle[1]], vertices[triangle[2]]]
                triangles_vertices.append(new_triangles_vertices)
            return np.array(triangles_vertices)

        def volume_triangular_prism(triangle):
            p1, p2, p3 = triangle
            x1, y1, z1 = p1
            x2, y2, z2 = p2
            x3, y3, z3 = p3
            area = (1 / 2) * (x1 * y2 - x2 * y1 + x2 * y3 - x3 *
                y2 + x3 * y1 - x1 * y3)
            height = (1 / 3) * (z1 + z2 + z3)
            volume = area * height
            return area, volume
        # Plot the triangles
        triangles_vertices = get_triangles_vertices(index.
           simplices, points)
        for triangle in triangles_vertices:
            x = [point[0] for point in triangle]
            y = [point[1] for point in triangle]
            z = [point[2] for point in triangle]
            ax.plot(x + [x[0]], y + [y[0]], z + [z[0]], color='red
               ',
                    linewidth=0.2)  # Connect the last point to
                       the first to close the triangle

        total_area = 0
        total_volume = 0
        for triangle in triangles_vertices:
            area, volume = volume_triangular_prism(triangle)
            total_area += area
            total_volume += volume

        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')

        canvas = FigureCanvas(fig)
        self.point_cloud_layout.addWidget(canvas)

        # Close the "Waiting" message
        waiting_message.close()

        QMessageBox.information(self, "Volume␣and␣Area␣Calculation
           ",
                                f"The␣volume␣is␣{round(
                                   total_volume,␣3)}␣m^3\n"
                                f"The␣surface␣area␣is␣{round(
                                   total_area,␣3)}␣m^2")

        return total_volume, total_area

    def save_visualization(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getSaveFileName(self, "Save␣
           File", "", "PNG␣(*.png);;JPEG␣(*.jpg␣*.jpeg)")
        if file_path:
            self.latest_fig.savefig(file_path)

    def export_results(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getSaveFileName(self, "Save␣
           File", "", "Text␣Files␣(*.txt)")
        if file_path:
            with open(file_path, 'w') as file:
                file.write(f"Volume:␣{round(self.calculated_volume
                   ,␣3)}␣m^3\n")
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = PointCloudVisualizer()
    window.show()
    sys.exit(app.exec_())
```