UNIVERSIDADE D
COIMBRA

Alexandre Dias Xavier

Perception System for Forest Cleaning with UGV

Master's Dissertation in MIEEC, supervised by Professor Doctor Aníbal T. de Almeida and Prof. Doctor A. Paulo Coimbra and presented to Faculty of Science and Technology of the University of Coimbra

October 2021

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Perception System for Forest Cleaning with UGV

Alexandre Dias Xavier

Coimbra, October 2021

Perception System for Forest Cleaning with UGV

Supervisor:

Prof. Doctor A. Paulo Coimbra

Co-Supervisor:

Professor Doctor Aníbal T. de Almeida

Jury:

Prof. Doctor Lino José Forte Marques

Prof. Doctor Rui Alexandre de Matos Araújo

Prof. Doctor A. Paulo Coimbra

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, October 2021

Acknowledgements

Despite all the knowledge we may have, nothing happens without having someone to support us and push us when we need it most. Thus, I would like to express my gratitude to the people who made this dissertation possible. First of all, I would like to thank my supervisors Prof. Dr. António Coimbra and Professor Dr. Aníbal T. de Almeida for giving me the opportunity to work on their project and for their continuous support throughout the development of the work. I also want to thank my project colleague Paulo Mendes for the support he gave me in the data acquisition. Last but not least, I would like to thank my family, my group of friends who always support us and my girlfriend Joana for her support and love.

Resumo

O constante desenvolvimento de sistemas robóticos autónomos tem aumentando o interesse em utilizar os robôs como alternativa ao ser humano no desempenho de tarefas repetitivas, árduas e perigosas.

Face a uma alta densidade florestal existente em Portugal, mas também noutros países da Europa e de outros continentes, a necessidade de diminuir a matéria inflamável existente na floresta tornou-se um dos grandes objetivos da prevenção de grandes incêndios florestais.

Os desenvolvimentos na área da robótica permitem aos robôs mapear os ambientes florestais de modo a obter informação útil que permita percecionar qual a matéria inflamável existente.

A necessidade de perceber qual a vegetação que o robô deve ou não cortar, torna-se uma tarefa muito importante para o desempenho do robô.

Esta dissertação está focada na perceção do ambiente que rodeia o robô, ou seja, perceber quais os objetos que rodeiam o robô, quais são obstáculos, qual a vegetação a cortar e não cortar.

São propostas soluções usando LiDAR ou usando uma câmara RGB. Em relação ao LiDAR as soluções implementadas têm como base a altura dos objetos, a reflexão do "laser" do LiDAR conforme a superfície do objeto e também o tamanho dos objetos. Enquanto usando a câmara RGB a solução passa pelo uso de índices de vegetação e segmentação.

As soluções foram validadas usando data 'sets' e fotografias de ambiente real. No final foi possível classificar os objetos como obstáculos, neste caso carros, paredes e troncos, mas também vegetação cortada através de um trator equipado com uma capinadeira e vegetação não cortada.

Palavras Chave: LiDAR, *Point Cloud* Objetos, *Point Cloud* Solo, *Point Cloud Clustering*,*Point Cloud* - Classificação de objectos, Índices de vegetação RGB e Segmentação de Imagens, Classificação de Vegetação.

Abstract

The constant development of autonomous robotic systems has open up the interest in using robots as an alternative to humans in the performance of repetitive, arduous, and dangerous tasks.

Given the high forest density in Portugal, but also in other countries in Europe and other continents, the need to reduce the inflammable matter in the forest has become one of the major goals in the prevention of large forest fires.

Developments in robotics allow robots to map forest environments in order to obtain useful information to understand the existing inflammable matter.

The need to understand which vegetation the robot should or should not cut becomes a very important task for the robot performance.

This dissertation is focused on the perception of the environment that surrounds the robot, that is, to understand which objects surround the robot, which are obstacles, which vegetation to cut and not to cut.

Solutions are proposed using LiDAR or using an RGB camera. Regarding LiDAR the solutions implemented are based on the height of the objects, the reflection of the LiDAR laser according to the surface of the object and also the size of the objects. While using the RGB camera the solution goes through the use of vegetation indexs and segmentation.

The solutions were validated using data sets and real environment photographs. In the end it was possible to classify the objects as obstacles, in this case cars, walls and trunks, but also vegetation cut by a tractor equipped with a clearing machine and uncut vegetation.

Keywords: LiDAR, Point Cloud of Objects, Point Cloud of Ground, Point Cloud Clustering, Point Cloud - Objects Classification, RGB Vegetation Indexes and Image Segmentation, Vegetation Classification.

"Life is like riding a bicycle. In order to keep your balance, you must keep moving."

— Albert Einstein

Contents

Acknowledgements ii

Resumo iii

Abstract iv

List of Acronyms x

List of Figures xii

List of Tables xvi

1 Introduction 1

2 Background and State of the Art 5

2.1 Point Cloud Library 5

2.1.1 Point Cloud 5

2.1.2 Filtering 7

2.1.3 Point Cloud Registration 8

2.1.4 Identifying Ground in Airborne LiDAR Measurements 9

2.2 Mapping 14

2.2.1 Simultaneous localization and Mapping (SLAM) 16

2.2.2 Octomap 18

2.2.3 Velodyne Height Map 20

2.3 Camera Calibration 22

2.4 Remote Sensing vegetation 26

2.5 Factorization-Based Texture Segmentation 28

2.5.1 Factorization based Image Model 29

2.5.2 Factorization based Segmentation 31

2.6 Software and Hardware . . . 33
2.6.1 Robot Operating System (ROS): Robot Operating System . . . 34
2.6.2 Software available in ROS . . . 34
2.6.3 Ranger . . . 36
2.6.4 LiDAR . . . 38
2.6.5 RGB camera . . . 38

3 Developed Methodology **39**
3.1 LiDAR . . . 39
3.1.1 Separation of Objects from the Ground . . . 39
3.1.2 Height map . . . 45
3.1.3 Octomap and Z-coordinate cut . . . 47
3.1.4 LiDAR Intensity . . . 49
3.1.5 Classifying Objects by Size . . . 56
3.2 RGB camera . . . 58
3.2.1 Vegetation indexes . . . 59
3.2.2 Segmentation . . . 62
3.2.3 Fusion of RGB vegetation indexes with factorization-based texture segmentation . . . 63

4 Presentation and Discussion of Other Results **65**
4.1 Results . . . 65
4.2 Discussion . . . 71
4.2.1 Applying the Progressive Morphological Filter . . . 71
4.2.2 Point Cloud - Z Coordinate Cut . . . 71
4.2.3 LiDAR Intensity . . . 72
4.2.4 Object Classification by Diagonal Size . . . 73
4.2.5 Segmentation using RGB Vegetation Indexes . . . 73

5 Conclusion **75**
5.1 Future Work . . . 76
5.1.1 Automatic cutting at the Point Cloud . . . 76
5.1.2 LiDAR-Camera Fusion . . . 76

6 Bibliography **78**

List of Acronyms

A-LOAM	Advanced LOAM
ALS	Alternating Least Squares
ARVI	Atmospherically Resistant Vegetation Index
DoF	Degrees of freedom
DTM	Digital Terrain Model
EKF	Extended Kalman Filter
FoV	Field Of View
GPS	Global Positioning System
ICP	Iterative Corresponding Point
IMU	Inertial Measurement Unit
KF	Kalman Filter
LOAM	LiDAR Odometry and Mappin
LeGO-LOAM	Lightweight and Ground-Optimized LiDAR Odometry and Mapping
LiDAR	Light Detection And Ranging
LoG	Laplacian of Gaussian
NDT	Normal Distribution Transform
NDVI	Normalised Difference Vegetation Index
NIR	Near Infrared

NNLS	Non-Negative Least Squares
PCL	Point Cloud Library
RANSAC	RANdom SAmple Consensus
RGB	Red, Green, and Blue
ROS	Robot Operating System
SLAM	Simultaneous localization and Mapping
SVD	Singular Value Decomposition
TGI	Triangular Green Index
TIN	Triangulated Irregular Network
UAV	Unmanned aerial vehicle
UC	University of Coimbra
UGV	Unmanned Ground Vehicle
VARI	Visible Atmospheric Resistance Index
VI	Vegetation Index

List of Figures

1.1 The role of the perception module in the control scheme of mobile robots. [1] 2

2.1 Steps of the progressive morphological filter. 13
2.2 Comparison between the two most common types of maps. 14
2.3 General scheme for simultaneous localisation and map construction [1]. . . . 16
2.4 Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right. 19
2.5 Ray-casting from sensor origin to end point, the last voxel is marked as occupied, all the other voxel along the ray are marked as free. 19
2.6 Example of a 3D map of an urban street scene created with Octomap. 20
2.7 LiDAR scanning and potential obstacles using Velodyne height map. 22
2.8 Camera calibration applications.[2] 22
2.9 Extrinsic and intrinsic camera parameters [2]. 23
2.10 Extrinsic and intrinsic camera parameters, relationship between coordinates [2]. 23
2.11 Extrinsic parameters of a camera [2]. 23
2.12 Pixel pitch.[2] 24
2.13 Types of radial distortion. 25
2.14 Tangential distortion. [2] 26
2.15 (left) A textured image. (right) Segmentation result using least squares estimation [3]. 30
2.16 Illustration of the smoothing effect. (left) Synthetic image containing two regions with different Gaussian noises. (middle) Segmentation result using the method porposed in [3]. (right) Segmentation result with a very large integration scale 33
2.17 3D reconstruction of a fenced agricultural field using RTAB-Map. 35

2.18 Nodelets of hdl_slam. 36
2.19 Ranger, robot used in the project.[4] 37
2.20 Hardware distribution in Ranger.[4] 37

3.1 Original Point cloud represented by Intensity (Point cloud colour). 40
3.2 Scene corresponding to the point cloud of the figure 3.1. 40
3.3 Point cloud corresponding to the ground after the application of the morphological filter, using the default parameters. 41
3.4 Point cloud corresponding to the objects after the application of the morphological filter, using the default parameters. 41
3.5 Point cloud corresponding to the ground after the application of the morphological filter, using updated trial and error parameters. 42
3.6 Point cloud corresponding to the objects after the application of the morphological filter, using updated trial and error parameters. 42
3.7 Flowchart of the algorithm described above. Slope Calculation. 44
3.8 Vertical Field Of View (FoV) of the LiDAR. 44
3.9 Original point cloud, input for the height map. 45
3.10 Partial photo of the scene corresponding to the point cloud in figure 3.9. . . 46
3.11 Overlapping point clouds, red for obstacles and green for the non-obstacles. . 46
3.12 Point cloud containing the objects/ground represented (colour) by their height. 47
3.13 Original point cloud. 48
3.14 OctoMap representation of the point cloud containing the ground points resulting from the application of the progressive morphological filter to the original cloud. 48
3.15 OctoMap representation of the point cloud containing the objects points resulting from the application of the progressive morphological filter to the original point cloud. 48
3.16 Octomap representation of the z-cut made between 0.5 m and 0.8 m above the LiDAR origin in the point cloud. 49
3.17 Grid map resulting from the z-cut in the point cloud. 49
3.18 Point cloud represented by intensity. 50
3.19 Photos of the scene of the above point cloud. 50
3.20 Point cloud split by intensity - in intervals of 5 in steps of 5. 51
3.21 Point cloud of intensity [35-40] slited represented in steps of 1. 51

3.22 Representation of the point cloud corresponding to the interval [37-39]. The colour of the points is according to the cluster they belong to. 52
3.23 Representation of the point cloud corresponding to the tree trunk clusters after the clusters below the 500 points threshold are removed. 53
3.24 Representation of the point cloud corresponding to z-cut between 0.3 and 0.35 metres and the circle corresponding to the diameter of each trunk. 53
3.25 Point cloud whit intensity between [0 36] - intensity below that used previously. 54
3.26 Point cloud corresponding to two parts of two cars, i.e. point cloud formed by the two clusters with 500 or more points. 54
3.27 Point cloud corresponding to two parts of two cars (metal), i.e. obstacles that are not vegetation and the corresponding circles. 55
3.28 Point cloud with the main obstacles classified (red circles for non-vegetation obstacles and yellow circles for trunks). 55
3.29 Obstacles represented in octomap and their classification (red circles for non-vegetation obstacles and yellow circles for trunks). Also represented the original point cloud (green points) for a better perception of the obstacles. . . . 56
3.30 Point cloud resulting from the application of the ROS package **"hdl_graph_slam"**, and the trajectory of the LiDAR (small red balls). 57
3.31 On the left, original point cloud (in blue) and classified objects. The classification is for trunks or poles (green circles) and larger objects like a car or a wall (red polygons). On the right, z-cut between [0.8;1.1] of the point cloud and classified objects. 58
3.32 Z-cut between [0.8;1.1] of the point cloud and classified objects. The classification is trunks or poles (green circles) and objects like a car or a wall (red polygons). 58
3.33 Comparison between different RGB vegetation indexes (scene 1). 60
3.34 Comparison between different RGB vegetation indexes (scene 2). 61
3.35 RGB image used as input to the segmentation algorithm. 63
3.36 Comparison of different values of **ws** parameter used in the factorization-based texture segmentation algorithm. 63
3.37 Various stages of the fusion of the RGB indexes with the segmentation algorithm. 64

4.1 Photos of the new scene where the new example data was recorded. 65

4.2 Results of the application of the progressive morphological filter with parameter **slope**=0.5. 66
4.3 Framework of the methodology developed using LiDAR. 67
4.4 Results of the application of the Height Map. 67
4.5 Original point cloud and the yellow circle corresponding to the diameter of the trunk. 68
4.6 Results of the application of the Classification of obstacles by diagonal size. . 69
4.7 Framework of the methodology developed using RGB Camera. 70
4.8 Results of the application of the fusion of RGB vegetation indexes with factorization-based texture segmentation. 70

List of Tables

2.1 Comparison of different map types. 15
2.2 Comparison of some common SLAM techniques [5]. 18

3.1 Comparison of results of the ground point cloud by changing the **slope** a parameter. 43
3.2 RGB vegetation indexes 59

4.1 Comparison of the real value with the calculated value of the diameter by the LiDAR Intensity method. 73

1 Introduction

The constant depopulation of the Portuguese interior over the years has left a large area of forest and agricultural land abandoned. This, together with the growing problem of global warming, has made the abandonment of forests and agricultural land a huge problem not only in Portugal, but also in countries such as the United States of America and Australia. The average number of residents in central Portugal is 78.6 inhabitants per square kilometre, while in the Lisbon Metropolitan Area it is 899.6 inhabitants per square kilometre, illustrating well the shortage of population in the interior. In 2017, around half a million hectares were burnt in Portugal, representing more than 50% of the total area burnt that year in southern European countries [6].

The cleaning of vegetation near populations with the creation of fuel management bands and clearing in forest corridors, particularly along high voltage power lines, is of critical importance to minimise the risks of large forest fires. With an area of over three million hectares of forest in Portugal, it makes the forest the main occupation of the Portuguese territory, making fire prevention a crucial task.

Given the large extension of the forest and the human resources available in the most affected areas, adequate forest cleaning is a very difficult task without the use of large machines or forest cleaning robots. Thus, the creation of specialised robots that move around in a forest environment, cleaning the forest, mapping their surroundings, and locating vegetation to be cut down to reduce flammable fuel that could cause a large fire, are one area of focus to prevent and devastating fires.

The robots can be the beginning of a solution because they can tackle dangerous and repetitive problems in a more efficient way, and at the same time safeguard human lives. Allowing better monitoring of the forest environment by mapping lesser-known locations, can be crucial in helping firefighting organisations to fight fires more directly and effectively. With this, many professional and civilian lives can be saved, and the forest can be cared for and preserved. Following this thought, accurate mapping of the environment with the

help of a robotic system has been a much-investigated area and several methods have been developed [7] [8]. However, the necessity of not only mapping but also understanding what surrounds the robot is very important for a safe navigation.

For outdoor field robotics, there are many challenges that the indoors environments do not have. Some of the challenges include, for example, uncertain and non-linear paths, non-ideal environmental conditions, lack of geometric features and deep information, presence of people, animals or even plant species that need to be preserved, i.e., not cut [58]. Forest mapping presents most of the problems reported above giving even more importance to a coherent set of data that can be easily interpreted by the robot.

Before creating the global map, it is necessary to have the perception of the environment (figure 1.1), in other words, it is necessary to capture the environment through the sensors that make up the console dedicated to perception. It is in this block called perception that the information is extracted and interpreted.

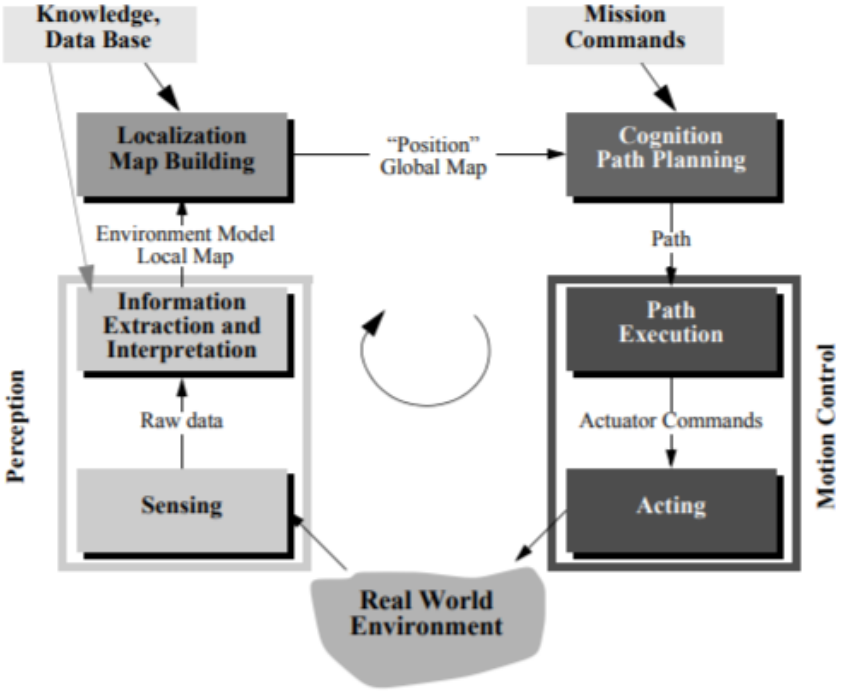


Figure 1.1: The role of the perception module in the control scheme of mobile robots. [1]

Light Detection And Ranging (LiDAR) is one of the most popular technologies on the perception block. The main factors are its usefulness and versatility. The highly accurate point clouds captured by LiDAR, that are not easily affected by adverse conditions such as rain or light, due to the laser wavelength used, provide a complement to technologies such as 3D reconstruction, autonomous driving, 3D mapping, calibration, and registration. Thus LiDAR is an essential device in any robotic system that wants to autonomously map

complex unstructured environments such as the forest.

The advantages of LiDARs result from the provision of very detailed 3D point clouds, which can be manipulated and processed, providing also useful data, namely distances, intensity of light reflections and the geometry of the environment that often allows the distinction of object types.

Another sensor widely used in the perception block are cameras, allowing through colour or features to distinguish between patterns and or obstacles.

This dissertation is focused on the perception module for a forest cleaning robot i.e. perceiving the environment surrounding the robot. Through the use of sensors obtain, process and understand the data obtained and have the perception of the type of vegetation present, the state in which it is (green, dry, cut, not cut, etc.) and perceive the obstacles that are in the environment.

This dissertation is part of the SafeForest project whose partners are the company Ingeniarius, the Institute for Systems and Robotics (University of Coimbra), ADAI (Association for the Development of Industrial Aerodynamics), the company SILVAPOR and Carnegie Mellon University.

With the aim of developing field robotics, the SafeForest project aims to provide an autonomous robot with the ability to operate in the forest to clear flammable vegetation. Among others, the robot should be able to map and navigate within the forest environment where it operates and detect flammable material (i.e. vegetation to be cut). SafeForest implies cooperative perception between a Unmanned Ground Vehicle (UGV) and a team of nmanned aerial vehicles (UAV's), in which each member contributes to the overall knowledge of the system by cooperatively sharing and processing data and perceiving each other, combining the sensory capabilities, perspectives and processing power of multiple agents to achieve better results. In this dissertation the sensors to be used are LiDAR and RGB camera.

This Dissertation is organised into five main chapters. Chapter 2 covers the basic and fundamental knowledge of Point Clouds, Filtering, Mapping, Segmentation and ROS, essential for the understanding of the whole document. A summary of the most popular and used approaches for mapping, segmentation as for point clouds registration is also made. In Chapter 3 the methods and techniques used for perception of the environment surrounding the robot using LiDAR and RGB cameras are presented. Then, in Chapter 4 the results obtained using the developed methods are presented and discussed making a critical appreciation of what should or should not be improved. Finally, in Chapter 5, a conclusion is

made presenting on the advantages and disadvantages of our methods, as well as possible future work.

2 Background and State of the Art

2.1 Point Cloud Library

The Point Cloud Library (PCL) is a large scale, open project [1] for point cloud processing.

PCL is cross-platform, being able to compile successfully on Linux, MacOS, Windows, and Android. The programming language used is c++.

The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. For example, these algorithms are used to filter outliers from noisy data, stitch together 3D point clouds, segment relevant parts of a scene, extract key points and calculate descriptors to recognise objects in the world based on their geometric appearance and create surfaces from point clouds and visualise them. In the following, several algorithms important to the Dissertation are described.

2.1.1 Point Cloud

A point cloud is a data structure used to represent a set of multidimensional points and is commonly used to represent three-dimensional data. In a 3D point cloud the points generally represent the X, Y and Z geometric coordinates of an underlying sampled surface. When colour information is present, the point cloud becomes 4D.

A point cloud can be captured using sensors such as RGB-D cameras, stereo cameras, 3D laser scanners, time-of-flight cameras. It is also possible to obtain point cloud by using simulation software.

Basic Structures

The basic data type in PCL is a PointCloud. A PointCloud is a c++ class with the following data fields:

- **width (int)** - specifies the width of the point cloud dataset in number of points;
  - the total number of points in the cloud (equal to the number of elements in points) for unorganised data-sets;
  - the width (total number of points in a row) of an organised point cloud data-set;
- **height (int)** - specifies the height of the point cloud data-set in number of points;
  - set to 1 for unorganised point clouds;
  - the height (total number of rows) of an organised point cloud data-set;
- **points (std::vector <PointT>)** - vector that contains the data array where all the points are stored;
- **isdense (bool)** - specifies if all the data in points is finite (true), or whether the XYZ values of certain points might contain Inf/NaN values (false);
- **sensororigin(Eigen::Vector4f)** - specifies the sensor acquisition (origin/translation). This member is usually optional, and not used by the majority of the algorithms in PCL;
- **sensororientation(Eigen::Quaternionf)** - specifies the sensor acquisition pose (orientation). The quaternion is zero if not used. This member is usually optional, and not used by the majority of the algorithms in PCL;

Point types

- **PointXYZ** - float x, y, z;
- **PointXYZI** - float x, y, z, intensity;
- **PointXYZRGB** - float x, y, z, RGB;
- **PointXYZRGBA** - float x, y, z, uint32t rgba;
- **Normal** - float normal[3], curvature;
- **PointNormal** - float x, y, z, normal[3], curvature;
- **Histogram** - float histogram[N];
- it is possible to define new types to meet requirements.

2.1.2 Filtering

The raw point cloud is often noisy and contains outliers. Thus, it is crucial to remove noise and outliers from the point cloud while preserving its features, in particular, its fine details.

When working with 3D data, there are many reasons to filter data such as, restricting range (PassThrough), downsampling (VoxelGrid), outlier removal (Statistical Outlier Removal / Radius Outlier Removal) or selecting indices.

PassThrough Filter

PassThrough uses the base Filter class methods to pass through all data that satisfies the user given constraints. Once the user defines the constraint, this filter will remove all points that do not satisfy the constraint. For example, reduce the point cloud to a range of z between 0.2 and 0.3 meters.

Euclidean Cluster Extraction

A data clustering method needs to divide an unorganised point cloud model P into smaller parts so that the total processing time of P is significantly reduced. A simple data clustering approach in a Euclidean sense can be implemented by making use of a 3D grid subdivision of space using fixed-width bins, or more generally, an octree data structure. This particular representation is very fast to construct and is useful for situations where a volumetric representation of the occupied space is required, or the data in each resulting 3D box can be approximated with a different structure.

To find and segment the individual point clusters of objects in the plane, the system first needs to understand what a point cluster of objects is and what differentiates it from another point cluster.

Mathematically, a cluster is defined as follows. Given two distinct point clusters $O_i = \{p_i \in P\}$ and $O_j = \{p_j \in P\}$ if:

$$min \left\| p_i - p_j \right\|_2 \geq d_{th} \tag{2.1}$$

where d_{th} is a maximum imposed distance threshold.The above equation states that if the minimum distance between a set of points $p_i \in P$ and another set $p_j \in P$ is larger than a given distance value, then the points in p_i are set to belong to a point cluster O_i and the ones in p_j to another distinct point cluster O_j. From an implementation point of view, it is

therefore important to have a notion on how this minimal distance between the two sets can be estimated.

Assuming that we use a Kd-tree structure for finding the nearest neighbors, the algorithmic steps for that would be [9]:

1. create a Kd-tree representation for the input point cloud dataset P;
2. set up an empty list of clusters C, and a queue of the points that need to be checked Q;
3. then for every point $\boldsymbol{p}_i \in P$, perform the following steps:
   - add $\boldsymbol{p}_i$ to the current queue Q;
   - for every point $\boldsymbol{p}_i \in Q$ do:
     - search for the set P_i^k of point neighbors of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$;
     - for every neighbor $\boldsymbol{p}_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q;
   - when the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list;
4. the algorithm terminates when all points $\boldsymbol{p}_i \in P$ have been processed and are now part of the list of point clusters C

2.1.3 Point Cloud Registration

Point cloud registration is a fundamental problem in 3D computer vision. Given several sets of points in different coordinate systems, the goal of registration is to find the transformation that best aligns all of them in a common coordinate system. According to the purpose, current approaches can be divided into coarse registration and fine registration. Coarse registration is mainly used to compute a rough estimate of the hard transformation between two point clouds. The goal of fine registration is to make the transformation between two clouds as accurate as possible [10].

The Iterative Corresponding Point (ICP) is the most common method. This method minimizes the predefined error function by iteratively optimizing processes and at each iteration of the method it can be divided into two steps, the search for the match and the calculation of the transformation [10].

The ICP algorithm is of high accuracy and stability but low efficiency. It requires an inclusion relation of the sets to ensure that each point in the target set has a corresponding point in the reference set. This method looks for matching by geometric distances between pairs of points so that it is easier to fall into the local optimum [10].

The Normal Distribution Transform (NDT) algorithm registers point clouds using statistical information from the data instead of using the points. For this method, the probability distribution is calculated by the normal distribution of the points and then the registration results are optimised using standard optimisation techniques. The NDT algorithm is more efficient than ICP [10].

2.1.4 Identifying Ground in Airborne LiDAR Measurements

Identifying points that belong to the ground in LiDAR datasets is a challenging task.

Kraus and Pfeifer [11] in aerial remote sensing used linear least squares iterative interpolation to removing tree measurements and generating a Digital Terrain Model (DTM) in forested areas. This method was extended later to filter buildings and trees in urban areas [12]. The iterative linear interpolation method extracts a low-degree polynomial trend surface from the original elevation data to produce a set of small elevation values.

Vosselman [13] proposed a slope-based filter that identifies ground data by a comparison of slopes among a LiDAR point and its neighbours. A point is classified as a ground measurement if the maximum value of slopes between this point and any other point within a given circle is lower than a predefined threshold. The lower the slope of the threshold, the more objects will be removed. The threshold slope for a given area is either a constant or a function of distance. A sensible threshold slope can be obtained using prior knowledge of the terrain in the study area.

Morphology mathematics composes operations based on set theory to extract features from an image. Haugerud and Harding [14] developed an algorithm to filter tree points in forested areas by a comparison of local curvatures of point measurements. Ground measurements were selected by removing tree vertices iteratively from an Triangulated Irregular Network (TIN) constructed from LiDAR measurements. Alternatively, soil points can be classified by iteratively selecting soil measurements from an original dataset.

Other commonly utilised algorithm to remove non-ground objects is a mathematical morphology filter that is applied to a greyscale image [13], [15], [16]. The elevation of trees, cars and buildings is generally higher than that of the surrounding ground points. If the

LiDAR points are converted into a regular grayscale grid image in terms of elevation, then the shapes of buildings, cars and trees can be identified by the change in grey tone. It is well known that compositions of algebraic set of operations based on mathematical morphology can be used to identify objects in a greyscale image. Therefore, mathematical morphology can be used to filter LiDAR data [17].

The main objective of [17] is to develop a progressive morphological filter to enable the automatic extraction of ground points from LiDAR measurements with minimal human interaction.

Morphological Filters

Morphology mathematics composes operations based on set theory to extract features from an image. Two key operations, dilation and erosion, are typically used for increasing (dilating) or reducing (eroding) the size of features in binary images. The dilation and erosion operations can be combined to produce open and close operations [18]. The concept of erosion and dilation has been extended to multilevel (greyscale) images and corresponds to finding the minimum or maximum of the combinations of pixel values and the kernel function, respectively, within a given neighbourhood of each raster.

These concepts can be also extended to the analysis of a continuous surface, such as a digital surface model, as measured by LiDAR data. For a LiDAR measurement, the dilation of the elevation z at x and y is defined as [17]:

$$d_p = \max_{(x_p,y_p)\in w} (z_p) \tag{2.2}$$

where points (x_p, y_p, z_p) represent p's neighbors (coordinates) within a window, w. The window can be a one-dimensional (1-D) line or two-dimensional (2-D) rectangle or other shapes. The dilation output is the maximum elevation value in the neighborhood of p. Erosion is a counterpart of dilation and is defined as:

$$e_p = \min_{(x_p,y_p)\in w} (z_p) \tag{2.3}$$

´

The combining of erosion and dilation results in opening and closing operations that are used for filtering the LiDAR data. The opening operation is obtained by performing an erosion of the dataset followed by a dilation, while the closing operation is performed by first

performing a dilation and then an erosion. The ability of an opening operation to preserve features larger than the window size is very useful in some applications.

Kilian et al. [16] proposed a method to remove non-ground points using a morphological filter. In their method, a point with the lowest elevation within a given window size is first detected after an opening operation is performed on the dataset. Then, the points in this window that lie within a band above the lowest elevation are selected as ground points. The band range is determined by the accuracy of the LiDAR survey, which is typically 20-30 cm. All ground points are identified by moving the filter window over the whole dataset.

Lohmann et al. [15] used a dual-rank morphological filter proposed by Eckstein and Munkelt [19] to classify Airborne LiDAR point data. The dual-rank filter initially sorts the neighborhood of a point p in terms of elevation, and then selects an elevation with a given rank value i to perform a rank operation $R(p, i)$, where i ranges from 1 to n_p, and n_p is the total number of points of p's neighbors (including p). The neighbors of a point are delineated by a window that is usually a circle and can be any shape. The dual-rank filter is then defined as:

$$DR(p, i) = R(p, i) \circ R(p, n_p - i + 1). \tag{2.4}$$

The symbol $''\circ''$ indicates the succeeding operations: the points are processed by the first-level operation, and then the second-level operation is performed on the results of the first operation. The dual-rank filter becomes an opening operation when the rank value is one (i.e., $i = 1$) and a closing operation when the rank value is n_p ($i = n_p$).

The above-mentioned morphological filters need to be improved because they suffer from several problems, such as the requirement of a predefined filtering window size. In addition, a highly automatic filtering tool that identifies the ground measurements is desired due to the large volume of LiDAR data involved [17].

Progressive Morphological Filter

As seen above, morphological filters can remove measurements for buildings and trees from LiDAR data, but it is difficult to detect all non-ground objects of various sizes using a fixed filter window size. This problem can be solved by gradually increasing the window size of the morphological filters as described below [17].

An initial filtered surface is derived by applying an opening operation with a window of length l_1 to the raw data. The large non-ground features such as buildings are preserved

because their sizes are larger than l_1, while individual trees of size smaller than l_1 are removed. For the terrain, features smaller than l_1 are cut off and replaced by the minimum elevation within l_1.

In the next iteration, the window size is increased to l_2, and another opening operation is applied to the filtered surface, resulting in a further smoothed surface. The building measurements are removed and replaced by the minimum elevation of previous filtered surface within l_2, since the size of the building is smaller than the current window size.

When performing an aperture operation for laser-scanned data with a line window that increases gradually in size, the progressive morphological filter is capable of removing buildings and trees in various sizes from a LiDAR dataset. However, the filtering process has a tendency to produce a surface that is below the terrain measurements, leading to incorrect removal of measurements at the top of the high relief terrain. Even in areas of flat terrain, the filtered surface is usually lower than the original measurements. Therefore, most of the point measurements for the terrain are removed, and only a filtered surface is available if the aperture operation is performed directly for the LiDAR data. This problem can be overcome by introducing an elevation difference threshold based on the elevation variations of the terrain, buildings, and trees.

The buildings have a certain size and height. There is abrupt change in elevation between the roof and the base of a building, while the elevation changes of the terrain are gradual. The difference in elevation changes of buildings and terrain can help the filter to separate the building and terrain measurements.

Suppose that $dh_{p,1}$ represents the height difference between an original LiDAR measurement and the filtered surface in the initial iteration at any given point p, and $dh_{T,1}$ represents the elevation difference threshold. Point p is classified as a ground measurement if $dh_{p,1} \leq dh_{T,1}$ and as a non-ground measurement if $dh_{p,1} > dh_{T,1}$. Let $dh_{max(t),1}$ stand for the maximum height difference between the original terrain measurements and the filtered surface. If a $dh_{T,1}$ is selected such that the $dh_{max(t),1}$ is less than $dh_{T,1}$, then the LiDAR measurements for terrain will be preserved. In general, $dh_{T,1}$ will be a function of window size.

In the second iteration, suppose that the maximum height difference between the previous and this filtered terrain surface is $dh_{max(t),2}$. The ground measurements within $dh_{max(t),2}$ will be preserved as long as $dh_{max(t),2}$ is smaller than the elevation difference threshold $dh_{T,2}$ for the current operation. Suppose that the minimum elevation difference for the building between the previous and current filtering operation is represented by $dh_{min(b),2}$, which is

approximately the height of the building. The building measurements will be removed on the condition that $dh_{min(b),2}$ is larger than $dh_{T,2}$.

Generally, the elevation difference threshold $dh_{T,k}$ is set to be the minimum height value of the building objects in an analyzed area at iteration k. Taking $dh_{T,2}$ as the threshold, for any given point p at kth opening operation, we mark p as a ground measurement if $dh_{p,k} \leq dh_{T,k}$, and as a non-ground measurement otherwise. In this way, the measurements for buildings with various sizes can be identified by gradually increasing the widow sizes and applying an opening operation repeatedly until a window size is greater than the size of the largest building. Since there is also an abrupt elevation change from trees to adjacent ground, the above building filtering procedure can be applied to the removal of tree measurements as well. Note that the filtered surfaces from the opening operation are not utilized to generate the DTM, but used to help classify point measurements together with elevation difference thresholds.

The step-by-step for utilizing the progressive morphological filter to construct the DTMs are shown in figure 2.1 [17].

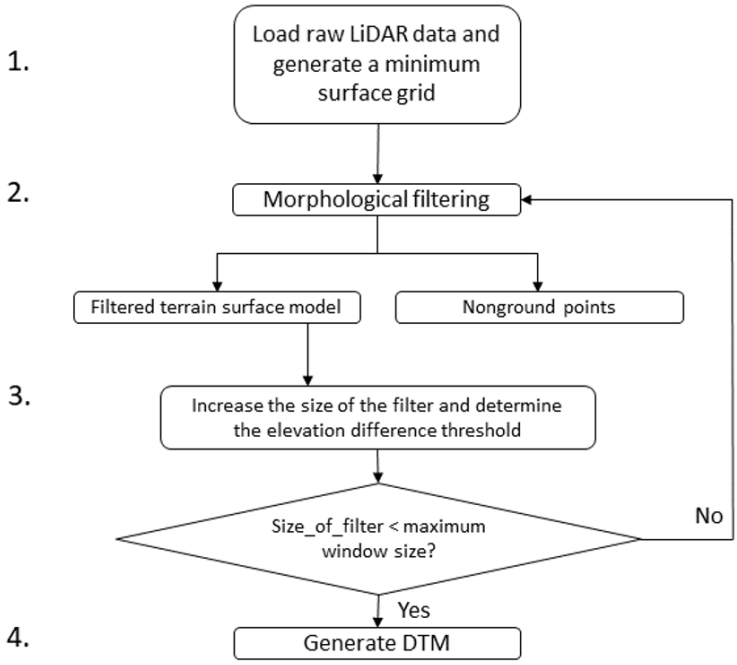


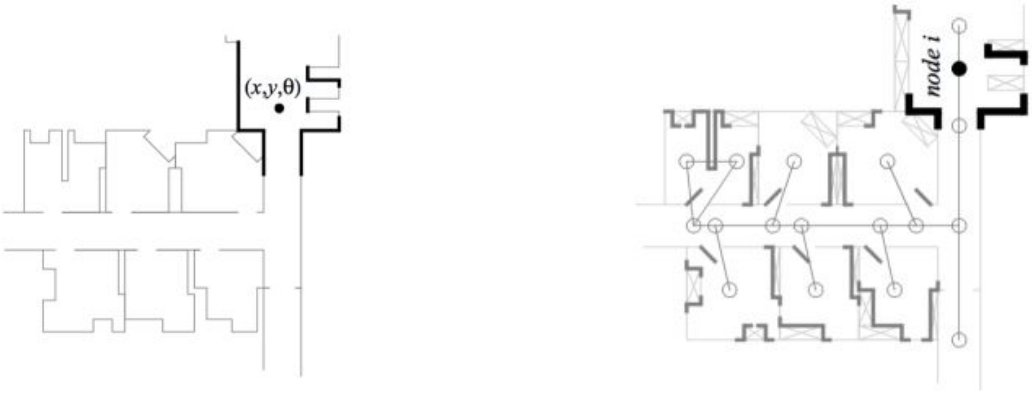
Figure 2.1: Steps of the progressive morphological filter.

Note that each minimum surface grid cell generated in Step 1 will contain an original or interpolated LiDAR point with elevation representing the cell value. Filtering operation performed to the grid is in fact applied to points in cells. The progressive morphological filter therefore classifies LiDAR measurements at the point level.

2.2 Mapping

Mapping consists of creating a representation of the environment from information obtained from sensors. There are two main types of maps, metric maps and topological maps (figure 2.2) [20].

Metric maps represent the environment using evenly spaced cells that form a grid and each cell represents a location in space. Occupancy grids are one of the most popular map types because they are easily constructed for large-scale locations and the geometry of the grid directly corresponds to the geometry of the environment, so that the position and orientation of the robot can be easily determined. The disadvantage of grid-based maps is that they take up a lot of memory space and have a high computational cost, as they need to process all the cells that make up the environment under analysis [21].



(a) Metric Map.

(b) Topological Map

Figure 2.2: Comparison between the two most common types of maps.

Topological maps represent the map by using a graph based on landmarks and their connectivity. In this method, the position of the robot is determined in relation to the model by means of landmarks. Thus, each node corresponds to a specific location and the existence of an edge connecting them indicates the possibility of navigation. However, topological maps have difficulty distinguishing between very identical locations. For example, if the robot passes two similar locations, it cannot quickly determine whether it is the same or a different location [22].

A comparison of pros and cons of the various types of maps is given in the table 2.1 [5].

Maps can be in 2D or 3D. 2D maps represent objects in two dimensions, length and

[1]Small-scale → a few users → uses a few resources.

[1]Large-scale → a large number of users → uses a large number of resources.

Table 2.1: Comparison of different map types.

Type of Map	Description, Pros and Cons
1. Occupancy Grid Maps	Map is defined as a grid, each cell of the grid holds a probability for occupancy. **Pros:** probabilistic, suitable for 2D mapping, suitable for dynamic environments. **Cons:** expensive on fine resolution, expensive for 3D mapping.
2. Feature Maps	Map is represented by features. **Pros:** efficient for localization, scales well[1]. **Cons:** needs feature extraction, data association of features is difficult.
3. Topological Maps	Map is represented by abstract spatial information. **Pros:** well suited for high-level planning. **Cons:** needs map processing, limited in waypoint following.
4. Semantic Maps	Map is represented by semantic information. **Pros:** well suited for high-level and goal-oriented reasoning. **Cons:** needs training, object recognition and classification.
5. Appearance Maps	Map is represented by a weighted graph and multiple views. **Pros:** very intuitive and useful for human and robot interaction. **Cons:** requires high storage capacity to record all views.
6. Hybrid Maps	Combination of different mapping methods. **Pros:** suitable for loop closure, can handle map inconsistency. **Cons:** needs map processing, requires coordination between maps.

width. 3D maps represent length, width and height, allowing the perception of relief.

In order for the robot to create a map of the environment based on the information acquired by the sensors, the robot must know where it is located . When a robot is placed in an unfamiliar environment, it starts without a map or any idea where it is. In this situation the robot must create a map while locating itself within that map. This is known as the simultaneous localization and mapping problem, or Simultaneous localization and Mapping (SLAM) (figure 2.3).

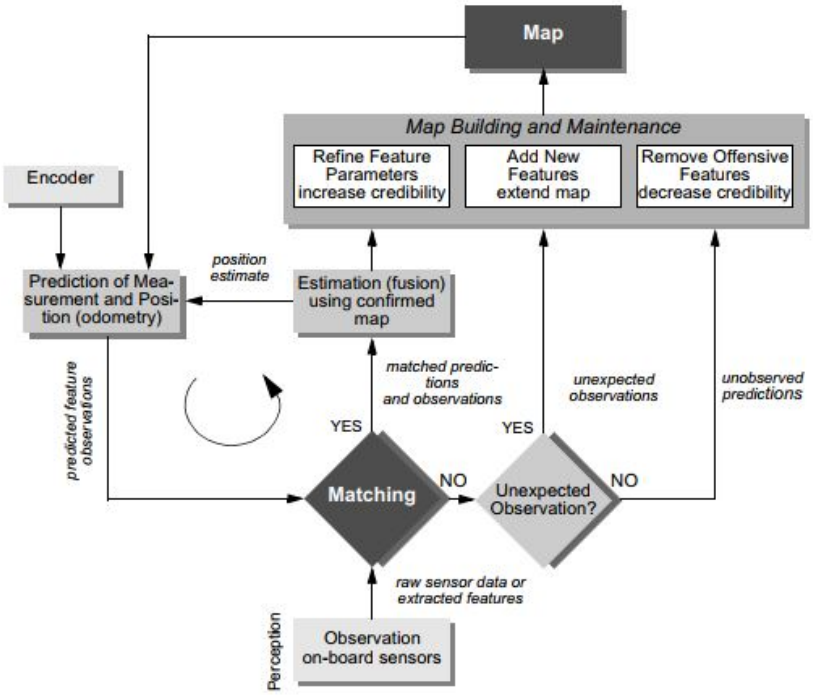


Figure 2.3: General scheme for simultaneous localisation and map construction [1].

2.2.1 SLAM

The SLAM problem is defined as a simultaneous localisation and mapping problem, in which a robot searches to acquire a map of the environment while simultaneously trying to locate itself in the same map it is creating [1].

Unlike mapping and localisation systems that require a priori knowledge of the environment or modification of the environment, SLAM systems can operate in situations where map information or the robot's position is not initially known. This capability makes these systems significantly more autonomous and ideal for a wide variety of applications.

SLAM is a method that aims to fuse different information such as odometry, measurements and motion commands in an optimal (or near-optimal) way to generate a map of the environment, while estimating the current position of the robot on the map [23].

There are two versions of the SLAM problem: online and global. The online problem seeks to estimate the instantaneous pose of the robot, while the global problem seeks to determine all poses.

Extended Kalman Filter (EKF) applied to the SLAM algorithm is one of the oldest SLAM algorithms. This algorithm applies the extended Kalman filter to the online SLAM problem. With known correspondences, the resulting algorithm is incremental. When the correspondences are unknown, the EKF-SLAM algorithm applies an incremental maximum

likelihood estimator to the correspondence problem [1].

EKF-SLAM has been applied quite successfully to a number of robotic mapping problems. Its main disadvantages are the need for sufficiently distinct reference points and the computational complexity required to update the filter [1].

In recent years many types of SLAMs have been developed, and a division can be made according to different criteria: such as state estimation techniques, map type, real-time performance, sensors, among others. However, the classification of techniques is made according to the type of problem that is solved in the following categories: feature-based SLAM, pose-based SLAM, appearance-based SLAM and other SLAMs [24]. In the table 2.2 a comparison is made between various types of SLAM, their maps and their pros and cons.

In [23] an analysis of the SLAM problem and also of probabilistic SLAM is given.

SLAM Method	Map	Description
Filtering-based		
EKF-SLAM	feature view appearance polygon	Based on the extended Kalman filter. **Pros:** works well if features are distinct. **Cons:** adding a new feature to state space needs quadratic time, requires feature extraction in feature-based SLAM, cannot identify absence of a feature.
EIF-SLAM	feature view	Based on the extended information filter **Pros:** measurement updates are performed in constant time, effective for multiple-robot SLAM due to additivity of information. **Cons:** information matrix needs to be sparsified. Recovering map and the pose requires large matrix inversion.
PF-SLAM	feature view	Based on particle filtering. FastSLAM is an efficient implementation of particle filtering. **Pros:** effective for loop closure, performs full and online SLAM, logarithmic complexity in number of features, no need for parametrization. **Cons:** quality of the estimation is dependent on the number of particles.
DP-SLAM	view	A fast implementation of particle filtering for SLAM. **Pros:** effective for large scale maps, optimizes memory requirements. **Cons:** requires processing to recover maps.
Set-based SLAM	feature	Based on random finite set (RFS) and finite set statistics (FISST). **Pros:** number of features and data association are estimated with the Bayesian filter. **Cons:** higher time complexity than vector-based solutions.
Smoothing-based		
GraphSLAM	feature view polygon	Uses smoothing techniques to estimate the trajectory and the map. **Pros:** the whole trajectory is updated. **Cons:** computationally demanding, hard to recover covariances.
Sub-map Matching	view feature	Matches small local maps to make a large global map. **Pros:** the whole trajectory is updated, suitable for large scale environments. **Cons:** size of local maps should be adjusted.
AI-based		
AI SLAM	appearance feature	Based on artificial intelligence. **Pros:** efficient, usually no mathematical model is required. **Cons:** error-prone, requires training or parameter tuning, training can be time-consuming.

Table 2.2: Comparison of some common SLAM techniques [5].

2.2.2 Octomap

An octree is a hierarchical data structure used to represent the 3D space as shown in figure 2.4 [8]. Each node in an octree represents a cubic volume of space usually called a voxel. This volume, if not uniform, is recursively subdivided into eight sub-volumes until a certain minimum voxel size is reached or until the voxel is uniform (totally free or totally occupied space). The minimum voxel size thus determines the resolution of the octree.

Essentially, Octomap performs the probabilistic occupancy estimation at each observed

voxel to maintain updatability and to deal with sensor noise. To represent not only the occupied space, but also the free and unknown areas, an octomap explicitly models the free volumes in the tree using ray casting in the sensor model for range measurements. Figure 2.5 illustrates how Octomap models the occupied and free voxels through ray-casting. An example of a 3D map created with Octomap is shown in figure 2.6 [8].

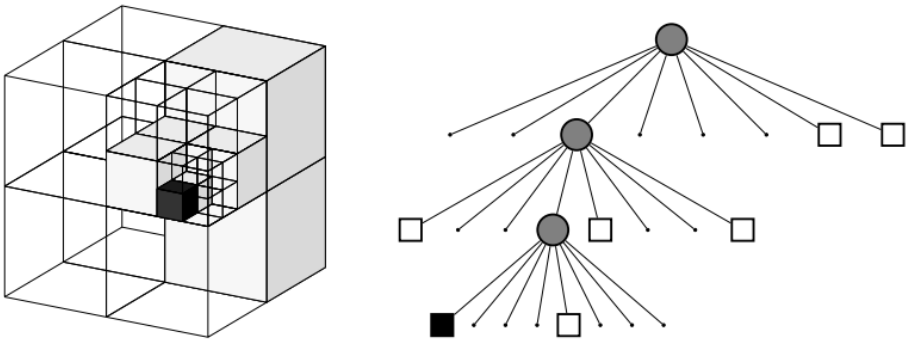


Figure 2.4: Example of an octree storing free (shaded white) and occupied (black) cells. The volumetric model is shown on the left and the corresponding tree representation on the right.

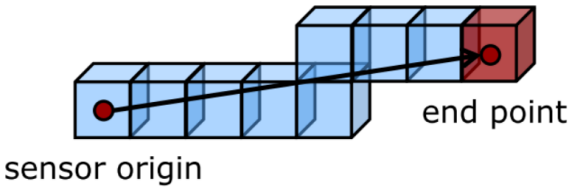


Figure 2.5: Ray-casting from sensor origin to end point, the last voxel is marked as occupied, all the other voxel along the ray are marked as free.

Occupancy Probability Estimation

Octomap integrates the sensor measurements using the occupancy grid mapping method which adopts a Bayes filtering based approach [8]. The probability that a leaf node n is occupied given the sensor data $z_{1:t}$ is estimated according to

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{2.5}$$

The occupancy probability update depends on the prior estimate $P(n|z_{1:t-1})$, the current measurement z_t and a prior probability $P(n)$. $P(n|z_t)$ denotes the probability of voxel n to be occupied given the measurement z_t. It represents the inverse sensor model and its value

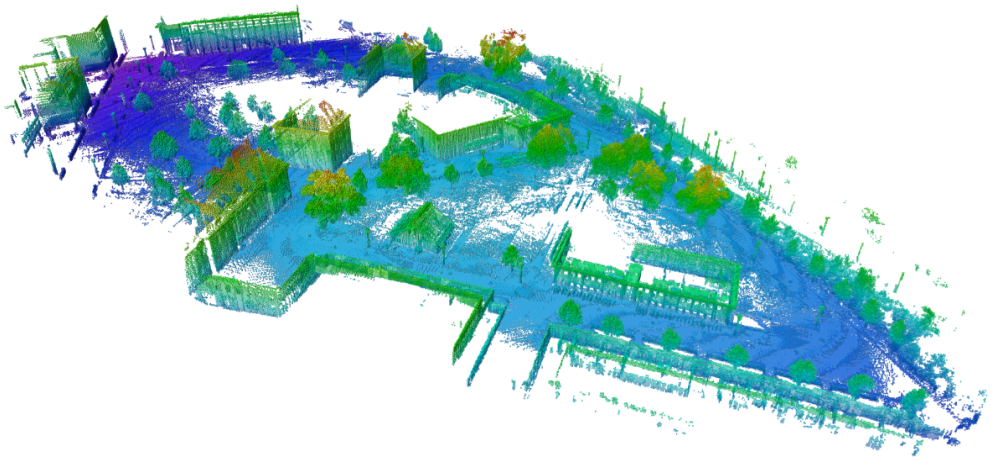


Figure 2.6: Example of a 3D map of an urban street scene created with Octomap.

depends on the sensor generating z_t. With the assumption of a uniform prior probability $P(n) = 0.5$ (unknown) and using log-odds notation, Eq. 2.5 becomes

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{2.6}$$

with

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{2.7}$$

Instead of using Eq. 2.6 directly, Octomap updates the occupancy estimate using a clamping update policy [8]:

$$L(n|z_{1:t}) = max(min(L(n|z_{1:t-1}) + L(n|z_t), l_{max})l_{min}) \tag{2.8}$$

where $lmin$ and $lmax$ denote the lower and upper limit on the value of log-odds. The modified update rule limits the number of updates required to change the state of a voxel [8].

2.2.3 Velodyne Height Map

The Velodyne height map is a Robot Operating System (ROS) package providing obstacle detection for Velodyne 3D point clouds using a height map algorithm [25]. At the basis of this algorithm is the difference between the highest and lowest z-coordinate in each xy-cell with defined "cell_size". According to the height set as threshold the objects are considered obstacles or not. For example, if the threshold is one meter, objects below one meter will be considered non-obstacles for the robot while obstacles of one meter or more in height will be considered obstacles for the robot. An example of potential obstacles is shown in figure 2.7

(the red squares).

Next are presented the Subscribed Topics, Published Topics and the parameters of the Velodyne height map package:

Subscribed Topics

→velodyne_points(sensor_msgs/PointCloud2)

3D data points.

Published Topics

→velodyne_obstacles(sensor_msgs/PointCloud2)

Points belonging to detected obstacles.

→velodyne_clear(sensor_msgs/PointCloud2)

Clear space: grid squares containing laser returns and no obstacle.

Parameters

→cell_size(double, default: 0.5)

Grid cell size (meters).

→full_clouds(bool, default: false)

Normally, height map only publishes one obstacle or clear point for each cell, significantly reducing output bandwidth while providing the information needed for most 2D navigation. When true, publish all obstacle and clear points.

→grid_dimensions(int, default: 320)

Number of grid cells in both the X and Y dimensions.

→height_threshold(double, default: 0.25)

Minimum height difference that counts as an obstacle (meters).

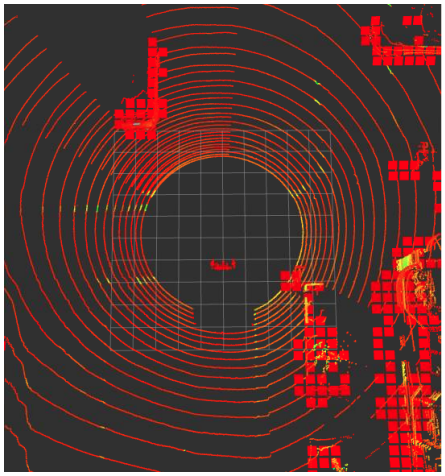


Figure 2.7: LiDAR scanning and potential obstacles using Velodyne height map.

2.3 Camera Calibration

Camera calibration is a necessary process to be able to match the points in the three-dimensional space with the pixels of the images [2].

Camera calibration, estimates the internal geometrical and optical parameters of the camera (intrinsic parameters) and the position and orientation of the camera with respect to a world coordinate system (extrinsic parameters). These parameters can be used to correct lens distortion, measure the size of an object in a plane or determine the location of the camera in the world (figure 2.8).

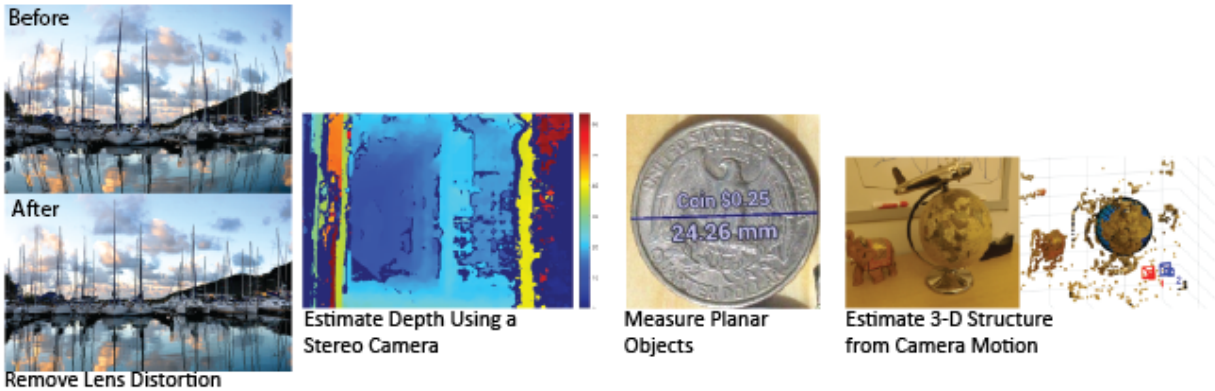


Figure 2.8: Camera calibration applications.[2]

The extrinsic parameters are represented by a rigid transformation from the 3D world coordinate system to the 3D camera coordinate system [2]. The intrinsic parameters represent a projective transformation from the 3D camera coordinates to the 2D image coordinates (Figure 2.9 and Figure 2.10).

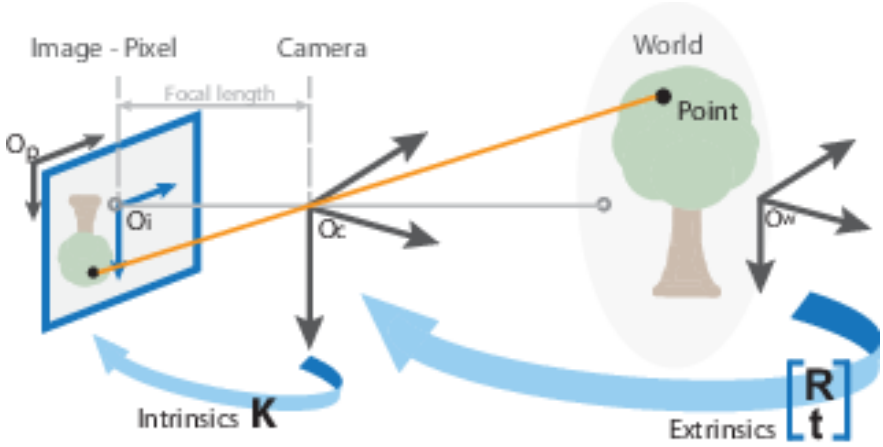


Figure 2.9: Extrinsic and intrinsic camera parameters [2].

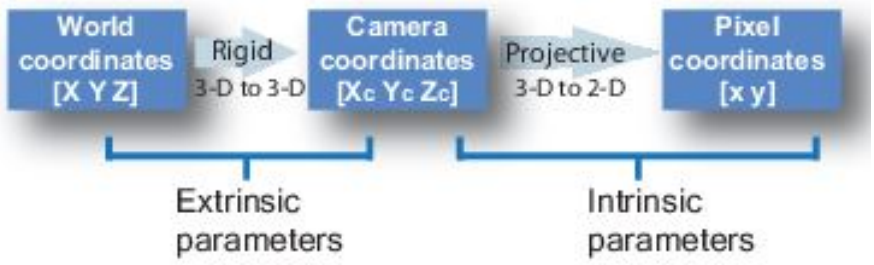


Figure 2.10: Extrinsic and intrinsic camera parameters, relationship between coordinates [2].

Extrinsic parameters

The extrinsic parameters represent the position and orientation of the camera's coordinate referential relative to another coordinate system. The origin of the camera's coordinate system is at the camera's optical centre and its xy axis defines the image plane [2]. The extrinsic parameters consist of a rotation, R, and a translation t (figure 2.11).

Figure 2.11: Extrinsic parameters of a camera [2].

Intrinsic parameters

The intrinsic parameters allow a mapping between the camera coordinates and the pixel coordinates in the image frame. The camera model in general is a mapping from the world to the image coordinates. It is a 3D to 2D transformation. These parameters depend only on the physical characteristics of the camera (sensor size, lens shift, and focal length) [2].

The intrinsic parameters include the focal length, the optical centre, and the slope coefficient. The intrinsic camera matrix, K, is defined as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ s & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

where:

- $[c_x, c_y]$ is the optical centre, in pixels.
- (f_x, f_y) is the focal length, in pixels.
- $f_x = \frac{F}{P_x}$
- $f_y = \frac{F}{P_y}$
- F is the focal length, usually expressed in millimetres.
- s is the coefficient of inclination, which is non-zero if the image axes are not perpendicular.
- $s = f_x \tan\alpha$

The pixel pitch is defined as (figure 2.12):

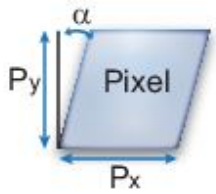


Figure 2.12: Pixel pitch.[2]

- (P_x, P_y) is the pixel size.

Camera distortion

Camera lenses can create changes in the relationship between points in the world and points in the image, i.e. distortion. Distortion can be radial or tangential.

Radial distortion happens when light rays bend closer to the edges of the lens than to the optical centre. The distortion will be greater the smaller the lens is. There are three types of radial distortion: the barrel type, the pincushion type and another more complex type which is a junction of the two types as illustrated in figure 2.13 [26].

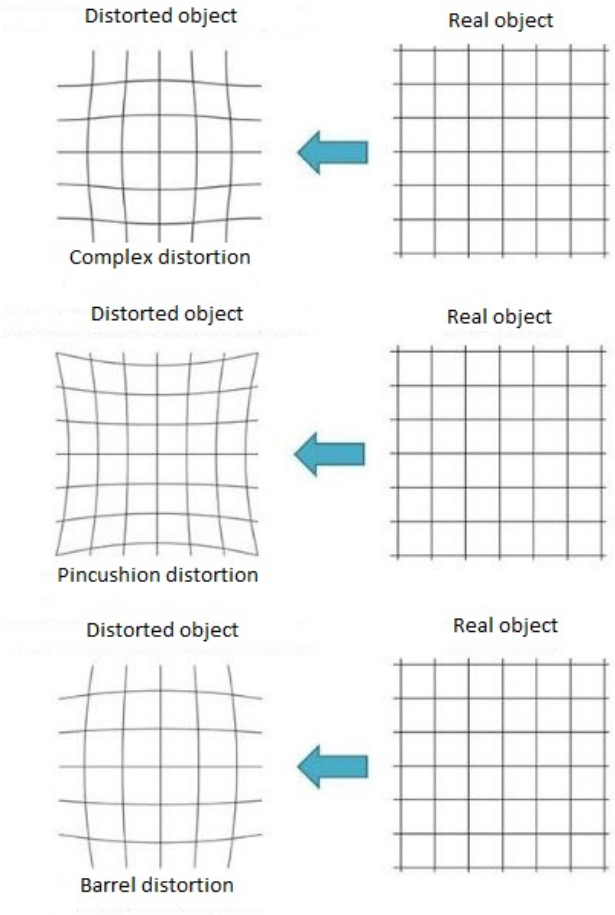


Figure 2.13: Types of radial distortion.

The new corrected coordinates are calculated by [2]:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.10}$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.11}$$

k_1, k_2, k_3 are the lens radial distortion coefficients and:

[2]https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

$$r^2 = x^2 + y^2 \tag{2.12}$$

Tangential distortion happens when the lens and the image plane are not parallel (figure 2.14). Tangential distortion coefficients model this type of distortion.

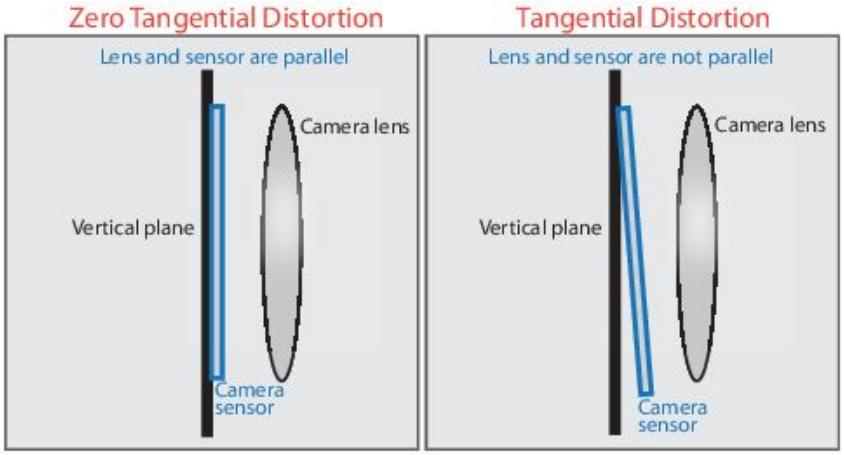


Figure 2.14: Tangential distortion. [2]

The new corrected coordinates are calculated by [3]:

$$x_{distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)] \tag{2.13}$$

$$y_{distorted} = x + [2p_2xy + p_1(r^2 + 2y^2)] \tag{2.14}$$

k_1, k_2, k_3 are the lens tangential distortion coefficients and:

$$r^2 = x^2 + y^2 \tag{2.15}$$

2.4 Remote Sensing vegetation

A Vegetation Index (VI), (also called vegetative index), is a unique number that allows quantification of plant biomass and/or plant vigour for each pixel in a remotely sensed image.

Several vegetation indexes have been developed, the Normalised Difference Vegetation Index (NDVI) being the most studied and commonly used. To generate an NDVI map, a multispectral sensor of relatively high cost is required.

[3] https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

Hunt et al. [27] studied multispectral UAV imagery for crop monitoring, and a good correlation between leaf area and green NDVI was found [28]. A map of vine vigour was proposed [29] using NDVI, calculated with a high-resolution multispectral camera. In addition to the healthy state index, NDVI can provide, for each plant, specific characteristics that can be used for plant detection [30]. NDVI has been used for many varied purposes such as spatial referencing, crop and climate monitoring, attribute mapping and in various decision support systems[31].

Today, conventional RGB cameras with high resolution are available from various manufacturers and most UAVs are equipped with standard RGB cameras [32]. Thus, a number of plant indexes have been created using only conventional RGB channels to make data collection more affordable. RGB data have been developed to generate maps similar to NDVI and minimise the cost of data acquisition, such as the Triangular Green Index (TGI) and the Visible Atmospheric Resistance Index (VARI) [33].

Arai et al. [34] proposed a method to estimate Near Infrared (NIR) reflectance using visible cameras, achieving a high correlation between green reflectance and NIR reflectance. Using the concepts of Atmospherically Resistant Vegetation Index (ARVI) [35] to reduce atmospheric effects in the vegetation index calculated using the visible spectrum, Gitelson et al. [36] proposed the VARI, which should be used to estimate the fraction of vegetation in a scene with low sensitivity to atmospheric effects. However, as pointed out by the authors, the reflectance of vegetation green surfaces is not as high and the difference between the reflectivity levels of the visible channels is not greater than between NIR and red, used to obtain NDVI.

The TGI, presented by Hunt et al. [37], uses the area of a triangle defined by three points: (480 nm, R480), (550 nm, R550), and (670 nm, R670), where R is the wavelength reflectance in the spectral features of the chlorophyll region and was developed to facilitate image acquisition using only RGB channels instead of multispectral channels to generate a chlorophyll content index.

Vegetation Indexes: NDVI and VARI

NDVI (generated using multispectral images) and VARI (generated using RGB images) are some of the most commonly used indexes for vegetation assessment. Considering that plants have a high reflectance in the NIR bands and low reflectance in the red bands, the NDVI formula evaluates this difference as it is presented in Eq. 2.16. In [38], Rouse et al. showed

good correlation between NDVI and grassland vegetation data (dry biomass, green biomass, and percent green estimates) was shown.

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{2.16}$$

To monitor the plant fraction of wheat canopies, in [36] a VI based on red and green wavelengths was proposed, as shown in Eq. 2.17:

$$VI_{green} = \frac{R_{green} - R_{red}}{R_{green} + R_{red}} \tag{2.17}$$

Then, using the concept of ARVI to reduce the atmospheric effects [35], assuming that the effect in the blue wavelengths is two times bigger than in the green and red wavelengths, the VARI was generated by the subtraction of the blue wavelength from the denominator of Eq. 2.17. The VARI, shown in Eq. 2.18, is used for estimating the vegetation fraction in a scene with low sensitivity to atmospheric effects.

$$\text{VARI} = \frac{R_{green} - R_{red}}{R_{green} + R_{red} + R_{blue}} \tag{2.18}$$

The VARI is one of the most commonly used indexes for collecting data from vegetation by utilizing only RGB cameras. It was not developed to predict NDVI values, but as an RGB-based crop index [33].

2.5 Factorization-Based Texture Segmentation

Image segmentation is an important task for a large range of applications, including remote sensing, medical imaging, and autonomous robots. Image segmentation divides an image into different parts according to its characteristics and properties. Each pixel in an image is assigned at least one of these properties. Each of the pixels within the same region is analogous with regard to some computational feature or property, like colour, intensity, texture or continuity.

The Texture segmentation literature addresses two main issues [3]:

- finding an image model that defines region homogeneity;
- devising a strategy for producing segments.

A segmentation methodology with positive results usually links a decent image model with an efficient segmentation strategy.

Much of texture segmentation strategies consist of extracting features from local image patches and then feeding them to general clustering or segmentation algorithms . Several features are assigned for the characterisation of texture appearance . The most widely used are based on filtering which uses filter banks to decompose an image into a set of sub-bands, and statistical modelling which characterises texture as resulting from some underlying probability distributions [3].

In [3] it is possible to find out that texture descriptors developed based on the local distribution of filter responses show positive behaviour in texture discrimination [39], [40], [41]. Descriptors of this type can also be combined with well-defined segmentation methods to segment textured images [41], [42].

This procedure has two main problems. The first problem originates from the high dimensionality of the multiple filter responses and their distribution representations. The second problem results from texture descriptors that are generated from image windows across boundaries [3].

As a solution to these problems already identified in [3], a factorization-based segmentation method is porposed. This is a particular method of texture descriptors based on the local distribution of filter responses, called local spectral histogram [43]. A singular value decomposition and non-negative matrix factorization is used to factorize the feature matrix.

2.5.1 Factorization based Image Model

Local Spectral Histograms

For a window $\mathbf{W}$ in an input image, a set of filter responses is computed through convolution with a chosen bank of filters $\{F^{\alpha}, \alpha = 1, 2, ..., K\}$. For a sub-band image $\mathbf{W}^{\{\alpha\}}$, the corresponding histogram is denoted as $H_{\mathbf{W}}^{(\alpha)}$.Then, the spectral histogram with reference to a filter bank is defined as [3]:

$$H\mathbf{w} = \frac{1}{|\mathbf{W}|}\left(H_{\mathbf{W}}^{(1)}, H_{\mathbf{W}}^{(2)}, ..., H_{\mathbf{W}}^{(K)}\right), \tag{2.19}$$

where $||$ denotes cardinality. A local spectral histogram centred on each pixel location is calculated over the square window. In order to obtain meaningful features, the integration scale has to be large enough.

Image Model

Consider an image composed of homogeneous texture regions, figure 2.15(left). Assume that spectral histograms within homogeneous regions are approximately constant. From windows within each region local spectral histograms representative of each region can be calculated [3]. Considering for now only the intensity filter, which gives the intensity value of each pixel as the filter response. Then the local spectral is equivalent to the histogram of a local window.

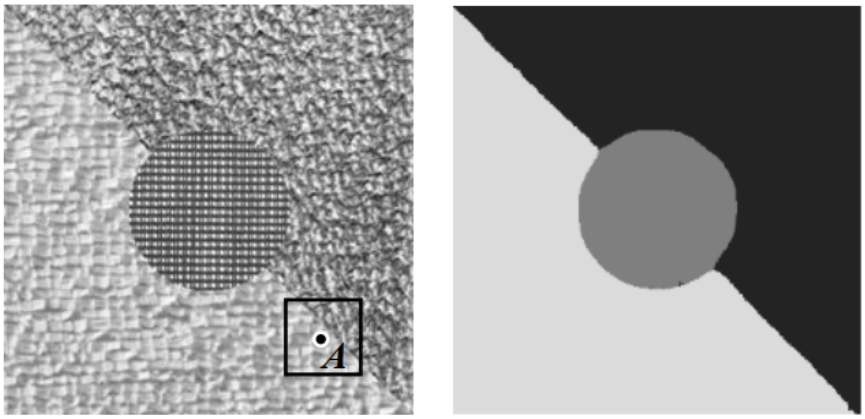


Figure 2.15: (left) A textured image. (right) Segmentation result using least squares estimation [3].

The feature of each pixel can be considered as the linear combination of all representative features weighted by the corresponding area coverage. In the case where a window is completely within a region, the weight of the representative feature for that region is close to one, while the other weights are close to zero.

Given an image with N pixels and feature dimensionality of M, all the feature vectors can be compiled into an $M \times N$ matrix, $\mathbf{Y}$. Assuming that there are L representative features, the image model can be expressed as:

$$\mathbf{Y} = \mathbf{Z}\beta + \varepsilon, \tag{2.20}$$

where $\mathbf{Z}$ is an $M \times L$ matrix whose columns are representative features, β is an $L \times N$ matrix whose columns are weight vectors, and ε is model error.

The representative feature matrix $\mathbf{Z}$ can be computed from manually selected windows within each homogeneous region, and β is then estimated by least squares estimation [3]:

$$\hat{\beta} = \left(\mathbf{Z}^T\mathbf{Z}\right)^{-1}\mathbf{Z}^T\mathbf{Y}. \tag{2.21}$$

Segmentation is obtained by examining $\hat{\beta}$ – each pixel is assigned to the segment where the corresponding representative feature has the largest weight.

2.5.2 Factorization based Segmentation

For fully automatic segmentation, both $\mathbf{Z}$ and β are unknowns,and the objective is to estimate these two matrices by factoring $\mathbf{Y}$. In this section, the factorisation algorithm, which can produce segmentation with high accuracy and efficiency, is presented [3].

Low Rank Approximation

In order to exist a unique solution in 2.20, the $\mathbf{Z}$ must be full-level full rank, so that the $(\mathbf{Z}^T\mathbf{Z})$ in 2.21 is invertible.

A typical solution to low rank approximation is Singular Value Decomposition (SVD) [44], where the feature matrix is decomposed into:

$$\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{2.22}$$

here $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices of size $M \times M$ and $N \times N$, respectively. The columns of $\mathbf{U}$ are the eigenvectors of matrix $\mathbf{Y}\mathbf{Y}^T$, and the columns of $\mathbf{V}$ are the eigenvectors of matrix $\mathbf{Y}^T\mathbf{Y}$. $\mathbf{\Sigma}$ is an $M \times N$ rectangular diagonal matrix, where the diagonal terms, called singular values, are the square roots of the eigenvalues of the matrix $\mathbf{Y}\mathbf{Y}^T$, or $\mathbf{Y}^T\mathbf{Y}$. The wellknown Eckart-Young theorem [21] states that the best rank-r approximation to $\mathbf{Y}$, in the least-squares sense, has the same form of SVD, except that $\mathbf{\Sigma}$ is replaced with a new matrix that contains only the first r singular values (the other singular values are replaced by zero).

It is necessary to determine the underlying rank of the feature matrix, which corresponds to the number of representative features, or segments. Let $\mathbf{Y'}$ be the approximated matrix of rank-r. The approximation error can be obtained as follows:

$$\|\mathbf{Y} - \mathbf{Y}'\| = \sqrt{\sum_{i=r+1}^{M} \sigma_i^2}, \tag{2.23}$$

where $\|\|$ denotes the Frobenius norm, which is the square root of the sum of all squared matrix entries. $\sigma_1, \sigma_2, ..., \sigma_M$ are singular values in a nonincreasing order. The error then corresponds to the singular values discarded in the approximation. It is thus possible to

determine the number of segments through the error threshold. That is, the segment number n is estimated as

$$n = min\left\{ i : \frac{1}{N}\sqrt{\sum_{i+1}^{M} \sigma_i^2} < \omega \right\}, \tag{2.24}$$

where ω is a pre-specified threshold that depends on the noise level of images and specific tasks.

SVD Based Solution

On the principle that the first r singular values are chosen using 2.24, 2.22 can be rewritten as:

$$\mathbf{Y'} = \mathbf{U'\Sigma'V'}^T, \tag{2.25}$$

where **U'** and **V'** consist of the first r columns of **U** and **V** in the SVD of matrix Y, respectively. $\mathbf{\Sigma'}$ is an $r \times r$ matrix with the largest r singular values on the diagonal. If defined $\mathbf{Z}_1 = \mathbf{U'}$ and $\beta_1 = \mathbf{\Sigma'V'}^T$, the two matrices $\mathbf{Z}_1$ and β_1 are of the same size as the matrices $\mathbf{Z}$ and β in 2.20. Thus, $\mathbf{Z}_1$ and β_1 can serve as a solution in the model in 2.20, which simultaneously ensures a minimum least square error due to the Eckart-Young theorem. However, the decomposition is not unique due to the fact that

$$\mathbf{Y'} = \mathbf{Z}_1\beta_1 = \mathbf{Z}_1\mathbf{Q}\mathbf{Q}^{-1}\beta_1 \tag{2.26}$$

where $\mathbf{Q}$ can be any invertible square matrix, suggesting that $(\mathbf{Z}_1\mathbf{Q})$ and $(\mathbf{Q}^{-1}\beta_\mathbf{1})$ can also be possible solutions. $\mathbf{Z}_1$ and β_1 generally differ from the desired matrices that represent underlying representative features and combination weights. Although the decomposition cannot directly give a valid solution, it leads to a self-evident fact that the representative features must be in the form of $\mathbf{Z}_1\mathbf{Q}$, i.e., a linear transformation of $\mathbf{Z}_1$. Similarly, the combined weights must be a linear transformation of β_1.

For a segmentation result, it is necessary to estimate $\mathbf{Q}$. On the principle that the desired matrix of representative features is a linear transformation of $\mathbf{Z}_1$, it is known that the representative features have to be located in an r-dimension subspace, delimited by the columns of $\mathbf{Z}_1$. Since $\mathbf{Z}_1$ forms an orthonormal basis, each column of $\mathbf{Q}$ corresponds to the Cartesian coordinate of each representative feature in the subspace.

Influence of Integration Scale

Local spectral histograms involve multiple scale parameters, including filter scales and integration scales. For a structure tensor, one scale corresponds to the computational gradient scale, and the other describes the extent of the local patches on which the structure tensor is built. For local spectral histograms, with multiple filters, it is more complicated to couple the filter scales with the integration scales.

The selection of the integration scales has a direct effect on the segmentation results. More specifically, as the integration scale increases, the proposed method produces smoother boundaries. To illustrate such an effect, an image containing clipped boundaries is shown in figure 2.16(left), where a square window is used to calculate a local feature. According to the coverage of the two regions within the window, the proposed method segments the corresponding pixel (the point) into the darker region, as shown in figure 2.16(middle). With the integration scale large enough, we obtain a segmentation result shown in figure 2.16(right), where the boundary is close to a straight line.

Figure 2.16: Illustration of the smoothing effect. (left) Synthetic image containing two regions with different Gaussian noises. (middle) Segmentation result using the method porposed in [3]. (right) Segmentation result with a very large integration scale

.

2.6 Software and Hardware

In this section is made a presentation of the software and hardware used. A brief analysis is made of the Robot Operating System (ROS), and of some software already implemented in ROS and also a brief description of the robot and sensors that will be used.

2.6.1 ROS: Robot Operating System

The ROS[4] is a collection of software frameworks for robot development. Most of this software is open source, allowing other users to test, reuse or improve already implemented software.

The basic concepts[5] of ROS are nodes, master, parameter server, messages, services, threads and bags.

The advantages of using ROS in this dissertation are as follows:

- Availability of a large set of packages developed for robots.
- Provides a message exchange interface that allows communication between processes.
- Provides common robot-specific libraries and tools, such as mapping, navigation, localization, pose estimation, diagnostics, etc.
- Provides three-dimensional visualisation of many types of sensors using the **rviz** package, signals and graphical data using the **rqt** package.

One of the disadvantages of ROS is that it is not cross-platform, i.e. it is not fully supported on platforms such as *Windows* or *Android*.

One of the disadvantages of ROS is that it is not cross-platform, i.e. it is not fully supported on platforms such as *Windows* or *Android*. ROS is designed for *Unix* type systems. One way to use ROS in a *Windows* environment is through Matlab's ROS Toolbox.

2.6.2 Software available in ROS

LiDAR Odometry and Mappin (LOAM)

LOAM is a method for state estimation and mapping, using a 3D, real-time LiDAR [45]. Essentially, LOAM uses range measurements captured by a LiDAR to locate itself in an environment and build a map of it. The program contains two major threads running in parallel. An "odometry" thread computes motion of the LiDAR between two sweeps, at a higher frame rate. It also removes distortion in the point cloud caused by motion of the LiDAR. A "mapping" thread takes the undistorted point cloud and incrementally builds a map, while simultaneously computes pose of the LiDAR on the map at a lower frame rate. The LiDAR state estimation is combination of the outputs from the two threads [45].

[4]http://wiki.ros.org/Documentation

[5]http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Concepts.html#ROS_Computation_Graph_Level

Lightweight and Ground-Optimized LiDAR Odometry and Mapping (LeGO-LOAM)

LeGO-LOAM [46] is computationally lightweight and can be used in an embedded system. LeGO-LOAM is ground-optimised as it takes advantage of the presence of the ground plane in its segmentation and optimisation steps. In [46], a comparison is made with the LOAM method, where it is shown that the LeGO-LOAM method can achieve similar or better results in accuracy.

RTAB-Map (Real-Time Appearance-Based Mapping)

RTAB-Map [47] is a graph-based SLAM approach that has been integrated into ROS as a package[6]. RTAB-Map (Real-Time Appearance-Based Mapping) is an RGB-D Graph SLAM approach based on a global Bayesian loop closure[7] o detector. It is a graph-based SLAM technique because it determines the probability of a new image coming from a previous location or a new location through loop closure process . When a good assumption is accepted, these new images are added and optimized by a graph optimizer. The approach in [48] consists of four steps: extracting visual features from images and then matching these features with others from previous images. After that, they obtain a set of 3D point correspondences between any two frames and finally they can estimate the relative transformation that relates the frames. One of the major strengths of these techniques is their compatibility with a wide variety of devices, namely a handheld Kinect, a stereo camera or 3D LiDAR for 6DoF mapping, or a laser rangefinder for 3DoF mapping. Figure 2.17 illustrates an example of a map with the RTAB-Map [49].

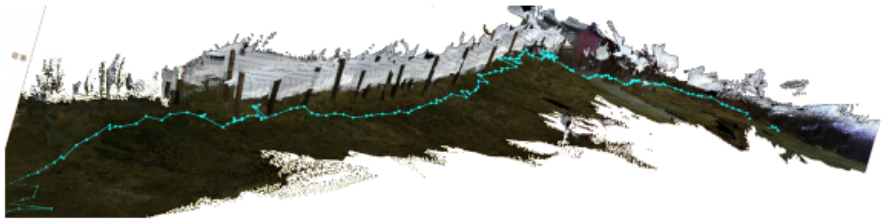


Figure 2.17: 3D reconstruction of a fenced agricultural field using RTAB-Map.

[6] http://wiki.ros.org/rtabmap$_r os$

[7] Loop closure consists of detecting when the robot has returned to a known previous position and thus determining the deviation between the actual and the calculated position. This allows errors in the estimated trajectory to be drastically reduced.

3D LiDAR-based Graph SLAM

The **hdl_graph_slam** is an open source ROS package for real-time 6DoF SLAM using a 3D LiDAR. It is based on 3D graphical SLAM with odometry estimation based on NDT matching and loop detection. This package has been used with Velodyne (HDL32e, VLP16) and RoboSense (16 channels) sensors in indoor and outdoor environments [50]. The **hdl_graph_slam** has as input a point cloud obtained through LiDAR and has as output a point cloud resulting from the registration of the various point clouds obtained during navigation.

The input point cloud is first downsampled by the node **prefiltering**, and then passed to the next nodelets as shown in figure 2.18. While the **scan_matching_odometry** nodelet estimates the sensor pose by iteratively applying a scan match between consecutive frames (i.e., odometry estimation), the **floor_detection** nodelet detects the floor planes by RANdom SAmple Consensus (RANSAC). The estimated odometry and the detected floor planes are sent to the **hdl_graph_slam** node. In order to compensate for the accumulated error of scan matching, it performs loop detection and optimises a pose graph.

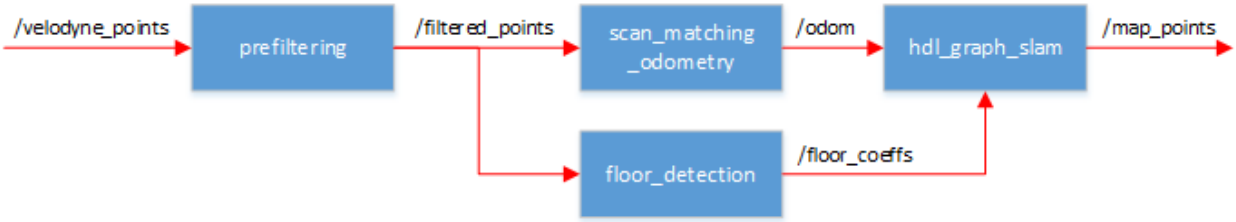


Figure 2.18: Nodelets of hdl_slam.

2.6.3 Ranger

The Ranger, the robot that will be used in the SafeForest project is based on a 4000 kg Bobcat T190 (figure 2.19). This platform carries all equipment (cutting tool, sensor kit, computer or processing system) necessary to complete the mission. Being a widely used machine in a variety of fields, it allows for specialized maintenance to be readily available.

In figure 2.20 is possible to see the hardware distribution used in Ranger, which may suffer future changes.

Figure 2.19: Ranger, robot used in the project.[4]

The use of sensors such as cameras and LiDAR will make it possible to observe the tracks of the machine, the tool and the areas immediately in front of and behind the robot, ensuring the safety of personnel and animals in the vicinity of the machine during operation. The use of thermal cameras enables the Ranger to detect temperature differences and thus to detect people and animals, as well as ignition sources caused by the use of the cutting tool in dry vegetation.

The use of a multispectral camera or an RGB camera on the Ranger allows the detection of different wavelengths, such as visible light, infrared, ultraviolet or any other band of the spectrum allowing to distinguish elements that can be identical in the visible light band, but different in other bands allowing to verify various types of vegetation and also vegetation in various states (cut, dry,etc).

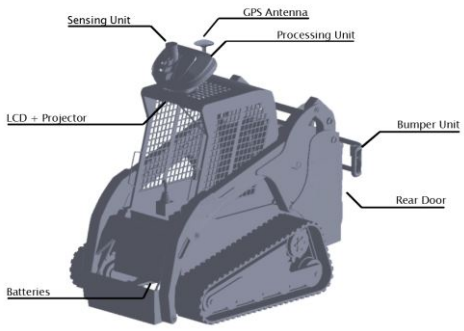


Figure 2.20: Hardware distribution in Ranger.[4]

2.6.4 LiDAR

The LiDAR that was used in this Dissertation is the RS-LiDAR-M1 of Real Sense [51]. RS-LiDAR-M1 is an automotive grade solid-state LiDAR, that RoboSense specially designed for massive production vehicles. Some of the main features of RS-LiDAR-M1 are:

- Wavelength: 905 nm.
- Compact and lightweight.
- Accuracy: up to +/- 5 cm.
- Measuring capacity up to 200 m distance.
- Vertical Field Of View (FoV): 25^{o} (-12.5^{o} -+ 12.5^{o}).
- Vertical angular resolution: 0.2^{o}.
- Horizontal FoV: 120^{o} (-60.0^{o} -+ 60.0^{o}).
- Horizontal angular resolution: 0.2^{o}.
- Ability to resist interference from other LiDAR and ambient light.

2.6.5 RGB camera

A RGB camera is an imager that collects visible light (400 700nm). RGB cameras utilize wavelengths of light from 400 700nm, which is the same spectrum that the human eye perceives. In some digital cameras the image sensor is a charge-coupled device (CCD), while in others it's a CMOS sensor. One of the features most exploited by digital camera manufacturers is the resolution of the camera sensor, measured in megapixels. Other important factors for the quality of photos/videos are the quality of the lens and the algorithm (internal camera software that processes captured data).

In this Dissertation, the camera used was a 13MP smartphone camera[8].

[8]https://www.gsmarena.com/xiaomi_redmi_note_4-8531.php

3 Developed Methodology

In this section the whole methodology developed in this Dissertation is described based on the acquisition and processing of data acquired by LiDAR and also by RGB camera. The objective of the developed methodology is to distinguish between vegetation that can be cut, and vegetation that cannot be cut or other obstacles. Using LiDAR, the distinction is made through the height of the objects, the reflectance intensity value of each point cloud point and also through the size of each object, such as its diameter. Using the camera, the classification is done by merging vegetation indexes with segmentation.

3.1 LiDAR

In this subsection the work developed using LiDAR is presented: separation of point clouds on ground and objects, height maps, calculation of trees and their position, and finally obstacle classification.

3.1.1 Separation of Objects from the Ground

When the point cloud is captured by LiDAR, the point cloud is in the raw state, i.e. all points (x,y,z) and the reflection intensity of each point. For a better analysis, these points have to be processed by filtering. In this case, a progressive morphological filter 3.1.1 which is part of the Point-Cloud library[1] is applied.

The application of this filter splits the original point cloud in two. One contains the points considered ground while the other contains the points considered objects. In figure 3.1 can be seen the point cloud that corresponds to the scene in figure 3.2.

[1] https://pointclouds.org

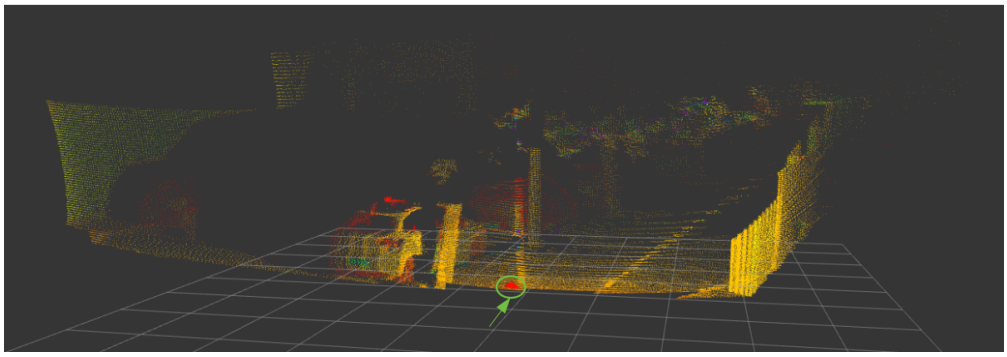


Figure 3.1: Original Point cloud represented by Intensity (Point cloud colour).

Note: in figure 3.1 the group of points marked in green with a circle and an arrow are result of the LiDAR error since they are within a zone where LiDAR should not capture any point.

Figure 3.2: Scene corresponding to the point cloud of the figure 3.1.

The filter implementation is run in $c++$ as demonstrated in the following code excerpts. After reading the original point cloud, the filter is then created, in this case with the default parameters. The output (the indices of ground returns) is computed and stored in ground.

```
1 // Create the filtering object
2 pcl::ProgressiveMorphologicalFilter<pcl::PointXYZ> pmf;
3 pmf.setInputCloud (cloud);
4 pmf.setMaxWindowSize (20);
5 pmf.setSlope (1.0f);  //terrain slope [-1.57 1.57]rad
6 pmf.setInitialDistance (0.5f); //initial elevation difference threshold [m]
7 pmf.setMaxDistance (3.0f); //maximum elevation difference [m]
8 pmf.extract (ground->indices);
```

To extract the ground points, the ground index are passed into a **pcl::ExtractIndices** filter.

```
// Create the filtering object
pcl::ExtractIndices<pcl::PointXYZ> extract;
extract.setInputCloud (cloud);
extract.setIndices (ground);
extract.filter (*cloud_filtered);
```

To get the points corresponding to the objects the filter is called with the same parameters but with the output negated.

```
// Extract non-ground returns
extract.setNegative (true);
extract.filter (*cloud_filtered);
```

The two output point clouds that result from the application of this filter can be seen in figures 3.3 and 3.4. In figure 3.3 it is possible to observe the point cloud that corresponds to the ground and in figure 3.4 is possible to observe the point cloud that corresponds to the objects.

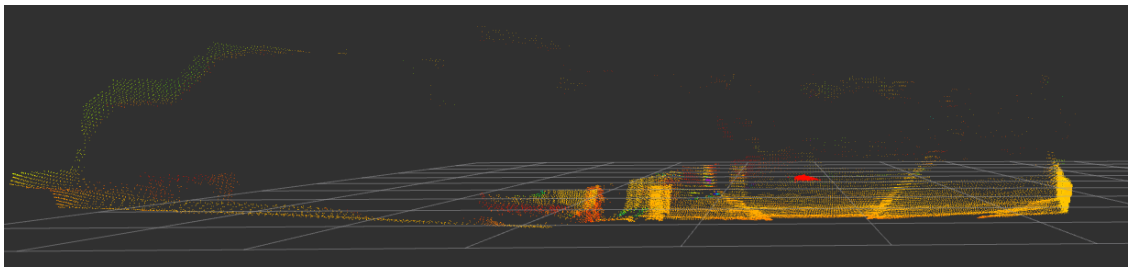


Figure 3.3: Point cloud corresponding to the ground after the application of the morphological filter, using the default parameters.

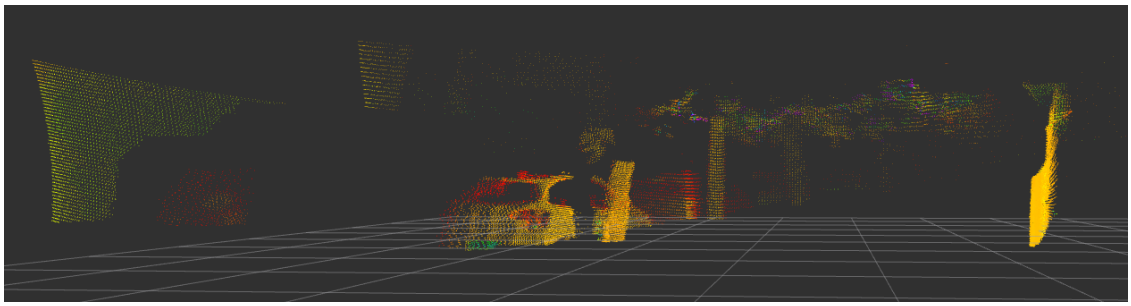


Figure 3.4: Point cloud corresponding to the objects after the application of the morphological filter, using the default parameters.

In figure 3.3 can be seen that most of the points are well filtered, however on the wall on the left it can be seen that some points above the projection of the first car have been

poorly separated. Also some points corresponding to the trees were incorrectly separated. It is also possible to observe that the objects closest to the LiDAR have been cut by the filter i.e. lower parts of the objects have been wrongly separated.

Through trial and error tests carried out altering the filter parameters (next code excerpt) it is possible to observe in figure 3.5 and 3.6 that by altering the parameters, particularly the maximum distance *"pmf.setMaxDistance"* it was possible to filter the point cloud more effectively. The parameter *"pmf.setSlope"* is not changed since the terrain has no slope.

```
1  // Create the filtering object
2  pcl::ProgressiveMorphologicalFilter<pcl::PointXYZI> pmf;
3  pmf.setInputCloud (cloud);
4  pmf.setMaxWindowSize (20);
5  pmf.setSlope (0.2f);
6  pmf.setInitialDistance (0.1f);
7  pmf.setMaxDistance (1.5f);
8  pmf.extract (ground->indices);
```

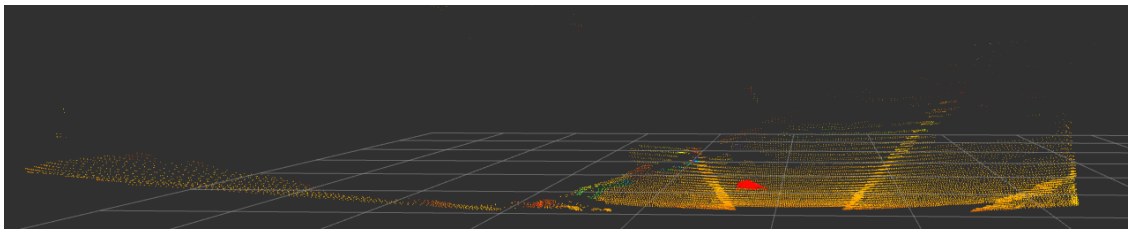


Figure 3.5: Point cloud corresponding to the ground after the application of the morphological filter, using updated trial and error parameters.

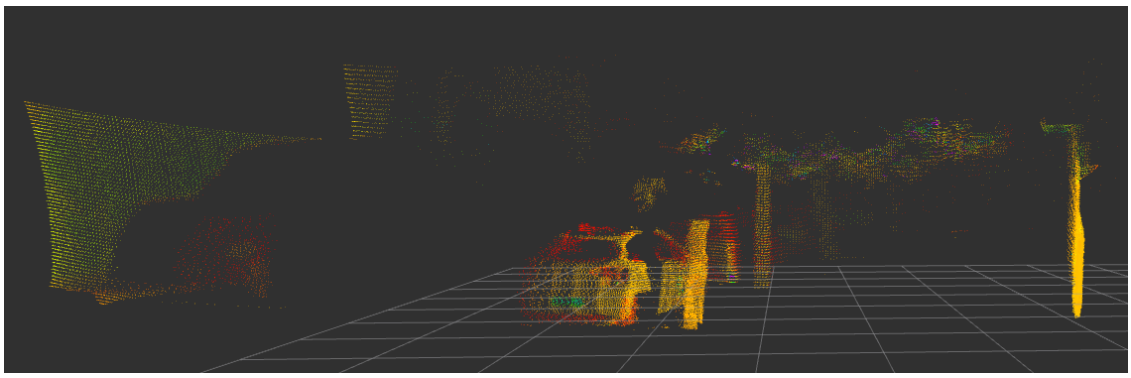


Figure 3.6: Point cloud corresponding to the objects after the application of the morphological filter, using updated trial and error parameters.

From figure 3.5 it is verified that all the points obtained by filtering correspond to points on the ground, i.e., calculating the difference between z-max and z-min at a given area is

zero. In figure 3.6 it can be seen that the left side-walk has been considered as an obstacle, however this is an acceptable error as the side-walk is distant from the LiDAR and is one plateau above the road.

Slope Parameter Calculation

The parameter **slope** needs to be updated along the robot's path due to the variation of inclination of the terrain in front of the robot. The parameter **slope** is the relative slope that the terrain has in front of the robot, and is relative to the LiDAR horizontal plane. For calculation of the parameter **slope** there are various options to take into account. One of them is to provide a pre-defined value initially and then adjust the value of the **slope** using one or more comparison elements. Since the **Z-max** and **Z-min** of point cloud of ground do not change by modifying the value of the **slope** it is necessary to find another comparison parameter.

The parameter chosen was **"areas where the difference between z-min and z-max is more than 20 centimetres"**. Areas with a height of more than 20 centimetres are taken into account to increase or decrease the slope value.

By supplying the slope parameter with a value of 0.5f, it is then compared to the parameter **"areas where the difference between z-min and z-max is more than 20 centimetres"** which must converge to one or zero. Another parameter for comparison is the number of points that are part of the ground. Since the original point cloud normally has 78750 points, the ground will hardly have less than 10000 points so it will also be a parameter for comparison (see figure 3.7).

Table 3.1: Comparison of results of the ground point cloud by changing the **slope** a parameter.

Parameter	**Areas where the difference between z-min and z-max is more than 20 centimetres**	**Number of ground points**
Slope: 0.5f	2	24820
Slope: 0.0f	0	478
Slope: 0.1f	1	17391
Slope: 0.2f	1	24780
Slope: 0.3f	1	24796
Slope: 0.4f	2	24804
Slope: 0.6f	4	24840

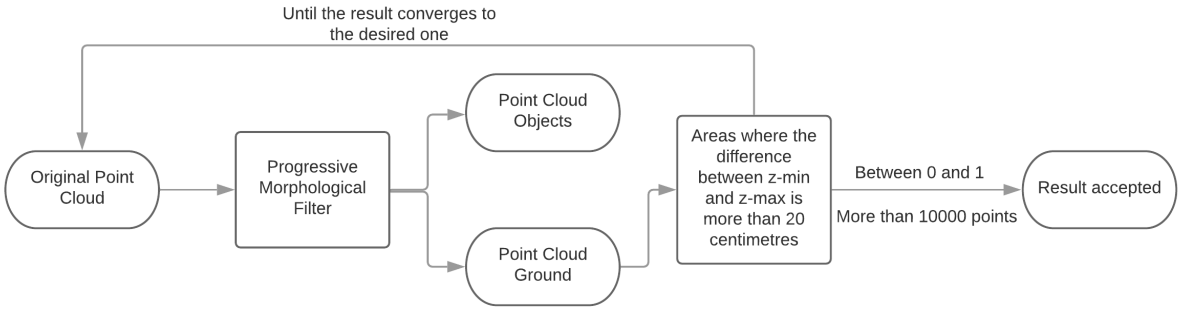


Figure 3.7: Flowchart of the algorithm described above. Slope Calculation.

Another way to obtain a value for the slope using the FoV of LiDAR is shown below (see figure 3.8):

1. Using the point cloud of the ground obtained using a pre-defined initial slope value.
2. Knowing how high **h** the LiDAR was off the ground at the time the data was obtained;
3. If the LiDAR is placed **h** meter above the ground, and the ground is parallel to the LiDAR's axis, as the LIDAR has a vertical FoV of 12.5^{o}, it will only hit the ground at the distance **d**= $\frac{h}{tg12.5}$ of the Lidar;
4. By checking all points at distance **d** and averaging the z-min of each (x,y) at that distance it is then possible to calculate the difference between the expected value **h** if the ground was parallel and the value obtained by LiDAR;
5. With this difference it is possible to calculate an approximate value of the slope. The angle of the slope will be calculated using the expression $\alpha = tg^{-1}(\frac{difference}{d})$, were α is the new slope.

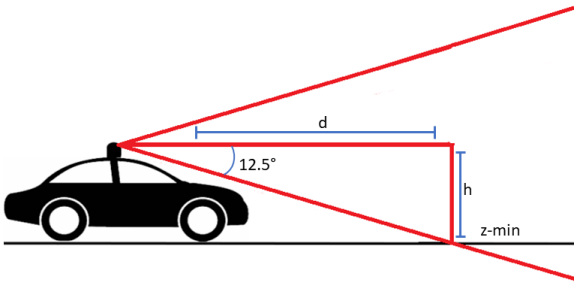


Figure 3.8: Vertical FoV of the LiDAR.

Another option is for the robot to be equipped with an inertial measurement unit (IMU), or inclination measure. Using such a device would be useful if the changes in the slope of the terrain were not too abrupt, i.e. in the forest for example, there may be times when it is going downhill and then suddenly it is going uphill. However in this case, it is not applicable as only LiDAR is being used.

3.1.2 Height map

By using the ROS package "Velodyne height map" it is possible to generate a height map using the original point cloud (see figure 3.9) of the scene in figure 3.10. The package "Velodyne height map" divides the point cloud into cells, in this case cells of size 0.4 x $0.4m$ and calculates the difference between z-min and z-max to calculate the "height". This package will separate obstacles from non-obstacles by a height threshold, which in this case is 1 metre.

By setting the height threshold desired by the robot user on the terrain, the package will separate objects into obstacles and non-obstacles by the height of the objects. For example, if the threshold considered is one meter, all objects taller than one meter will be considered obstacles by the robot i.e. assuming that the navigation is done in a forest where the trees are already big, it is possible to navigate with only trees as obstacles. Bushes lower than one metre are considered non-obstacles and should be cut back.

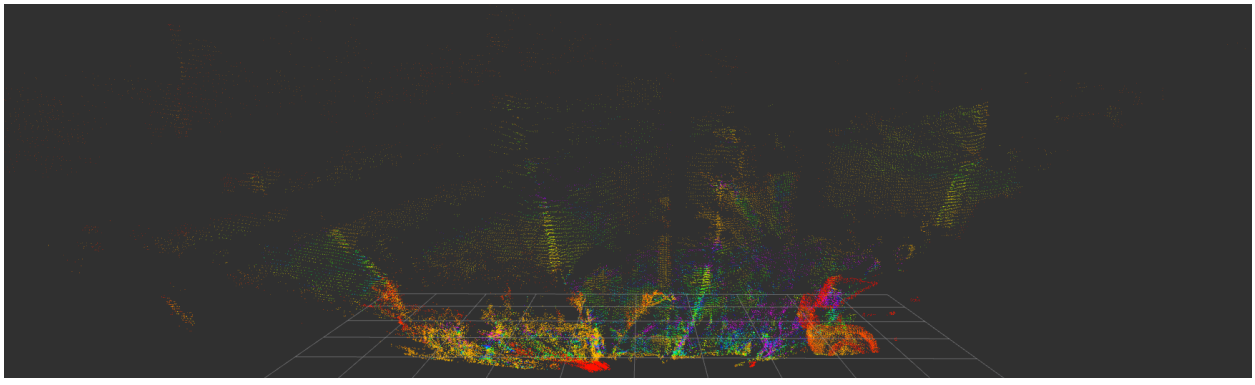


Figure 3.9: Original point cloud, input for the height map.

Figure 3.10: Partial photo of the scene corresponding to the point cloud in figure 3.9.

The application of this ROS package allows the creation of two point clouds from the original point cloud. Each point cloud created has each point corresponding to the size of the cell 0.4 x 0.4m and with z equal to the difference between z-min and z-max in that cell. One point cloud contains the obstacles, i.e. objects that exceed the defined height threshold are considered obstacles, and another point cloud contains the objects or the ground where the objects are below the defined threshold and are considered non-obstacles. This can be seen in figure 3.12, where red corresponds to the point cloud containing the obstacles and green corresponds to the non-obstacles or the ground.

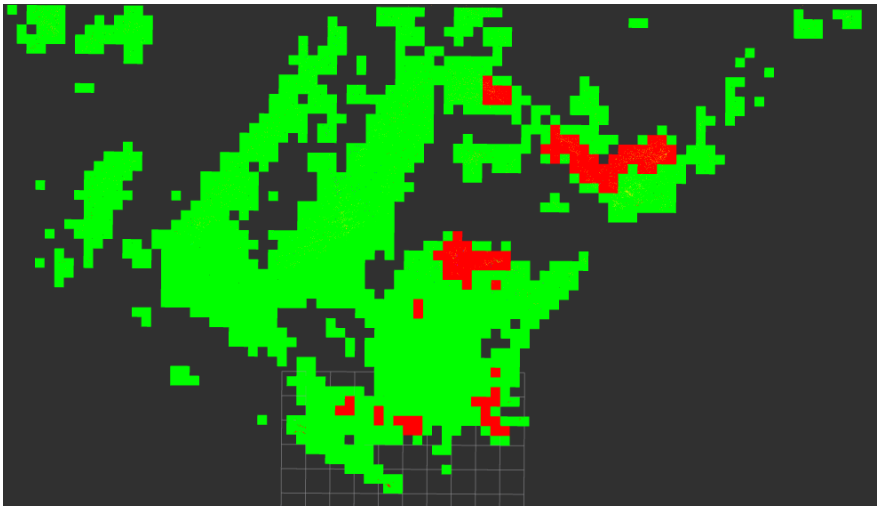


Figure 3.11: Overlapping point clouds, red for obstacles and green for the non-obstacles.

Using the difference between z-min and z-max calculated by the package "Velodyne height map" it is possible to create a point cloud with the heights of all the objects present, as it is

possible to observe in figure 3.12.The points are represented according to the present scale.

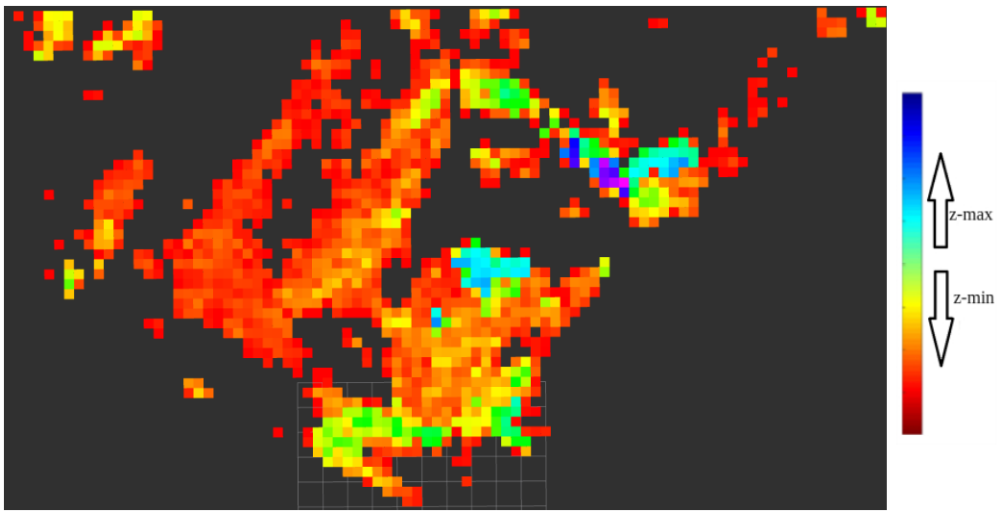


Figure 3.12: Point cloud containing the objects/ground represented (colour) by their height.

Depending on the type of navigation to be done, or in this case which obstacles to take into account for the robot's navigation, through this package it is possible to create a map according to the defined height. For example, in the case of clearing forest land, the height threshold of the package can be defined as a value that makes for example all trees present as obstacles. Using the point cloud of the figure 3.12 it is also possible to divide the terrain by heights.

3.1.3 Octomap and Z-coordinate cut

The Octomap library[2] implements a 3D occupancy grid mapping approach, i.e. 3D map of the scene. The Octomap library also allows the creation of a grid map from the 3D map, both with the same resolution. In the following examples the minimum resolution used is 0.2 meters. The input of the Octomap package is a point cloud, and as output, a 3D map and a grid map.

Using the progressive morphological filter described previously, the original cloud (see figure 3.13) was split in two point clouds (ground and objects). Using the ROS package **pcd2octo**, both point clouds were represented in an octomap. In figure 3.15 it can be seen the representation of the terrain using the octomap that can give information of the terrain and in figure 3.14 the representation of the obstacles giving information of their 3D shape.

[2]https://octomap.github.io

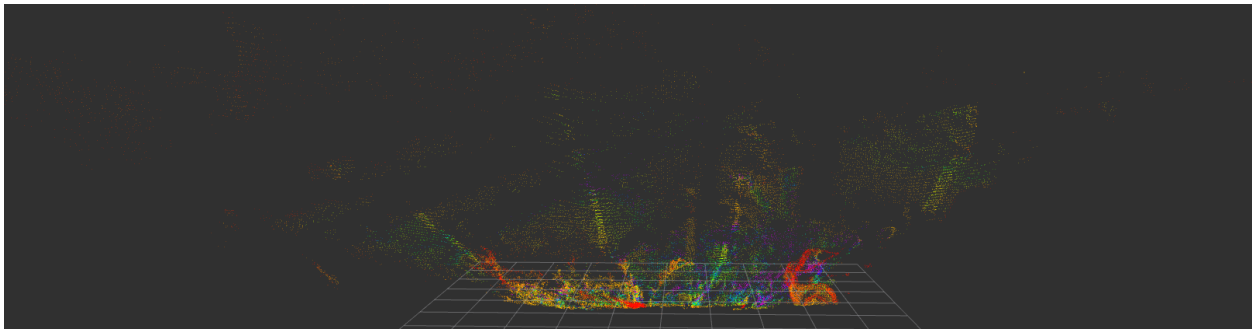


Figure 3.13: Original point cloud.

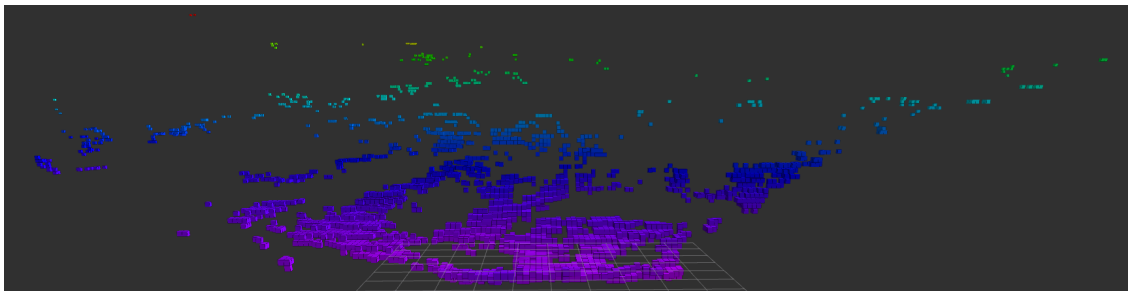


Figure 3.14: OctoMap representation of the point cloud containing the ground points resulting from the application of the progressive morphological filter to the original cloud.

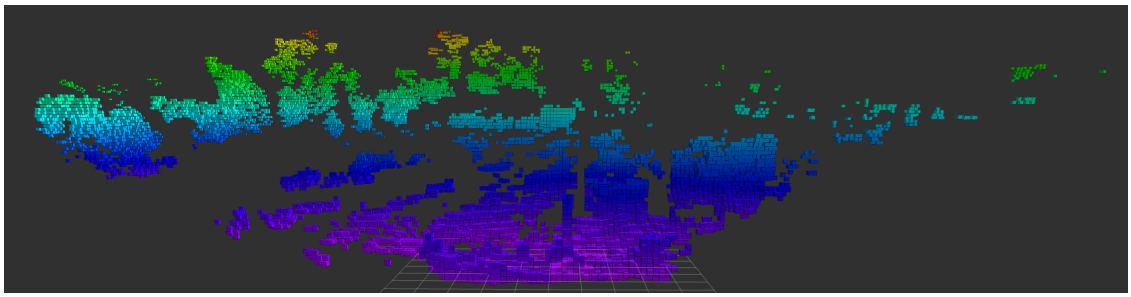


Figure 3.15: OctoMap representation of the point cloud containing the objects points resulting from the application of the progressive morphological filter to the original point cloud.

A **PassThrough** filter is applied to the point cloud objects, in this case in the z-coordinate dimension, allows us to make a cut in the cloud in a range. In figure 3.16 it is possible to see the original cloud, and also the octomap representation of the cut made between 0.5 m and 0.8 m above the LiDAR origin (z-coordinate).

In the case of a tree, when a grid map is created that contains it, its representation will be done by overlapping the crown with the trunk. Thus, using the points resulting from the application of the filter, it can just create a grid map of the tree trunk (semicircle). This method allows the perception of the objects' limits.

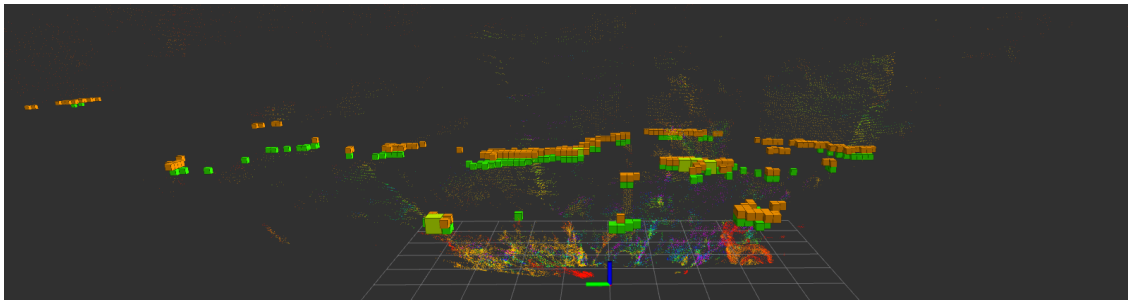


Figure 3.16: Octomap representation of the z-cut made between 0.5 m and 0.8 m above the LiDAR origin in the point cloud.

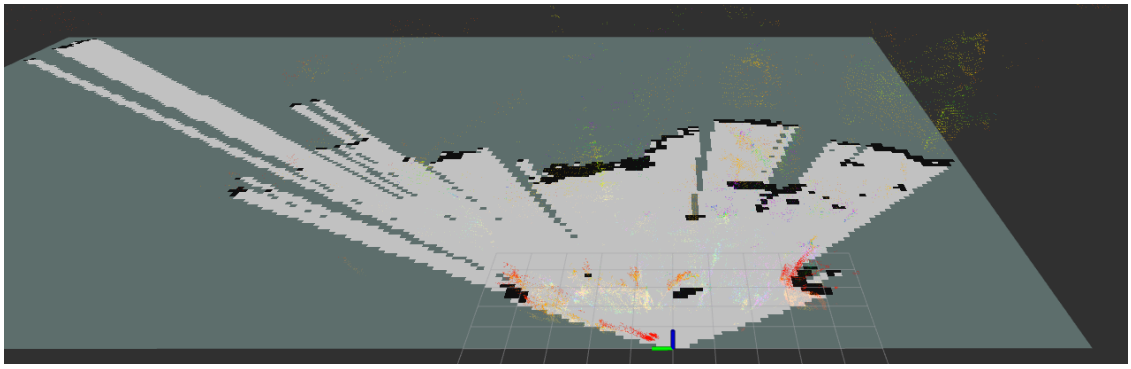


Figure 3.17: Grid map resulting from the z-cut in the point cloud.

3.1.4 LiDAR Intensity

LiDAR intensity is a value captured for each point of the return intensity of the laser pulse that generated the point. It depends, in part, on the reflectivity of the object hit by the laser pulse. The strength of the returns varies with the composition of the surface object reflecting the return. The LiDAR intensity values varies between 0 and 255.

Starting from LiDAR intensity the goal is to isolate vegetation that cannot be cut, such as tree trunks above a pre-defined value, and also other obstacles that cannot be cut, such as cars or walls.

In figure 3.18 it is possible to observe a new point cloud where the colour of the points is according to their intensity. In figure 3.19 it is possible to observe the scene in which the point cloud were captured. The region of interest of this point cloud was reduced to a window x[0;6], y[-3;3].

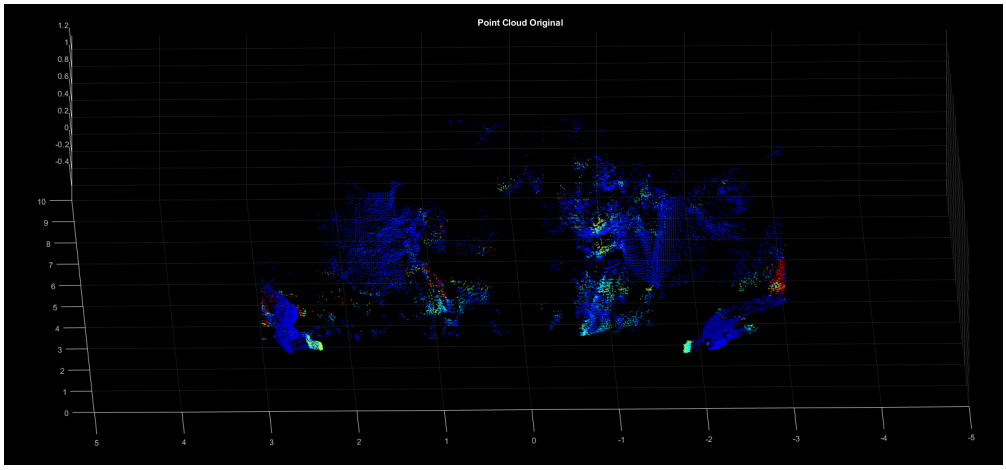


Figure 3.18: Point cloud represented by intensity.

Figure 3.19: Photos of the scene of the above point cloud.

As shown in figure 3.20 most of the points have intensity values between 0 and 45. Taking only the points within the range between 0 and 45 and dividing them into 9 point clouds with intensity ranges of only 5, it is possible to determine in which intervals are the points that are important for the objective, i.e. the detection of trunks.

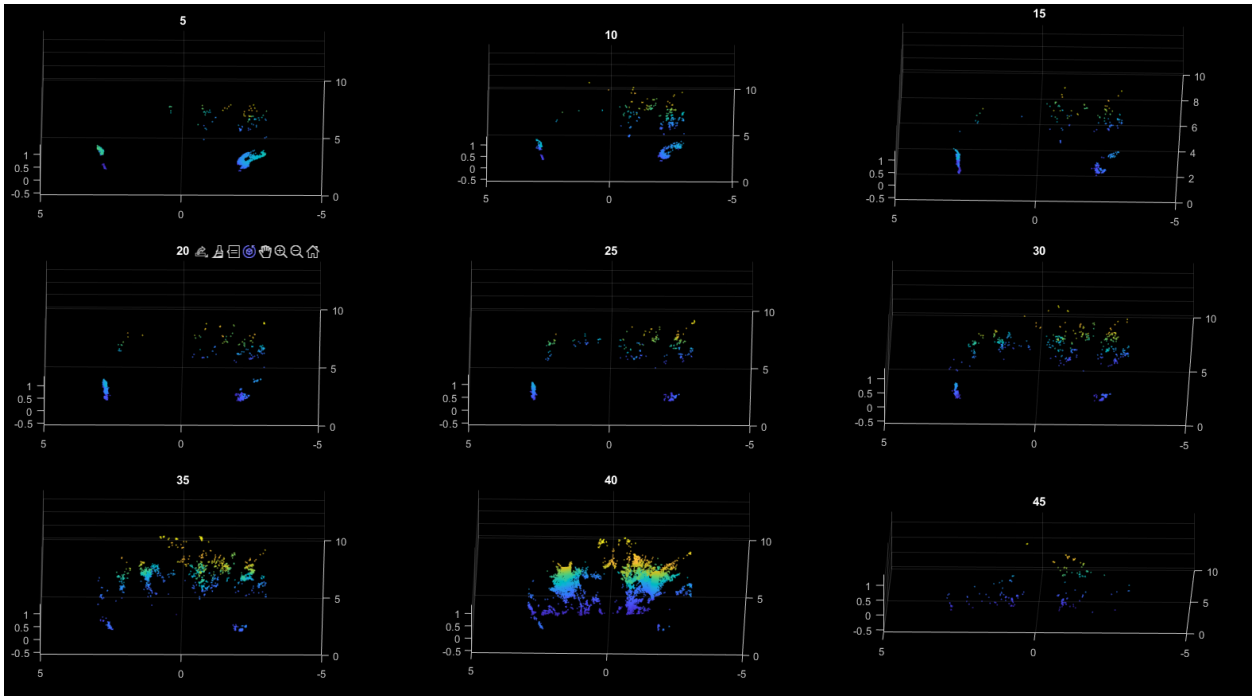


Figure 3.20: Point cloud split by intensity - in intervals of 5 in steps of 5.

From figure 3.20 it can be determined that the tree trunks are mainly in the range [35-40], allowing these points to be isolated. By dividing again this point cloud into others by intensity, i.e. 35, 36, ..., 40, it is possible to check in more detail the intensity that corresponds to the tree trunks. This permits to proceed to the next phase of the method, the clustering.

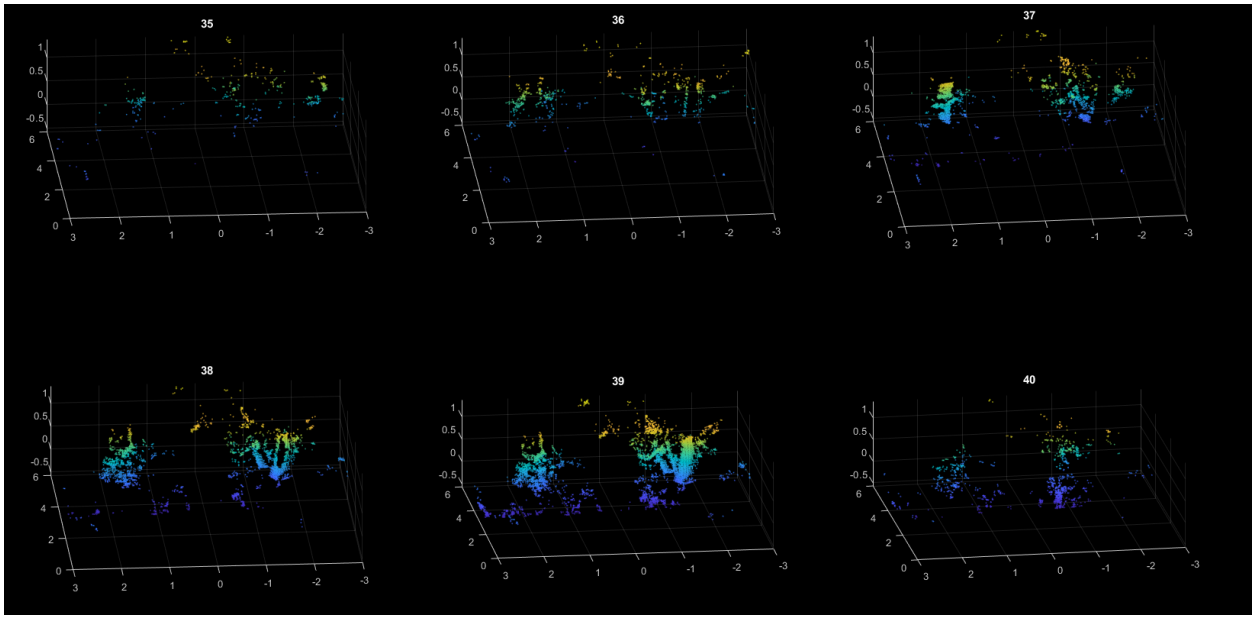


Figure 3.21: Point cloud of intensity [35-40] slited represented in steps of 1.

Clustering

Clustering or data mining is the set of data mining techniques that aim to automatically group data according to their degree of similarity. Each set of data resulting from the process is called a group or cluster. In this case a MATLAB algorithm based on Euclidean distance is used.

By applying the Matlab clustering algorithm to the point cloud with intensity between [37-39] with a Euclidean distance of 0.045 metres chosen through trial and error and taking into account the resolution of the LiDAR in the region of interest chosen, it is possible to obtain the following clusters represented in the figure 3.22 where each cluster is represented by a different colour. With this distance of 0.045 metres results 695 clusters, where only two have more than 500 points, i.e. many of the clusters are grasses, leaves, and other small vegetation.

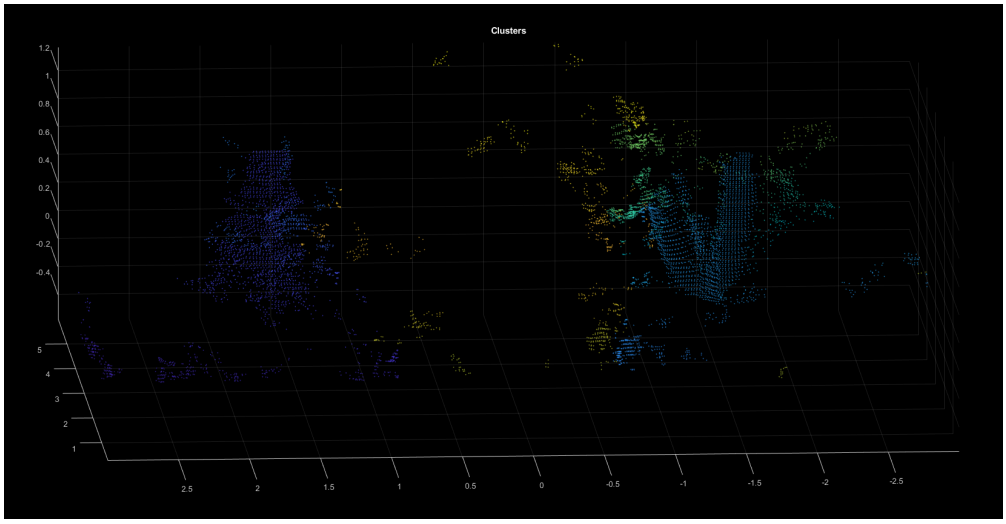


Figure 3.22: Representation of the point cloud corresponding to the interval [37-39]. The colour of the points is according to the cluster they belong to.

By setting the threshold at 500 points, all small clusters with fewer points will be removed. Having isolated the clusters of the trunks (clusters with more than 500 points), a cut is applied to the point cloud in z-coordinate around a pre-defined z (z-cut), in this case between 0.3 and 0.35 metres, relative to the LiDAR coordinate system (figure 3.24).

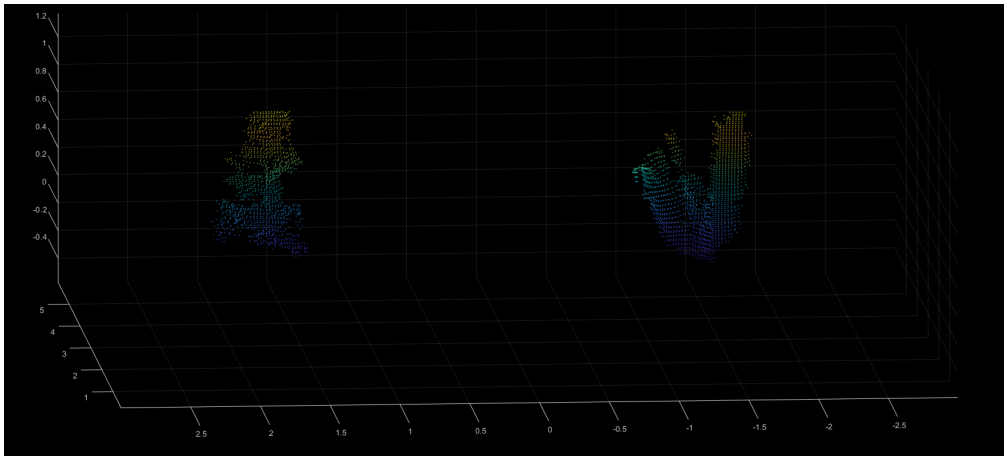


Figure 3.23: Representation of the point cloud corresponding to the tree trunk clusters after the clusters below the 500 points threshold are removed.

Using the point cloud resulting from the z-cut, the same clustering algorithm is again applied which separates the various trunks. In this clustering it is used a distance of 0.07 metres chosen taking into account the resolution of the LiDAR that at 6 metres is about 0.04m. After separating the trunks, an existing Matlab function **circlefit3d** is used. Given three points, the function returns a circle containing all of them. The function has as input 3 points of each cluster, the point with the smallest x and smallest y, the point with the largest x and largest y and the point with the smallest x and largest y.

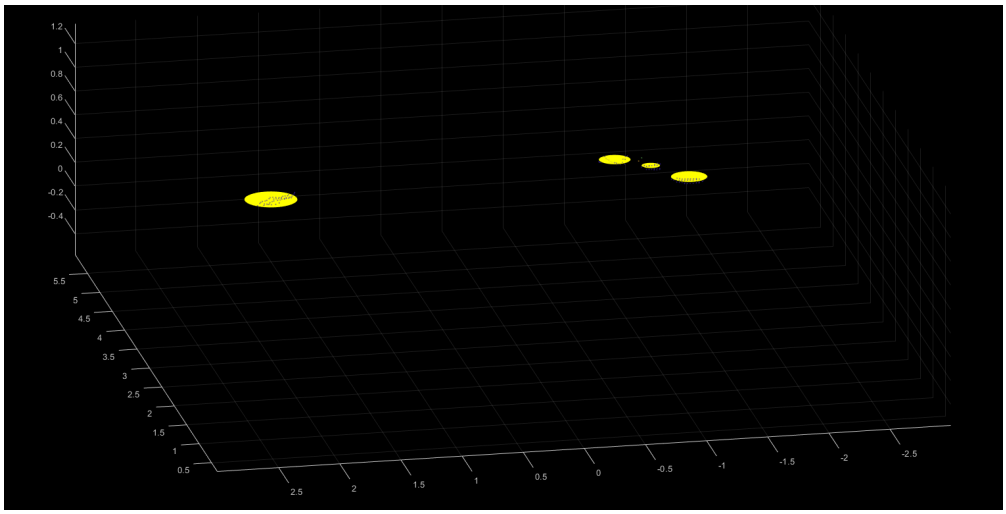


Figure 3.24: Representation of the point cloud corresponding to z-cut between 0.3 and 0.35 metres and the circle corresponding to the diameter of each trunk.

This method permits each trunk to be isolated and its location and diameter to be known.

Classifying other obstacles and Octomap

Using the return intensity of the laser pulse of LiDAR it was also possible to separate other obstacles, which in this case are cars or a car part (metal). To separate the other obstacles, the points with intensity in the range between [0-36] are used, i.e. those not used in the trunk detection (figure 3.25).

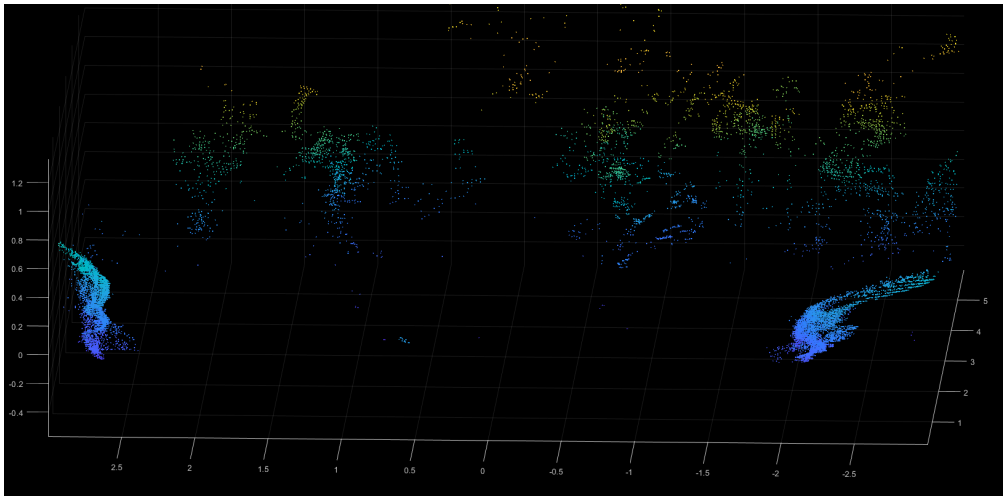


Figure 3.25: Point cloud whit intensity between [0 36] - intensity below that used previously.

Applying again the same clustering algorithm with a Euclidean distance of 0.045 meters chosen by trial and error and through LiDAR resolution the point cloud is divided into different clusters. With this distance of 0.045 metres results 1032 clusters, where only two have more than 500 points, i.e. as there are many clusters, most of them are grass and leaves where some points had a lower reflection due to the position or surface of the object.

Applying the threshold of 500 again all small clusters (below the threshold) were eliminated (see figure 3.26).

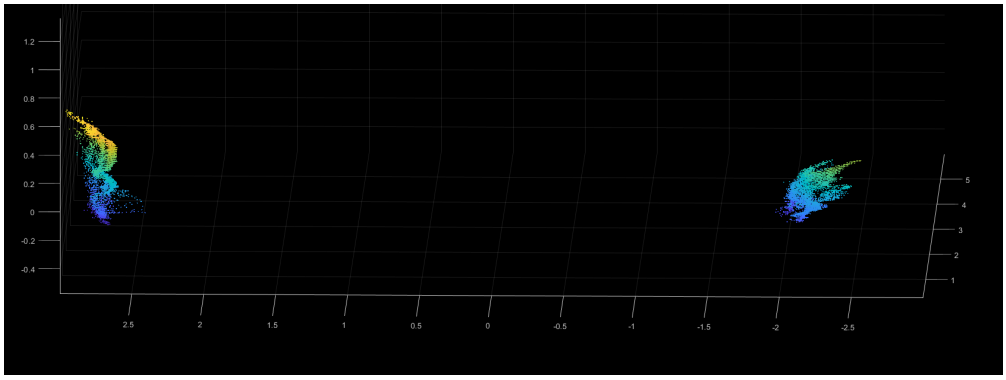


Figure 3.26: Point cloud corresponding to two parts of two cars, i.e. point cloud formed by the two clusters with 500 or more points.

Using again the Matlab function **circlefit3d**, an obstacle size can be obtained. It allows the obstacle to be considered even if it is not its true size (figure 3.27).

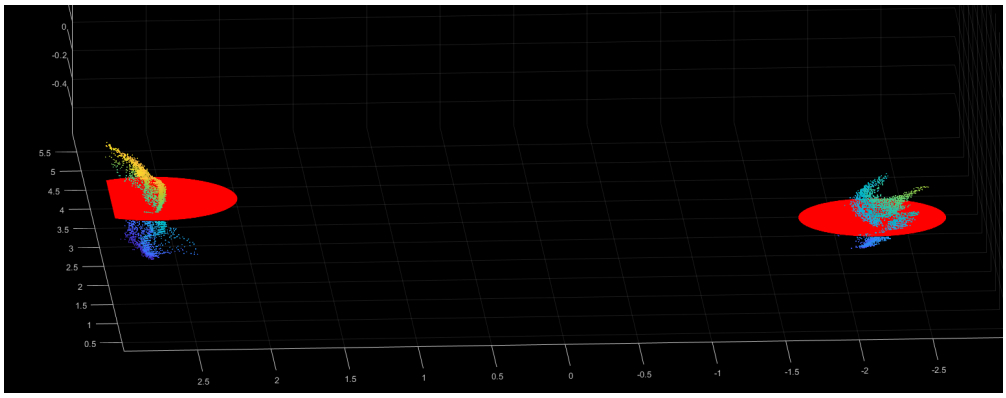


Figure 3.27: Point cloud corresponding to two parts of two cars (metal), i.e. obstacles that are not vegetation and the corresponding circles.

After the classification of the other obstacles, which in this case are cars or a car part (metal), it is possible to have the point cloud with the main obstacles classified as vegetation not to be cut i.e. trunks with a diameter above a pre-defined value according to the cleaning machine capabilities and the non-vegetation obstacles (figure 3.28).

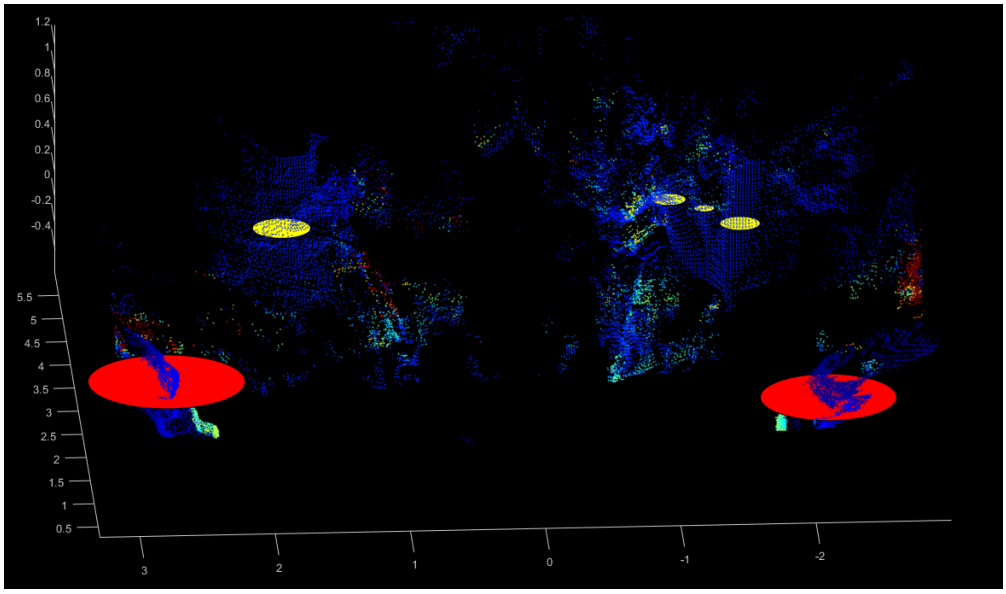


Figure 3.28: Point cloud with the main obstacles classified (red circles for non-vegetation obstacles and yellow circles for trunks).

An octomap (3D representation) of the main obstacles can then be created, with the position, size and classification as non-vegetation obstacles or vegetation that cannot be cut (figure 3.29).

In figure 3.29 it is possible to observe the original point cloud (green points) and the obstacles represented in octomap. With this result, the robot will have information about the environment to make the navigation, the size and positioning of each tree and also of each obstacle.

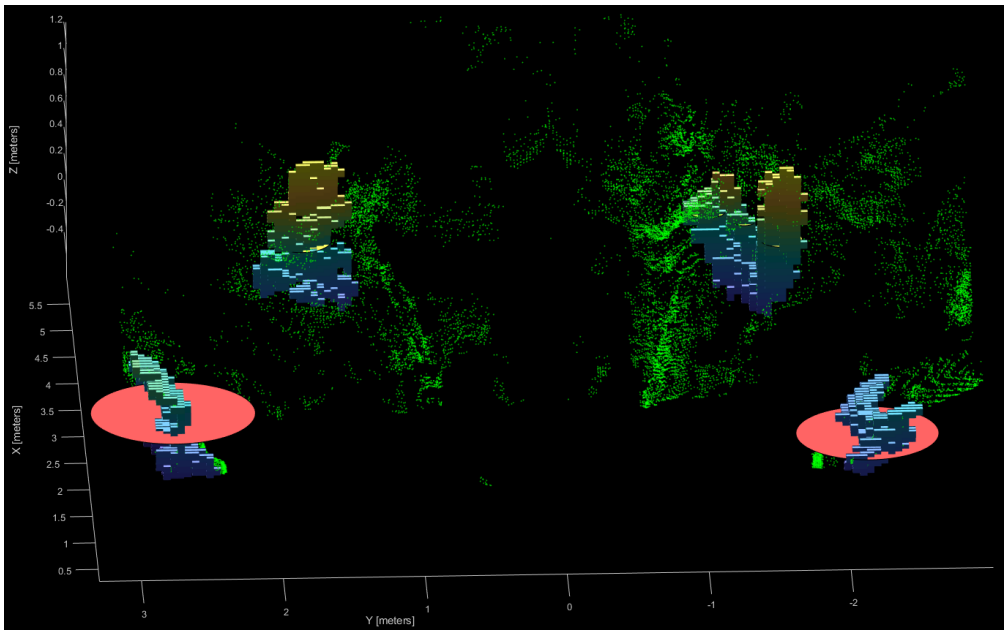


Figure 3.29: Obstacles represented in octomap and their classification (red circles for non-vegetation obstacles and yellow circles for trunks). Also represented the original point cloud (green points) for a better perception of the obstacles.

3.1.5 Classifying Objects by Size

The application of this method aims to classify large objects, such as trees, poles, walls and cars, with height well above the ground vegetation.

By making a z-coordinate cut around a pre-defined z above the ground vegetation in a point cloud, results in a horizontal slice of the point cloud, allowing a calculation of the contour of the objects. These contours will become separated from each other due to the slicing performed. As they are separated, the same clustering algorithm is applied to the point cloud after the z-cut, allowing the objects to be divided into clusters. On the resulting clusters, taking into account only the x and y coordinates, the diagonal will be calculated, i.e. the distance between the point with the smallest x (min(x)) and the point with the largest x (max(x)) and the distance between the point with the smallest y (min(y)) and the point with the largest y (max(y)). Using this diagonal length and a pre-defined threshold, if the object has a smaller diagonal than the threshold it is considered a trunk or a pole, if

the diagonal is larger than the threshold it is a car or a wall.

For a better understanding of the method implementation, it is used a point cloud obtained by registration of several point clouds by applying the ROS package **"hdl_graph_slam"** (figure 3.30, corresponding to the street of ISR-UC).

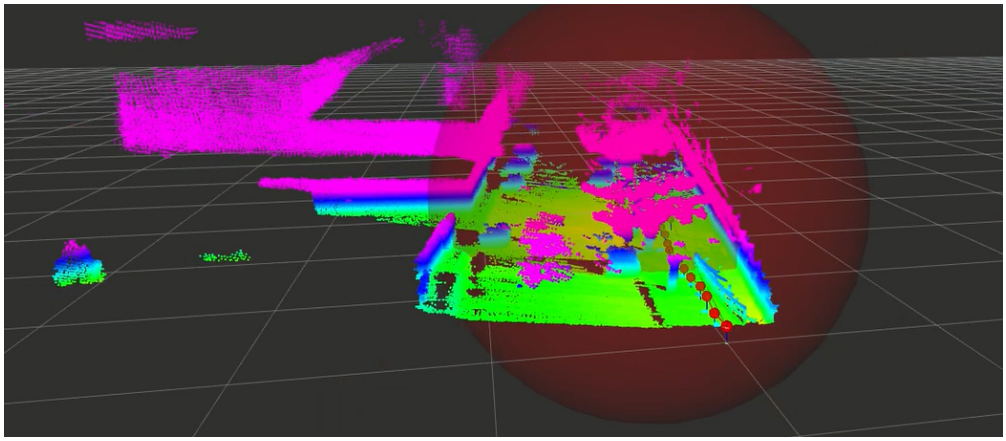


Figure 3.30: Point cloud resulting from the application of the ROS package **"hdl_graph_slam"**, and the trajectory of the LiDAR (small red balls).

Applying the cut around a pre-defined z [0.8; 1.1] (in meters) to the point cloud of figure 3.30, object contours can then be obtained, which corresponds to the point cloud in figure 3.31 (right). Then, applying the same clustering algorithm with a distance of 0.4 m all groups of points will be separated, because the objects after the cut are distant from each other.

Applying the calculation of diagonals of the clusters, the objects will then be classified. If the cluster has a diagonal smaller than the threshold is considered a trunk or a pole (green circles), if the distance is greater than the threshold is a car or a wall (red polygons). The green circles are created using the function already used before **circlefit3d**. The three points given to the function are: the closest point to the LiDAR origin (min(x)), the leftmost point of the LiDAR origin (min(y)) and the rightmost point of the LiDAR origin (max(y)). The red polygons are made using the Matlab function **fill3** which is given four points from each cluster: the point with the smallest y (min(y)), the point with the smallest x (min(x)), the point with the largest y (max(y)) and the point with the largest x (max(x)).

The classification using a threshold of 1 meter can be seen in figure 3.31.

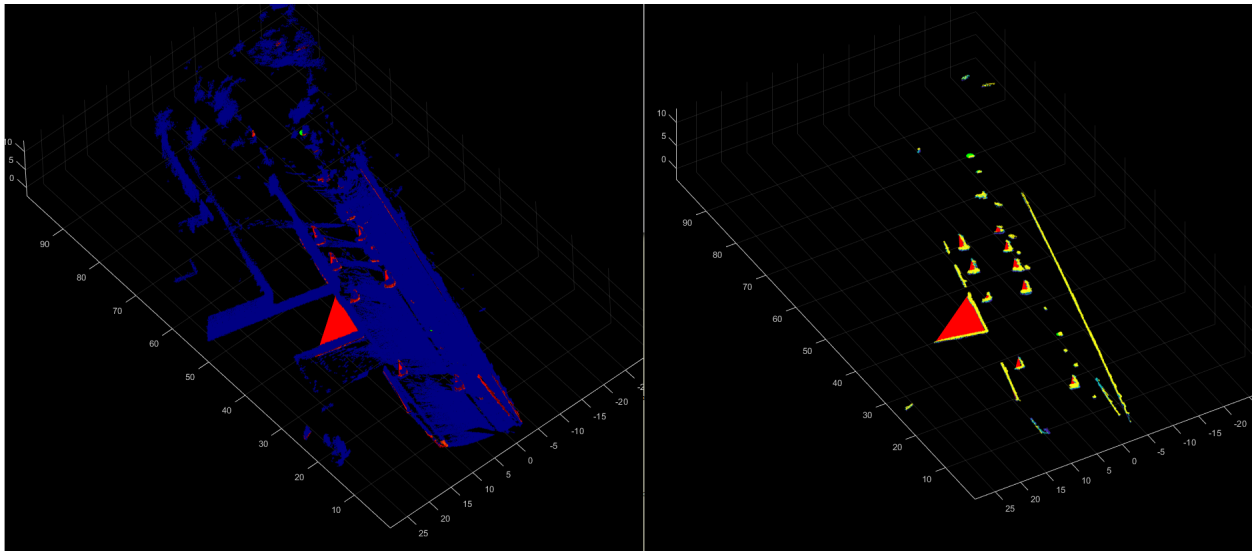


Figure 3.31: On the left, original point cloud (in blue) and classified objects. The classification is for trunks or poles (green circles) and larger objects like a car or a wall (red polygons). On the right, z-cut between [0.8;1.1] of the point cloud and classified objects.

In figure 3.32 it can be seen the point cloud cut around z between [0.8;1.1] and object classification, seen from another perspective for a better understanding.

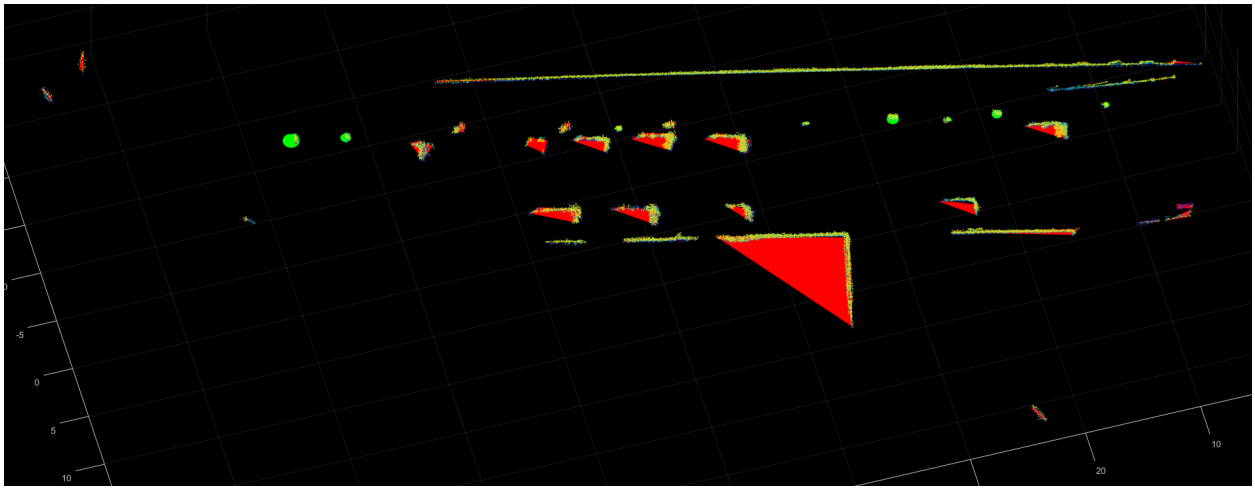


Figure 3.32: Z-cut between [0.8;1.1] of the point cloud and classified objects. The classification is trunks or poles (green circles) and objects like a car or a wall (red polygons).

3.2 RGB camera

In this section is presented the work developed using an RGB Camera, work related to vegetation indexes and segmentation of RGB images. The aim is to classify the vegetation present as vegetation cut by a clearing machine and vegetation yet to be cut.

3.2.1 Vegetation indexes

As seen in section 2.4, a vegetation index (also called a vegetative index) is a single number that quantifies vegetation biomass and/or plant vigor for each pixel in a remote sensing image. The index is computed using several spectral bands that are sensitive to plant biomass and vigor. One way to differentiate vegetation is through vegetation indexes using only the visible spectrum:

Table 3.2: RGB vegetation indexes

Index	**Definition**	**Reference**
R	red(0-255)	[52]
G	green(0-255)	[52]
B	blue(0-255)	[52]
r	R/(R+G+B)	[52]
g	G/(R+G+B)	[52]
b	B/(R+G+B)	[52]
ExG	2g-r-b	[53]
ExGR	ExG-(1.4r-g)	[53]
VEG	$g/(r^{0.667} \times b^{0.333})$	[54]
CIVE	0.441r-0.881g+0.385b+18.78745	[53]
COM	0.25ExG+0.3ExGR+0.33CIVE+0.12VEG	[54]
VARI	(G-R)/(G+R-B)	[53]
RGBVI	$(G^2 - RB)/(G^2 + RB)$	[55]
MGRVI	$(G^2 - R^2)/(G^2 + R^2)$	[55]
NGRDI	(G - R)/(G + R - B)	[54]
GLI	(2G-R-B)/(2G+R+B)	[54]
NDI	(G-R)/(G+R)	[54]
TGI	(G-0.39R) - 0.61B	[56]

In figures 3.33 and 3.34 a comparison between some RGB vegetation indexes was done. This comparison allows an evaluation of which index is most beneficial to distinguish the vegetation in the present scene. It also enables to verify that most of the indexes provide a very similar result in the scenes in question.

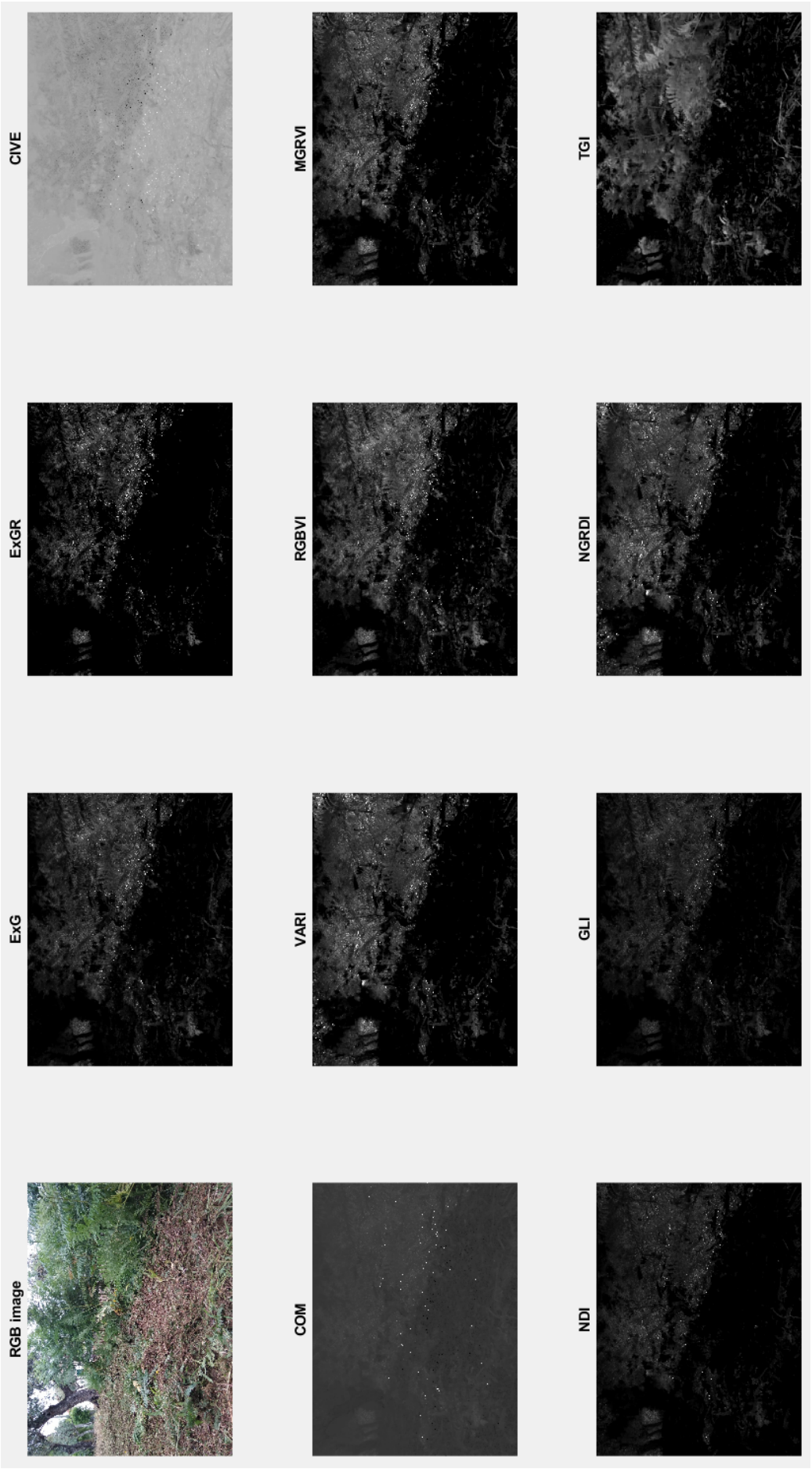


Figure 3.33: Comparison between different RGB vegetation indexes (scene 1).

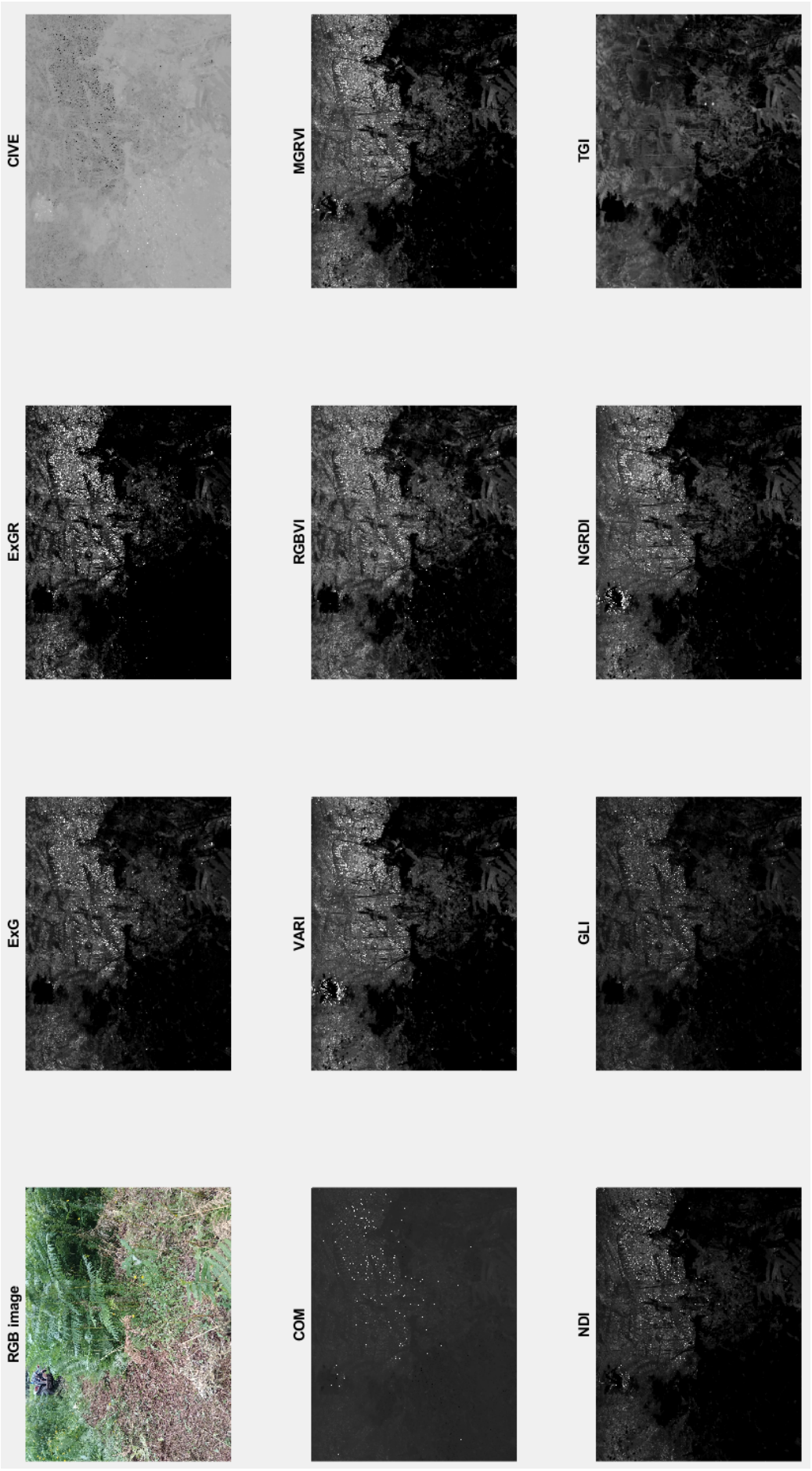


Figure 3.34: Comparison between different RGB vegetation indexes (scene 2).

Through the tests performed (e.g. figures 3.33 and 3.34) for the purpose of classifying vegetation as cut or uncut, the TGI index was the one that remained most congruent in various scenes tested.

3.2.2 Segmentation

Segmentation is commonly used to locate objects and boundaries (lines, curves, etc.) in images. Image segmentation is the process of assigning a label to each pixel in an image so that pixels with certain characteristics share the same label.

Factorization-based Texture Segmentation

In this section the **Factorization-based texture segmentation** algorithm is applied, which allows a segmentation of the RGB image by taking its textures.

As seen before, using local spectral histograms as features, a M $\times$ N feature matrix is constructed using M-dimensional feature vectors on an N-pixel image. Based on the observation that each feature can be approximated by a linear combination of several representative features, the factorisation of the feature matrix into two matrices is performed. One consists in the representative features, and the other contains the weights of the representative features at each pixel used for linear combination. The method uses local spectral histograms to discriminate regional appearances in a computationally efficient manner while accurately locating regional boundaries.

Parameters

The application of this algorithm has as input parameters **ws** and **segn**, namely, window size for computing features, and the number of parts in that image that will be segmented. Being the parameter **segn** the number of different segments desired, in the following example it will be set to 3 because the aim is to separate cut vegetation from uncut vegetation, and only the parameter **ws** will change in the experiment performed using the RGB image of figure 3.35.

As it can be seen in figure 3.36, the increase of the parameter **ws** will increase the size of the segments , which becomes more useful for the objective of separating the types of vegetation, because the segmentation obtained is better defined, that is, the segmented areas are larger and more continuous, eliminating noise.

Figure 3.35: RGB image used as input to the segmentation algorithm.

(a) ws=20 and segn=2. (b) ws=100 and segn=2. (c) ws=200 and segn=2.

Figure 3.36: Comparison of different values of **ws** parameter used in the factorization-based texture segmentation algorithm.

It can be seen that the resulting segmentation just by applying the segmentation algorithm to the RGB image, will separate just a bit of vegetation that is uncut on the left, and also not all the uncut vegetation on the right.

3.2.3 Fusion of RGB vegetation indexes with factorization-based texture segmentation

Using the RGB vegetation indexes and the segmentation algorithm and combining them together, it is possible to distinguish between vegetation that has been cut by a tractor using a clearing machine and vegetation that was not cut yet. For that, the vegetation index TGI is applied to the RGB image and then the factorization-based texture segmentation algorithm is applied with the parameter **ws=100** chosen after several tests in different scenes and the parameter **segn=2** it is intended to distinguish between cut and uncut vegetation. In the segmentation obtained the pixels will have a colour between black and white chosen by the

algorithm. The resulting segmentation will have two distinct parts, in black the already cut areas, and in grey, the uncut areas (figure 3.37c).

(a) Original image (input RGB indice).

(b) TGI, input for segmentation algorithm.

(c) Segmentation result (ws=100 and segn=2) using factorization-based texture segmentation.

Figure 3.37: Various stages of the fusion of the RGB indexes with the segmentation algorithm.

The fusion of the RGB vegetation indexes and the segmentation algorithm allows the robot to get a perception of the vegetation that has already been cut.

4 Presentation and Discussion of Other Results

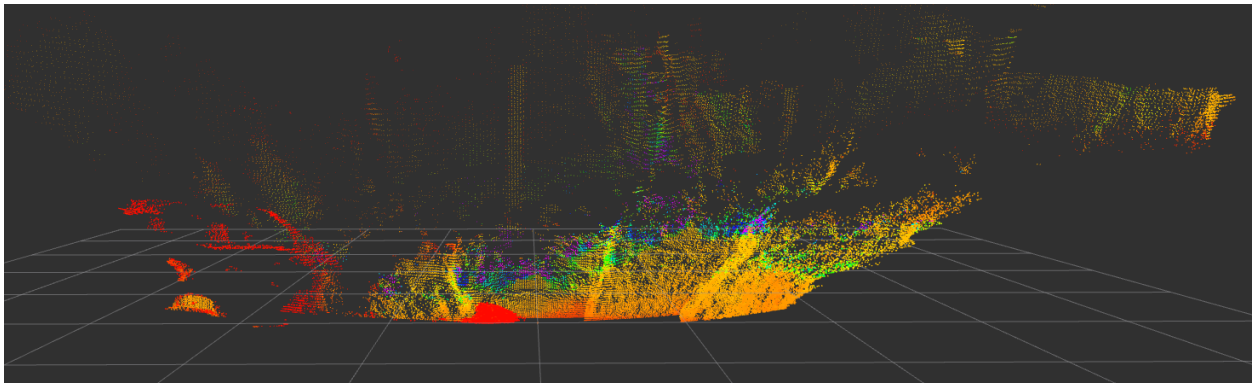
In this chapter a presentation and critical analysis of the results obtained using the methods described in chapter 3 is done. First, the methods using the LiDAR presented in section 3.1, and then, the methods using the RGB Camera presented in section 3.2.

4.1 Results

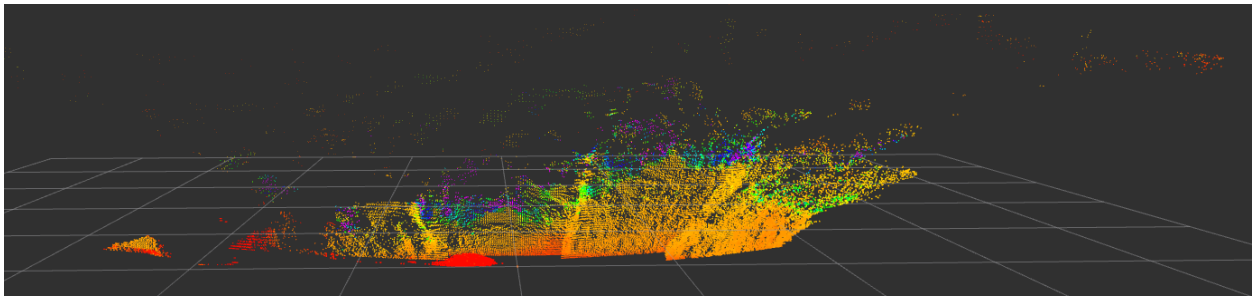
The work developed has two distinct starting points. In one of them, the input is a point cloud and in the other the input is an RGB image.

In figure 4.3 it is possible to see the developed methods in which the work developed with the LiDAR. In all methods, the original point cloud may not be filtered by the progressive morphological filter described in section 3.1.1 that splits the original point cloud into two, one containing the points corresponding to the ground and the other with the points corresponding to the objects. In figure 4.1 it is possible to see the scene where the new example data has been recorded. In figure 4.2 it can be seen the split of the original point cloud into the point cloud of the objects and the ground point cloud, the parameter slope=0.5 was used to obtain these results because the terrain is sloping.

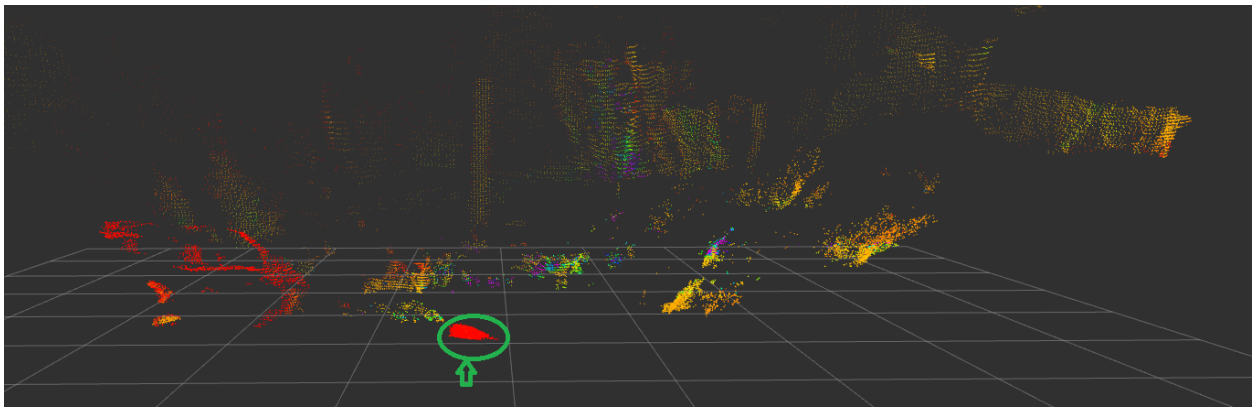
Figure 4.1: Photos of the new scene where the new example data was recorded.



(a) Original point cloud.



(b) Point cloud of ground.



(c) Point cloud of objects.

Figure 4.2: Results of the application of the progressive morphological filter with parameter **slope**=0.5.

Applying the Height Map method (presented in section 3.1.2) to the original point cloud (represented in 4.2a) two point clouds are obtained: one containing the points considered obstacles and the other containing the points considered non-obstacles. The Height Map method was applied with a resolution of 0.4 x 0.4 m and a height threshold of one meter. In figure 4.4a it is possible to see the two point clouds created. In green the objects considered not obstacles (e.g. ground vegetation) and in red the objects considered obstacles (e.g. the pole).

In figure 4.2 is marked with a circle and a green arrow a noise zone that LiDAR creates

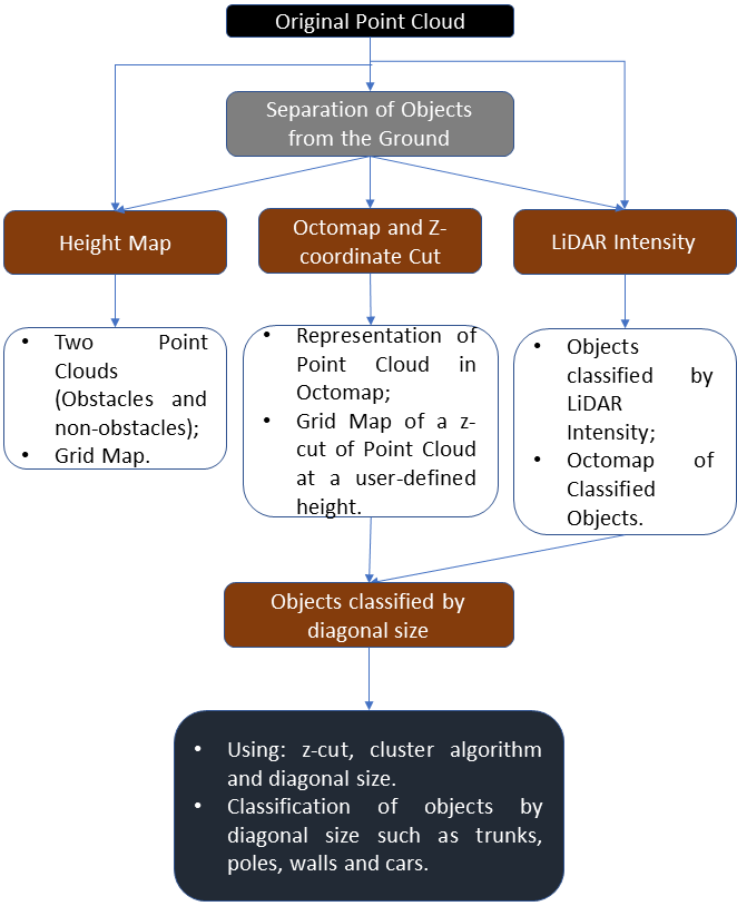
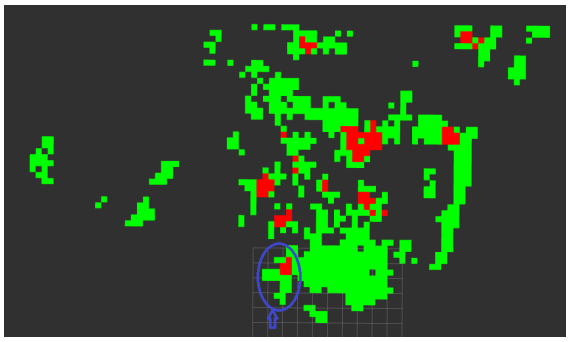
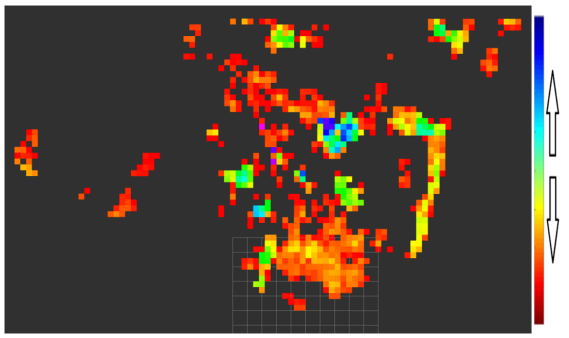


Figure 4.3: Framework of the methodology developed using LiDAR.

by default. Due to the noise, the points of the car present on the left of the figure are not recorded by LiDAR. So when applying the Height Map method only a part of the car was considered (see on the figure 4.4a the circle and the blue arrow).



(a) Overlapping point clouds, red for obstacles and green for the non-obstacles (top view).



(b) Height map of original point cloud (top view).

Figure 4.4: Results of the application of the Height Map.

In figure 4.5 it is possible to observe an application of the LiDAR Intensity method (pre-

sented in section 3.1.4) to a new point cloud (new scene), and the yellow circle corresponding to the diameter of the trunk. The detection of the trunk (yellow circle) was obtained using the same parameters used in the example described in section 3.1.4. The parameters were: region of interest x[0;6], y[-3;3]; z-cut between [0.3;0.35]; Euclidean distances for clustering were 0.045 and 0.07.

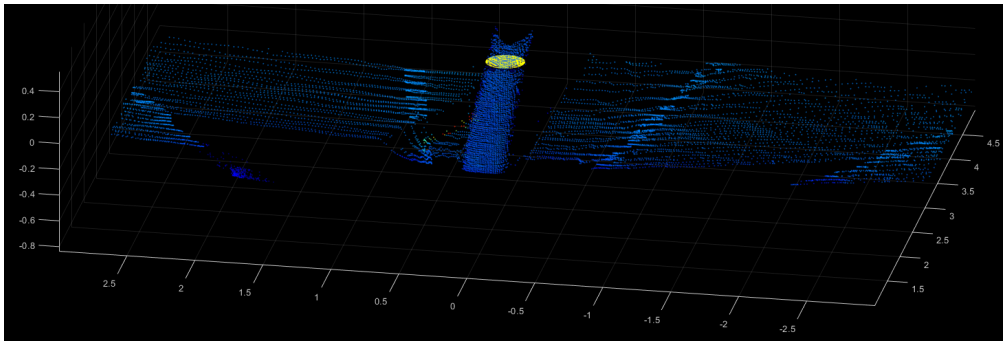
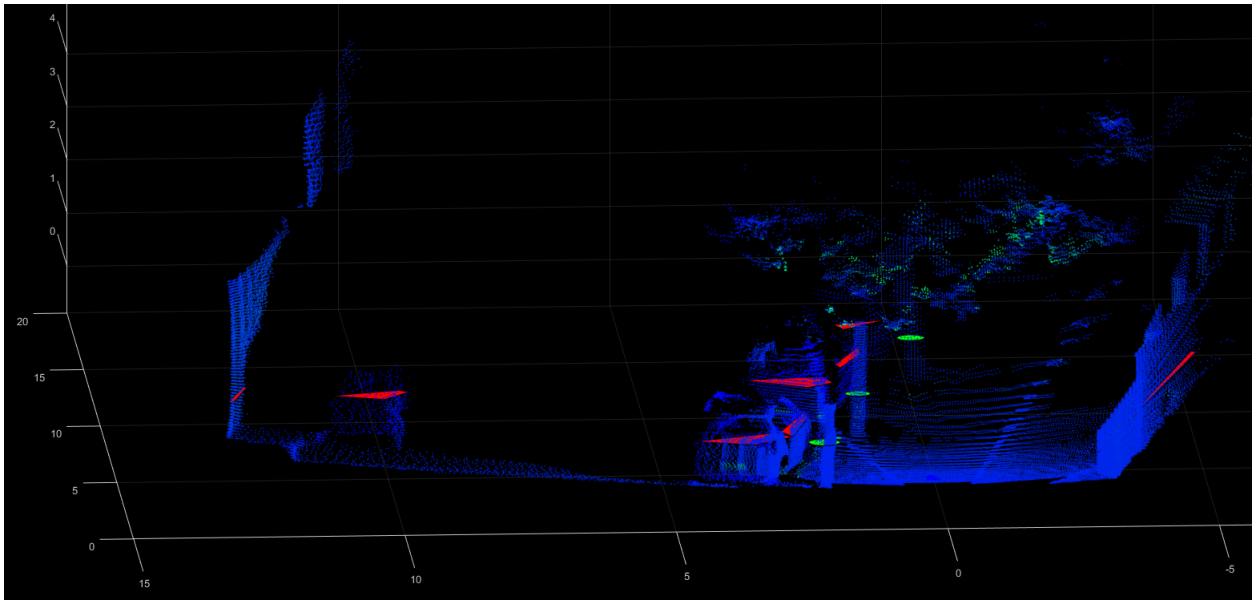


Figure 4.5: Original point cloud and the yellow circle corresponding to the diameter of the trunk.

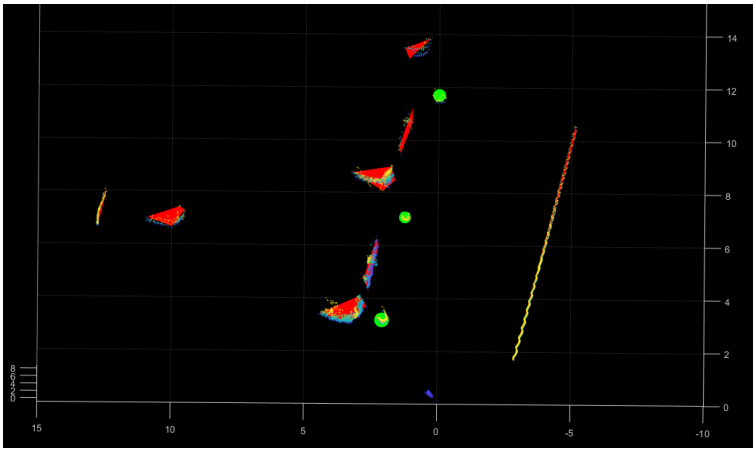
In figure 4.6a a new example of the application of the obstacle classification method by the size of the diagonal of each cluster (presented in section 3.1.5) can be observed. The parameters used were: a z-cut between 0 and 0.4 metres relative to the LiDAR origin (the LiDAR position was one metre high), the Euclidean distance used was 0.4 and the diagonal size threshold was one metre. The classification can be seen in the figure 4.6a and 4.6b (top view) where the green circles correspond to trunks and poles and the red polygons to cars or walls (size larger than the threshold).

In figure 4.7 it is possible to see the framework of the implemented methodology using an RGB image as a starting point. The aim of the described methodology is to classify cut and uncut vegetation.

In figure 4.8a a new scene can be observed where the method Fusion of RGB vegetation indexes with factorization-based texture segmentation (presented in section 3.2.3) was applied. The chosen vegetation index was the TGI and the parameters used in the segmentation were: segn=2 and ws=100. The segmentation result can be seen in figure 4.8b where black represents uncut vegetation and grey represents cut vegetation.



(a) Original point cloud, and classified objects. The classification is trunks or poles (green circles) and objects like a car or a wall (red polygons).



(b) Z-cut between [0.0;0.4] of the point cloud and classified objects (top view). The classification is trunks or poles (green circles) and objects like a car or a wall (red polygons).

Figure 4.6: Results of the application of the Classification of obstacles by diagonal size.

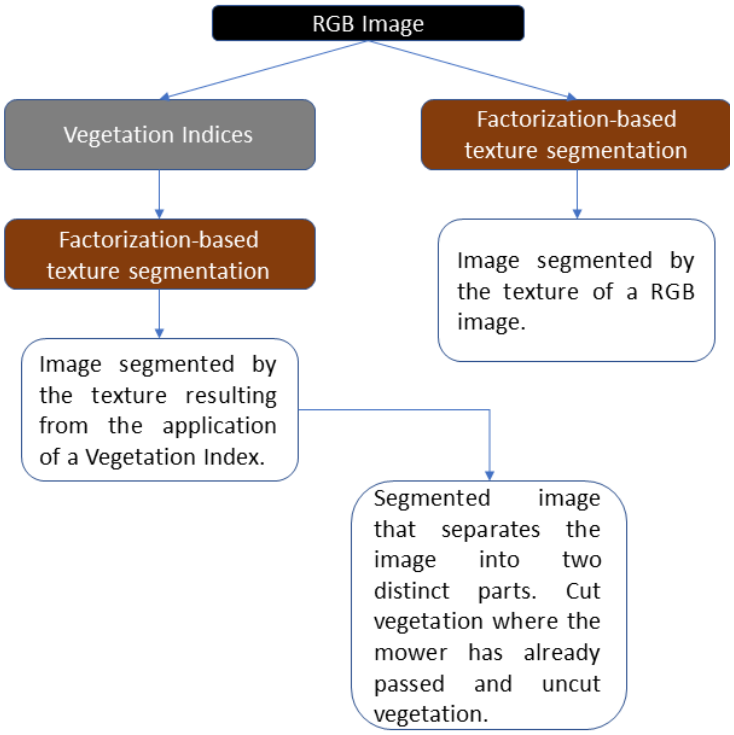


Figure 4.7: Framework of the methodology developed using RGB Camera.

(a) RGB image.

(b) Segmentation result.

Figure 4.8: Results of the application of the fusion of RGB vegetation indexes with factorization-based texture segmentation.

4.2 Discussion

In this section is done a critical analysis of the key points of each method implemented.

4.2.1 Applying the Progressive Morphological Filter

The application of the progressive morphological filter (presented in section 3.1.1) is dependent on an adequate choice of filter parameters. After several tests made in different scenes it was possible to adjust the parameters to values that allowed a acceptable separation of the points that belong to the ground from the points that belong to the objects.

The parameter **slope** is the parameter that creates the most difficulties. In all environments where LiDAR data was acquired, the terrain slope was always nearly zero. However, in the forest, the terrain brings other difficulties, such as many unevennesses and frequent changes of slope. The two ways of calculating the slope presented in section 3.1.1 had some weaknesses.

The first proposed algorithm use the comparison parameter **"areas where the difference between z-min and z-max is more than 20 centimetres"** to increase or decrease the slope value. The algorithm will be applied until the desired parameter are achieved, i.e., converge to one or zero. As the algorithm has to run until the comparison parameter converges to one or zero the computational time may increase depending on the input point cloud.

The second proposed algorithm to calculate the slope proposed in section 3.1.1 is based on LiDAR position. Based on the FoV of LiDAR, it is possible to know the z-min at a certain distance from LiDAR. Thus, knowing the height at which the LiDAR is from the ground and the z-min at which the LiDAR intersects the ground, and averaging the z-min at that distance, it is possible to obtain the terrain slope through the difference between the expected and the obtained value. In the forest, the amount of soil slope and dense vegetation makes it difficult to obtain a slope value.

4.2.2 Point Cloud - Z Coordinate Cut

When the terrain is parallel to the LiDAR XY plane, there is no problem with the cut, however, the unevenness of the terrain in the forest makes it necessary to pay attention to how the cut is made. If the cut is made in a region of interest close to the robot, the impact of the slope of the terrain is smaller. That is, the impact of the slope on the point cloud cut

increases as the point cloud increases.

4.2.3 LiDAR Intensity

Intensity is a value collected for each point of the return of the laser pulse that generated the point. It is based, in part, on the reflectivity of the object that the laser pulse hit.

Thus, the reflectivity of the laser is dependent on the surface of each object present. In section 3.1.4 two different surfaces were tested, tree trunks and cars (metal). As reflectivity varies with surface, other tree species with different trunk bark may not be in the same intensity range.

Once the trunks have been isolated, it is then possible to calculate the diameter of each trunk. Since the diameter is calculated by performing a z-cut to the cluster of the trunk, a deformation of the trunk which coincides with the cut can lead to an error in the calculated diameter. If the trunk has a reduced real diameter the error in percentage becomes larger.

The diameter of a tree decreases with the height at which the measurement is made i.e. if the measurement is taken close to the ground the diameter is larger than if the measurement is taken one metre above the ground. For the error to be zero, the Z-cut and the actual measurement would have to be taken in exactly the same place.

On the other hand, the data acquisition would have to be done in such a way that the XY LiDAR plane is parallel to the XY plane of the tree.

In the table 4.1 it is possible to observe the comparison between the real value and the calculated value of the diameter. It can be seen that when the trunk is smaller the error becomes greater, as seen in the fourth row of the table.

The second row of table 4.1 corresponds to the left trunk of figure 3.24, which is a trunk that contains many branches parallel to its trunk, i.e. the measurement taken with LiDAR will catch the parallel branches. This justifies justify the disparity between the real value and the calculated value.

The LiDAR Intensity method allows us to have a good perception of the size of the trunks existing in the navigation environment where the robot is, thus being able to decide which ones can be cut by the robot. This method can also be used to make inventories of tree plantations, allowing to verify the growth of trees by measuring their diameter.

Table 4.1: Comparison of the real value with the calculated value of the diameter by the LiDAR Intensity method.

Real Diameter (m)	**Calculated Diameter**(m)	**Error**
0.290	0.307	$\%error = \frac{\|0.307-0.290\|}{0.289} * 100 = 6\%$
0.318	0.291	$\%error = \frac{\|0.291-0.318\|}{0.318} * 100 = 8\%$
0.178	0.241	$\%error = \frac{\|0.241-0.178\|}{0.178} * 100 = 35\%$
0.280	0.430	$\%error = \frac{\|0.430-0.280\|}{0.280} * 100 = 53\%$
0.063	0.153	$\%error = \frac{\|0.153-0.063\|}{0.063} * 100 = 62\%$

4.2.4 Object Classification by Diagonal Size

With application of the method classifying objects by size it is possible to verify the good results obtained, however the algorithm again depends on the z-cut. For example, if a car is behind a tree, the algorithm will classify the car as two cars.

Another key point of the algorithm is the distance used in the clustering algorithm because it is necessary that the objects are well separated for the classification to be correct. As the clustering algorithm is applied after the z-cut, the choice of the Euclidean distance has to be made in such a way that all resulting clusters of points do not lose points and are well clustered. A distance of for example 0.4 m ensures that all points are well grouped, however if a tree is less than 0.4 m from a car or another tree the two groups of points will be grouped together into one. However, the goal of classifying the obstacles that the robot cannot cut through will not be affected, as the obstacles will not be considered anymore but will be considered as a bigger obstacle.

Likewise, at the end, it is possible to obtain the size of the obstacles and their classification, allowing the robot to navigate with information about size and type of obstacles.

4.2.5 Segmentation using RGB Vegetation Indexes

The need to understand if the vegetation has already been cut or not yet cut by the robot is very important for the navigation of the robot.

The application of the vegetation indexes presented good results because after a clearing machine cut the vegetation, what was left has a very reduced volume. It can be considered ground, so the RGB image will contain two distinct areas, vegetation and ground.

The tests were based on the choice of the index that best represents the uncut areas.

After several tests, the index that showed the best coherence was the TGI.

The method showed good results in all the scenes used, however the vegetation present was always in its green state, as it was cut between winter and spring. It was not possible to perform tests where the vegetation was already completely dry.

The fusion of this segmentation method with the object classification methods (LiDAR intendity method or the method classifying objects by size) will allow the robot to have a perception of the main characteristics of the environment. The robot will have information about the objects it cannot cut like trees, poles, cars or walls and it will have information about the state of the vegetation i.e. if the vegetation is cut or if the vegetation is not cut.

5 Conclusion

In this dissertation we proposed to develop a perception system for Unmanned Ground Vehicle (UGV) systems using LiDAR and also aa RGB camera.

As a starting point, we began by applying a progressive morphological filter so that the point cloud could be separated in two, i.e., objects and ground. For this, algorithms were proposed that allowed the parameter **slope** to be automatically calculated according to the slope of the terrain.

Using the Height Map method, it is possible to separate the objects of the point cloud, considering as obstacles those that are bigger than the height threshold pre-defined by the robot user, according to the environment. For example, in the middle of a pine tree forest, the definition of the threshold with 1.5 meters would consider all the adult pine trees as obstacles, thus allowing the navigation without cutting the adult pine trees.

The Octomap and Z-coordinate cut method starts from a pre-defined variable that is the height at which the z-cut range in the point cloud is made. This cut allows a grid map to be created containing all object contours at the defined height. For example, in a forest if z-cutting is done at a height of two metres only adult trees will be cut. It also allows the representation in an octomap of the environment where the robot is.

Two methods were also developed for obstacle classification, one based on LiDAR Intensity and the other on the diagonal size of each cluster. The first method, based on LiDAR Intensity, classifies obstacles according to the reflectivity of their surface, which allowed classifying the obstacles present in the environment as cars (metal) or trunks. It also allows calculating the diameter of the trunks. The results obtained had a percentage error around 10% for trees with diameters close to thirty centimetres, which is an acceptable error for the objective of detecting trees not to be cut. When the trunk is small the percentage error is high, but again for the purpose of detecting trees not to be cut, the tree was detected but with a larger diameter.

The second method allows the classification of obstacles by their diagonal size, in the

case used two ranges of sizes were defined. The classification showed interesting results as in the chosen region of interest all obstacles were well classified.

From an RGB image, a method was developed that classifies the vegetation as cut and uncut. With RGB image is calculated the TGI Vegetation index and then the texture-based segmentation algorithm is applied. This segmentation into two distinct areas allows to verify the vegetation cut by the cleaning machine and the vegetation that was not cut.

5.1 Future Work

This section describes future work that may be done after this Dissertation.

5.1.1 Automatic cutting at the Point Cloud

In the implemented methods, the decision of which range (z-cut) to cut was manual, that is, according to the scene or the obstacles present, a range to cut was chosen in a z range of the point cloud. Since the main obstacles that cannot be cut by the robot in the forest are trees, the height can be previously set. However, the terrain in front of the robot does not always have the same slope.

The goal is to create an algorithm that knowing the slope of the terrain converts that slope into a plane of the type $ax + by + cz + d = 0$ that is parallel to the ground. The height of the plane is pre-defined, i.e., if at the beginning the plane is pre-defined at a height of 1.5 meters, during the execution of the plane, the updated plane will have to be 1.5 meters from the ground.

One possibility is to use an Inertial Measurement Unit (IMU) and make adjustments to the plane defined for the cut according to the slope at which the robot is located.

5.1.2 LiDAR-Camera Fusion

Each sensor provides different data in order to complement each other.

For example, cameras provide rich colour and feature information that can be used by algorithms to detect objects of interest (pedestrians, cars, trees, etc.). LiDARs can provide rich structural information and if it can be matched with the camera, when a pedestrian is detected in an image, its exact 3D location can be estimated and be used by an autonomous car to avoid obstacles and prevent accidents.

Multiple sensors are used to provide redundant information, which reduces the possibility of having erroneous measurements. In the above cases, it is essential to obtain data from multiple sensors with a single reference frame so that the data can be merged and redundancy can be exploited.

In [57] an accurate and repeatable method is proposed to estimate extrinsic calibration parameters in the form of 6 degrees of freedom between a camera and a LiDAR. The algorithm described uses tags that can be easily printed and stuck on planar surfaces such as card boards or wooden planks. A point extraction pipeline is implemented to obtain corner points of the card boards from the point cloud recorded using LiDAR. The two sets of point correspondences are used to solve for the [R|t], extrinsic parameters.

The method described in [57] is based on a different LiDAR so it would have to be converted to our LiDAR.

The ultimate goal is to produce and continuously update a map for robot navigation during the clean-up mission.

6 Bibliography

[1] Yoshiaki Ichikawa and Norihoko Ozaki. *Autonomous Mobile Robot.*, volume 2. 1985.

[2] MATLAB & Simulink. Stereo Vision . https://www.mathworks.com/discovery/stereo-vision.html?s_tid=srchtitle.

[3] Jiangye Yuan, Deliang Wang, and Anil M. Cheriyadat. Factorization-Based Texture Segmentation. *IEEE Transactions on Image Processing*, 24(11):3488–3497, 2015.

[4] Technical Manual. SEMFIRE Project. 2020.

[5] Sajad Saeedi, Michael Trentini, Howard Li, and Mae Seto. Multiple-robot Simultaneous Localization and Mapping - A Review 1 Introduction 2 Simultaneous Localization and Mapping : problem statement. *Journal of Field Robotics*, 33(1):3–46, 2016.

[6] PORDATA. Densidade populacional. `https://www.pordata.pt/Municipios/Densidade+populacional-452`, 2020.

[7] John G. Mooney and Eric N. Johnson. A Comparison of Automatic Nap-of-the-earth Guidance Strategies for Helicopters. *Journal of Field Robotics*, 33(1):1–17, 2014.

[8] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, pages 1–18, 2013.

[9] Radu Bogdan Rusu. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Kunstliche Intelligenz*, 24(4):345–348, 2010.

[10] Yaru Xian, Jun Xiao, Ying Wang, Mengyi Shan, and Chong Zhou. A Review of Fine Registration for 3D Point Clouds. (19):108–113, 2016.

[11] K. Kraus and N. Pfeifer. Determination of terrain models in wooded areas with airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 53(4):193–203, 1998.

[12] N Pfeifer. Derivation Of Digital Terrain Models In The Scop++ Environment. *OEEPE Workshop on Airborne Laserscanning and Interferometric SAR for Digital Elevation Models*, page 13, 2001.

[13] George Vosselman. Slope based filtering of laser altimetry data. *International Archives of Photogrammetry and Remote Sensing, Vol. 33, Part B3/2*, 33(Part B3/2):678–684, 2000.

[14] Somerville. Some Algorithms For Virtual Deforestation (VDF) OF Lidar Topographic Survey Data. *Journal of the American Chemical Society*, 123(10):2176–2181, 2001.

[15] Peter Lohmann, Andreas Koch, and Michael Schaeffer. Approaches to the filtering of laser scanner data. *International Archives of Photogrammetry and Remote Sensing*, 33, 2000.

[16] Johannes Kilian, Norbert Haala, and Markus Englich. Capture Andevaluation of Airborne Laser Scanner Data. *Commission III*, 2001.

[17] Keqi Zhang, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne lidar data. *Geoscience and Remote Sensing, IEEE Transactions on*, 41:872 – 882, 2003.

[18] Robert M. Haralick, Stanley R. Sternberg, and Xinhua Zhuang. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(4):532–550, 1987.

[19] Wolfgang Eckstein and Olaf Muenkelt. Extracting objects from digital terrain models. *Remote Sensing and Reconstruction for Three-Dimensional Objects and Scenes*, 2572:43–51, 1995.

[20] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[21] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[22] David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. *Proceedings of the National Conference on Artificial Intelligence*, 2:979–984, 1994.

[23] T. Durrant-Whyte, H.; Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13, 2006.

[24] Erik Zamora Gómez. Map-building and planning for autonomous navigation of a mobile robot. *PHD-Thesis*, pages 1–175, 2015.

[25] ROS Wiki. velodyne_height_map. http://wiki.ros.org/velodyne_height_map.

[26] CameraNeon. Distorção radial de lentes na fotografia . http://cameraneon.com/tecnicas/distorcao-radial-em-fotografias/#2.

[27] E. Raymond Hunt, W. Dean Hively, Stephen J. Fujikawa, David S. Linden, Craig S.T. Daughtry, and Greg W. McCarty. Acquisition of NIR-green-blue digital photographs from unmanned aircraft for crop monitoring. *Remote Sensing*, 2(1):290–305, 2010.

[28] Anatoly A. Gitelson and Mark N. Merzlyak. Signature analysis of leaf reflectance spectra: Algorithm development for remote sensing of chlorophyll. *Journal of Plant Physiology*, 148(3-4):494–500, 1996.

[29] Sebastian Candiago, Fabio Remondino, Michaela De Giglio, Marco Dubbini, and Mario Gattelli. Evaluating multispectral images and vegetation indices for precision farming applications from UAV images. *Remote Sensing*, 7(4):4026–4047, 2015.

[30] Yiannis Ampatzidis, Victor Partel, Bo Meyering, and Ute Albrecht. Citrus rootstock evaluation utilizing UAV-based remote sensing and artificial intelligence. *Computers and Electronics in Agriculture*, 164:104900, 2019.

[31] Jaafar Abdulridha, Ozgur Batuman, and Yiannis Ampatzidis. UAV-based remote sensing technique to detect citrus canker disease utilizing hyperspectral imaging and machine learning. *Remote Sensing*, 11(11), 2019.

[32] Gilles Rabatel, Nathalie Gorretta, and Sylvain Labbé. Getting NDVI spectral bands from a single standard RGB digital camera: A methodological approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7023 LNAI:333–342, 2011.

[33] Lucas Costa, Leon Nunes, and Yiannis Ampatzidis. A new visible band index (vNDVI) for estimating NDVI values on RGB images utilizing genetic algorithms. *Computers and Electronics in Agriculture*, 172:105334, 2020.

[34] Kohei Arai, Kenji Gondoh, Osamu Shigetomi, and Yuko. Method for NIR Reflectance Estimation with Visible Camera Data based on Regression for NDVI Estimation and its Application for Insect Damage Detection of Rice Paddy Fields. *International Journal of Advanced Research in Artificial Intelligence*, 5(11):17–22, 2016.

[35] Y Kaufman and D Tanre. Atmospherically resistant vegetation index. *IEEE Transactions on Geoscience and Remote Sensing*, 30(2):260–271, 1992.

[36] Anatoly A Gitelson, Robert Stark, Don Rundquist, Anatoly A Gitelson, Yoram J Kaufman, Robert Stark, and Don Rundquist. DigitalCommons @ University of Nebraska - Lincoln Novel Algorithms for Remote Estimation of Vegetation Fraction. 2002.

[37] E. Jr, Paul Doraiswamy, James McMurtrey, Craig Daughtry, Eileen Perry, and Bakhyt Akhmedov. A visible band index for remote sensing leaf chlorophyll content at the canopy scale. *International Journal of Applied Earth Observation and Geoinformation*, 21:103–112, 04 2013.

[38] RH Haas J.W. Rouse, R.W. Haas, J.A. Schell, D.W. Deering, J.C. Harlan, JW ROUSE, RH Hass, JA Schell, DW Deering, JC Harlan, JW Rouse, J. Rouse. Monitoring the Vernal Advancement and Retrogradation (Greenwave Effect) of Natural Vegetation. *NASA/GSFCT Type III Final Report*, 1974.

[39] M. Crosier and L. D. Griffin. Using basic image features for texture classification. *International Journal of Computer Vision*, 88(3):447–460, 2010.

[40] Li Liu and Paul Fieguth. Texture classification from random features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):574–586, 2012.

[41] Paragios Nikos and Deriche Rachid. Geodesic active regions and level set methods for supervised texture segmentation. *International Journal of Computer Vision*, 46(3):223–247, 2002.

[42] Kaan Ersahin, Ian G. Cumming, and Rabab K. Ward. Segmentation of polarimetric SAR data using contour information via spectral graph partitioning. *International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 2240–2243, 2007.

[43] Xiuwen Liu and De Liang Wang. A spectral histogram model for texton modeling and texture discrimination. *Vision Research*, 42(23):2617–2634, 2002.

[44] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, volume 148. Fourth edi edition.

[45] Ji Zhang and Sanjiv Singh. Loam : Lidar odometry and mapping in real-time. *Robotics: Science and Systems Conference (RSS)*, pages 109–111, 01 2014.

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

[47] Mathieu Labbé and François Michaud. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[48] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. *Proceedings - IEEE International Conference on Robotics and Automation*, 3(c):1691–1696, 2012.

[49] Mark A Post, Alessandro Bianco, and Xiu T Yan. Autonomous Navigation with ROS for a Mobile Robot in Agricultural Fields. 2017.

[50] Kenji Koide. GitHub - hdl_graph_slam: 3D LIDAR-based Graph SLAM.

[51] RoboSense. RoboSense LiDAR - Autonomous Driving, Robots, V2R. https://www.robosense.ai/en/RS-LiDAR-M1.

[52] Shigeto Kawashima and Makoto Nakatani. An algorithm for estimating chlorophyll content in leaves using a video camera. *Annals of Botany*, 81(1):49–54, 1998.

[53] Luis Fernando Sánchez-Sastre, Nuno M. S. Alte da Veiga, Norlan Miguel Ruiz-Potosme, Paula Carrión-Prieto, José Luis Marcos-Robles, Luis Manuel Navas-Gracia, and Pablo Martín-Ramos. Assessment of RGB Vegetation Indices to Estimate Chlorophyll Content in Sugar Beet Leaves in the Final Cultivation Stage. *AgriEngineering*, 2(1):128–149, 2020.

[54] Jadson Freire-Silva, Yenê Medeiros Paz, Pedro Paulo Lima-Silva, João Antonio dos Santos Pereira, and Ana Lúcia Bezerra Candeias. Índices de vegetação do Sensoriamento Remoto para processamento de imagens na faixa do visível (RGB). *Journal of Hyperspectral Remote Sensing*, 9(4):228, 2019.

[55] B. D.S. Barbosa, G. A.S. Ferraz, L. M. Gonçalves, D. B. Marin, D. T. Maciel, P. F.P. Ferraz, and G. Rossi. RGB vegetation indices applied to grass monitoring: A qualitative analysis. *Agronomy Research*, 17(2):349–357, 2019.

[56] F. Fuentes-Peailillo, S. Ortega-Farias, M. Rivera, M. Bardeen, and M. Moreno. Comparison of vegetation indices acquired from RGB and Multispectral sensors placed on UAV. *IEEE ICA-ACCA 2018 - IEEE International Conference on Automation/23rd Congress of the Chilean Association of Automatic Control: Towards an Industry 4.0 - Proceedings*, 2019.

[57] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna. LiDAR-Camera Calibration using 3D-3D Point correspondences. *ArXiv e-prints*, May 2017.