UNIVERSIDADE D
COIMBRA

André de Sousa Nunes Teixeira

LiDAR-based Topological Mapping of Orchard Environments

Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, do ramo de especialização em Robótica, Controlo e Inteligência Artificial, orientada pelo Professor Doutor Lino José Forte Marques e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

LiDAR-based Topological Mapping of Orchard Environments

André de Sousa Nunes Teixeira

Coimbra, September 2022

UNIVERSIDADE D
COIMBRA

LiDAR-based Topological Mapping of Orchard Environments

Supervisor:

Prof. Doutor Lino José Forte Marques

Co-Supervisor:

Doutor Sedat Dogru

Jury:

Prof. Doutor António Paulo Mendes Breda Dias Coimbra

Prof. Doutor Cristiano Premebida

Prof. Doutor Lino José Forte Marques

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

Coimbra, September 2022

Acknowledgements

I would like to express my gratitude to everyone in my life who has helped make this dissertation possible in some way.

I would like to thank my supervisor, Professor Lino Marques, for always pointing me in the correct direction, for the transmitted knowledge, for all of the critical recommendations, and for providing me the opportunity to work in such an amazing lab.

I would like to express my very special thanks to my co-supervisor, Sedat Dogru, for all the invaluable suggestions and ideas to improve my work, as well as his continuous and unconditional assistance and availability to assist in problem solving.

I would like to thank my lab colleagues for all of the knowledge they shared, all of the questions they answered, all of the ideas they suggested, and all of their assistance in adjusting to a new environment.

Thank you to the Embedded Systems Laboratory of the Institute of Systems and Robotics (LSE-ISR) of the University of Coimbra for providing dedicated space and materials for this effort.

I must also thank my friends who never stopped encouraging me, who always supported me, and who believed in my aspirations. A special thanks to my friends and lab partners, Chaluça and JP, who gave unquestionable support and always came up with the most brilliant ideas.

Finally, a special thank you to my parents, brothers, and aunt, for always being there for me, no matter what life throws at me, for presenting me with this incredible opportunity, and for inspiring me to follow my ambitions and my own path in life.

Resumo

Nos últimos anos, à medida que a população humana cresce e as consequências do aquecimento global se tornam mais severas, os robôs móveis agrícolas autónomos têm vindo a ser cada vez mais utilizados em múltiplas tarefas. O comportamento autónomo poderá ser melhor alcançado se o robô for capaz de se localizar e construir um mapa fiável do ambiente que o rodeia. A grande dimensão dos ambientes agrícolas requer uma representação do mapa mais compacta para reduzir os requisitos de memória e processamento, e, nesse sentido, a utilização de mapas topológicos nesse tipo de ambientes é uma boa opção. Esta dissertação propõe uma nova aplicação de mapas topológicos com ênfase em pomares, onde as árvores com as suas características distintivas servem como nós e as vizinhanças das árvores servem como arestas. A solução proposta integra uma abordagem baseada na distância entre nós juntamente com um método baseado em Iterative Closest Point (ICP) para identificar de forma eficaz os nós do mapa topológico do pomar. Além disso, foi implementada uma estratégia de localização para mostrar a capacidade do sistema de estimar a posição do robô no mapa previamente construído. Foi utilizado Robot Operating System (ROS) para implementar um conjunto de etapas que filtram e segmentam nuvens de pontos 3D para extrair os troncos das árvores do pomar, construir um mapa topológico, e obter uma estimativa da posição do robô no mapa. São apresentados resultados preliminares utilizando dados reais de um Light Detection And Ranging (LiDAR) 3D recolhidos num olival para validar a abordagem proposta.

Keywords: Robótica Agrícola, Mapeamento Topológico, Pomares, LiDAR, ICP, Localização.

Abstract

Autonomous agricultural mobile robots have seen increasing use for multiple tasks in recent years, as the human population grows and consequences of global warming become more severe. Autonomous behaviour can be better accomplished if the robot is able to localize itself and build a reliable map of its surrounding environment. The large size of agricultural environments requires a more compact map representation to reduce memory and processing requirements, and using topological maps in those environments is a good option in that direction. This dissertation proposes a novel application of topological maps with a focus on orchards, where the trees with their distinctive features serve as the nodes and the neighborhoods of the trees serve as the edges for the topological maps. The proposed solution integrates a distance-based approach alongside an Iterative Closest Point (ICP)-based method to effectively recognize the nodes of the topological map of the orchard. In addition, a localization strategy was implemented to showcase the system's ability to estimate the robot's position in the previously built map. Robot Operating System (ROS) was used to implement a set of stages in a pipeline which filter and segment 3D point clouds to extract the tree trunks of the orchard, build a topological map, and obtain an estimate of the robot's position on the map. Preliminary results using real-world 3D Light Detection And Ranging (LiDAR) data collected in an orchard are presented to validate the proposed approach.

Keywords: Agricultural Robotics, Topological Mapping, Orchards, LiDAR, ICP, Localization.

"Life isn't just about passing on your genes. We can leave behind much more than just DNA."

— Solid Snake

Contents

List of Acronyms

2D	Two-Dimensional
3D	Three-Dimensional
AHRS	Attitude and Heading Reference System
CAD	Computer-aided Design
CPU	Central Processing Unit
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
KF	Kalman Filter
LiDAR	Light Detection And Ranging
OS	Operating System
PA	Precision Agriculture
PCA	Principal Component Analysis
PCL	Point Cloud Library
PF	Particle Filter
RAM	Random Access Memory
RANSAC	Random Sample Consensus
ROS	Robot Operating System

RTK	Real Time Kinematic
RTK-GNSS	Real Time Kinematic Global Navigation Satellite System
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

The world's population has been growing at an alarming rate for the past 100 years, with today's population being three times the value than it was in the mid 20th century [34]. Alongside this, the threats to global agriculture stand out as one of the most significant in the extensive list of potential issues caused by global warming. As human population grows and the increasing consequences of global warming become more severe, so does the need to maintain a high level of food production while minimizing the effects of human activity on the environment [10]. Increasing agricultural production rates and efficiency while minimizing utilized resources is key to solving the challenge of feeding a rapidly growing human population. This can be achieved by incorporating several state-of-the-art technologies to enhance agricultural productivity [44], which is directly tied to the concept of Precision Agriculture (PA), i.e., "the application of technologies and principles to manage spatial and temporal variability associated with all aspects of agricultural production for the purpose of improving crop performance and environmental quality" [35]. PA entails the collection and processing of a sizable volume of agricultural health data. The health of plants is influenced by a number of variables, including temperature and water content. A farmer can use PA to precisely determine the conditions that must exist for a crop to be healthy, as well as the locations and quantities of these conditions at any given time. One of the main drivers of PA is the automation of agricultural tasks to improve efficiency and reduce costs [35].

In recent years, automation in agricultural environments has become more common with the use of autonomous robots that can perform several agricultural tasks such as weed detection and removal, crop collection, seeding, mowing, among others [18, 45, 11]. This arises from the need to save labour costs, prevent people from performing heavy and risky operations, and provide the farmer with precise information that will help increase productivity and efficiency [37]. One of the main requirements for automation is the ability of a mobile

(a) Olive orchard.

(b) Maize field.

Figure 1.1: Example of different agricultural environments.

robot to navigate entire crop fields or orchards in an autonomous and reliable way. To do so, the robot should be capable of localizing itself and building a map of its surrounding environment. The map creation process is usually important to provide information to human operators about the environment or to be used as a reference in the robot's localization and planning procedures [1]. There are multiple problems associated with this process, such as the inherent unstructured nature and harsh conditions of agricultural environments; the limited precision and computing power of the robot's on-board sensors and processing unit; and the dynamic environment that changes drastically over the year, all of which make mapping a difficult task [1]. Orchard environments simplify some of these problems due to the easiness of tree detection compared with, for example, crop rows such as beet and maize (Figure 1.1). Olive orchards also present more distinctiveness between different trees. Therefore, they are good candidates to start developing the implementation proposed in this dissertation.

1.2 Objectives

The main goal of this dissertation is to take a step ahead towards the challenge of agricultural automation by developing and implementing a novel application of topological maps with a focus on unstructured orchard environments, where the trees with their distinctive features serve as nodes and the neighbourhoods of the trees serve as the edges of the topological map. This approach aims to improve memory and computational efficiency by only representing and saving the environmental features that matter, as well as allowing the robot to operate in a Global Navigation Satellite System (GNSS)-denied environment. Additionally, the secondary goal includes the implementation of a preliminary localization strategy to estimate

the robot's pose on the map. This also serves as a validation method for the map building process.

To accomplish the proposed objectives, the first step is to carry out a study of related work developed in topological mapping and localization in agricultural environments, followed by setting up the required materials and methods to collect field data and proceeding with the implementation of the proposed approach. The second step includes the development of the strategy itself, with a focus on obtaining an accurate and coherent topological map as well as an acceptable estimate of the robot's pose on the map. The final step is to perform some preliminary tests to validate the chosen approach and discuss the obtained results.

1.3 Contributions

To the best of the author's knowledge, this dissertation presents the first work to develop a topological map implementation for orchard environments, where the trees with their distinctive features serve as nodes and the neighbourhoods of the trees serve as the edges of the map. The main contribution is the construction of a topological map of the orchard and the development of a strategy to uniquely describe each node using ICP and distance-based matching. Although localization is not the main goal of this dissertation, it is useful to validate the quality of the obtained map and take a step towards autonomous navigation in agricultural environments.

Some of the work developed in this dissertation was submitted and accepted in a paper for the 5th Iberian Robotics Conference[1] [52]. The conference proceedings will be published by Springer - Lecture Notes in Networks and Systems.

This dissertation was developed within the scope of the project *GreenBotics: Intelligent Robotic System for Digital Agriculture*[2], under reference PTDC/EEI-ROB/2459/2021, supported by FCT (Fundação para a Ciência e Tecnologia). The GreenBotics project seeks to make a substantial contribution to PA by combining sensing, field robotics, probabilistic machine learning, and agricultural science to create a novel soil-moisture monitoring system for crop plantations (namely maize). The project has the goal of improving the precision and reliability of early anomaly identification and monitoring in maize plantations by merging robotics, data sensors, machine learning, and agricultural expertise, both experimental and computational.

[1] See `https://www.iberianroboticsconf.eu/home`

[2] See `https://sites.google.com/view/agribots/project-greenbotics`

1.4 Document Structure

This dissertation is organized as follows:

- **Chapter 2** introduces some basic background concepts in robotic mapping as well as significant related works in topological mapping and localization in agricultural environments.
- **Chapter 3** presents some theoretical background on the methods required for the implementation of the proposed approach.
- **Chapter 4** discusses the development and implementation of the proposed approach for topological mapping and localization in orchard environments. This includes an overview of the system and its modes of operation; how the topological map is structured; and a full description of each stage of the mapping and localization pipeline.
- **Chapter 5** describes the preliminary experimental work performed to validate the proposed strategy, including the hardware and software frameworks and packages used to collect data and develop the strategy, the experimental setup and tests as well as a brief discussion of the obtained results.
- **Chapter 6** summarizes relevant conclusions and future work to improve the proposed approach.

2 Related Work

2.1 Environment Representations in Robotics

Robotic mapping seeks to solve the issue of acquiring spatial models of real-world environments using mobile robots. With the goal of creating totally autonomous mobile robots, the mapping challenge is recognized as one of the most crucial issues. We currently have reliable methods for mapping static, organized, and small-scale environments. The difficulty of mapping large-scale, dynamic, or unstructured environments is still largely unsolved [48].

There are four main types of environment representations in robotics: metric maps, topological maps, semantic maps, and hybrid maps (Figure 2.1), each with their own advantages and disadvantages.

Metric maps represent the geometry of objects in a continuous or discrete metric space and provide distance data that matches the distances actually found in the mapped real world. They have the potential for high accuracy but also for high computational costs in terms of both memory and time. The computational cost is more significant in agricultural environments, which are typically large in size and lack distinctive features for a landmark or feature-based map. Metric maps can be subdivided into two major groups: feature-based maps and grid-based maps. Feature-based maps (Figure 2.2a) include features or landmarks that are present in the environment being mapped, and these can usually be reliably detected by the robot's sensors. These types of maps have been commonly used in mobile robotics, especially for indoor environments. In [9], the features are extracted lines corresponding to walls, while [20, 51] use walls and corners as features. In grid-based maps, the workspace is decomposed into cells that hold one or more values representing the state of the cell. This decomposition can be exact or approximate. The most common type of grid-based maps and metric maps in general are occupancy grids, where each cell holds a value that represents the occupancy of the cell, i.e., either occupied, free, or unknown. There are various alternative ways to depict the occupancy of the cells in occupancy grid maps, with probability values

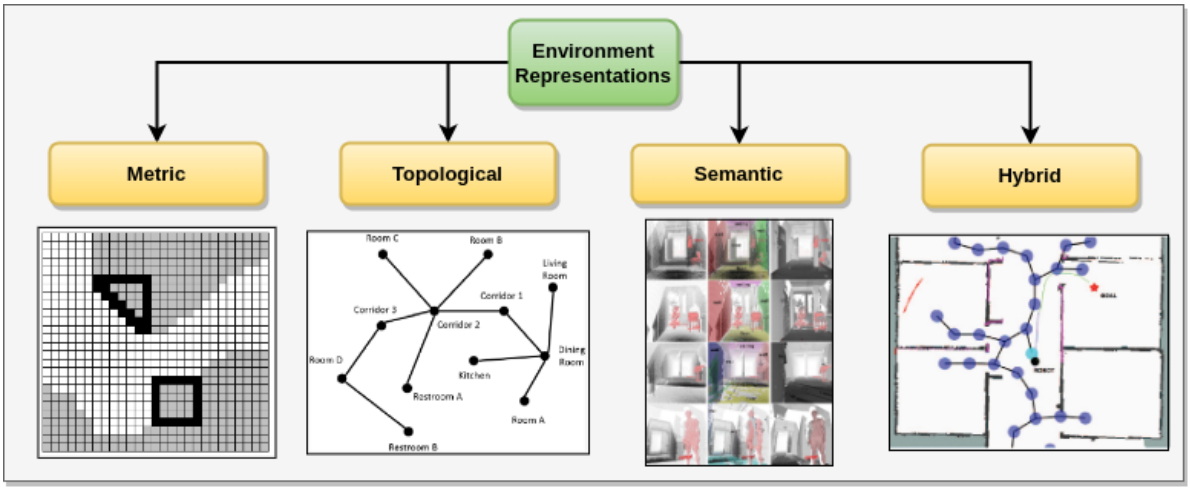


Figure 2.1: Types of environment representations (From left to right, images were retrieved from [32], [29], [33] and [39], respectively).

(a) Feature-based map [25].

(b) Occupancy grid-based map [23].

Figure 2.2: Example of metric maps.

being the most common [13]. Other representations where the cells don't have the same size have also been proposed, such as quadtrees and octrees [53, 24], to reduce memory usage while still accurately describing large environments.

Topological maps avoid direct measurement of geometric features and model the environment based on its structure and connectivity. They are frequently portrayed as graphs, where the nodes are significant locations and the edges are the links that connect them. This leads to the use of fewer computational resources overall, which is an important consideration in field robotics. Topological maps have been used in multiple implementations of mapping strategies such as in [6] and [30]. The edges are frequently described as the doors and corridors between the nodes, which are often described as some identifiable location like a room.

(a) Topological map [46]. (b) Semantic map [50]. (c) Hybrid map [12].

Figure 2.3: Example of topological, semantic and hybrid maps.

Semantic maps aim to obtain a spatial understanding beyond navigation by modelling semantic information through human-like representations. They add information about entities, such as objects, functions, or events that occur in space. The main purpose of semantic maps is to provide content that allows some type of reasoning like planning, prediction, and sensor data interpretation [33]. For that purpose, the mobile robot should be capable of object and place recognition as well as room categorization. These types of maps are used to guide high-level choices.

Hybrid maps are a combination of different types of maps, such as metric and topological, for example. Additionally, a hybrid map features extra linkages that join elements from several maps together. Since different map types have different strengths and weaknesses, the ability to make use of the unique capabilities of each map type is one of the main benefits of hybrid maps [7]. By using a hybrid map, it is possible to go beyond the drawbacks of each type of map and utilize the component of the hybrid map that is most appropriate for each activity. Multiple approaches have been proposed that utilize hybrid maps like in [47], [36] and [14]. Figure 2.3 shows examples of topological, semantic, and hybrid maps.

Topological maps have some drawbacks, such as having low accuracy, giving sub-optimal paths, as well as being more difficult to build and maintain overall. However, topological maps have several benefits, which can be helpful in agricultural environments, including good scalability to large environments, excellent for task planning, being less susceptible to errors, and not requiring a precise and reliable metric sensor model. For these reasons, topological maps are excellent choices for use in agricultural settings.

2.2 Topological Mapping in Agricultural Environments

Topological maps have seen a lot of applications in indoor or structured environments, and they have been constructed using LiDARs or cameras, sensors that also serve for navigation. Vision systems can be used for object detection and room categorization [16] as well as to perform visual place recognition [28], both of which can be used in the mapping process. LiDARs can be used to obtain geometric information for semantic labeling of places [49]. They can also be fused with vision sensors to generate topological maps [4]. Most of these applications require the robot to be operating in a structured or indoor environment.

The use of topological mapping in field applications and more specifically in agricultural environments such as crop fields and orchards, is recent. Unmanned Aerial Vehicle (UAV) or satellite images were used to extract topological maps [21, 43] to be later utilized by Unmanned Ground Vehicle (UGV) for autonomous navigation. There has been an implementation of topological mapping strategies using only UGVs as well [42]. All these topological maps focus on the free space and represent the traversable space for the robot with nodes, which are then connected according to their neighborhood in a graph. Then these maps are used for path planning for the robot. To the best of the author's knowledge, no work in topological mapping of orchard environments utilizing UGVs has been developed. Furthermore, topological mapping work in agricultural settings assumes that nodes represent free space.

2.3 Localization in Agricultural Environments

There are two main strategies for robot localization: absolute and relative. The most common approaches for absolute localization are supported by the use of Real Time Kinematic Global Navigation Satellite System (RTK-GNSS), and although this technology is becoming increasingly available and easier to incorporate into agricultural robotic systems [55], it still presents some drawbacks. Not only does it add a higher cost to the overall system both in terms of hardware (extra equipment such as a base station and antennas) and software fees (subscriptions to Real Time Kinematic (RTK) networks), but it is also susceptible to interference from atmospheric conditions and it can be blocked by the tree canopy, particularly in dense forests and orchards [15]. Degradation of the GNSS signals can quickly result in inaccuracies in the localization process, which in turn affect the mapping process negatively.

Relative localization, which relies on local sensing using LiDAR or cameras, can comple-

ment GNSS-based localization, and these sensors are also often used to facilitate sensor-based navigation and mapping [54]. This type of localization has proven useful when GNSS signals are of low quality or unavailable, which may happen often in the field. For these reasons, the use of relative localization alongside topological mapping can be convenient in agricultural environments.

A substantial amount of work has been done on localization in agricultural environments, most of which relies on a combination of multiple sensors to obtain an estimate of the robot's position. In [27], Delaunay triangulation is used to match local point clouds to a global tree map to estimate the robot's position. Winterhalter *et al.* [55] accomplishes localization by estimating the robot's pose relative to a GNSS-referenced map of crop rows, allowing it to fuse crop row detections with GNSS signals to obtain an accurate pose estimate. The most common methods of localization in agricultural environments rely on the Extended Kalman Filter (EKF) to fuse information from several sources [19]. More specifically, in orchards environments, a Particle Filter (PF) with a laser beam model and a Kalman Filter (KF) with a line-detection algorithm, using a 2D LiDAR, were used as localization strategies for robot navigation [5]. Overall, the works discussed here are concerned with localizing the robot on a previously established metric representation of the environment, with little emphasis on the mapping aspect.

3 Methods

This section gives some context for the methods utilized in the implementation of the proposed topological mapping and localization strategy.

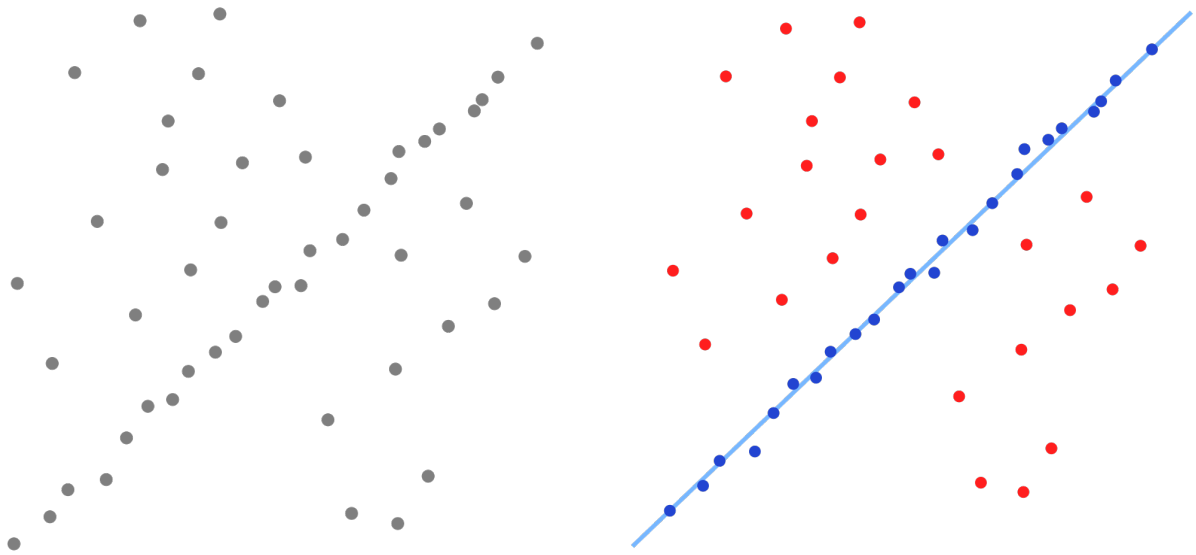
3.1 Random Sample Consensus (RANSAC)

RANSAC [17] is an iterative method for estimating a mathematical model's parameters from a set of data that includes outliers and, as such, can be considered a technique for detecting outliers. The algorithm makes the assumption that both inliers and outliers exist in all of the data we are examining, in which inliers can be fitted to a model with a specific set of parameters, while outliers never fit that model. It is a completely non-deterministic algorithm in which a result is produced with a certain probability that increases with the number of iterations.

In its most basic form, the algorithm is implemented in the following steps:

1. Select a random sample from the initial dataset.
2. Fit a model to the previous obtained subset.
3. Check the remaining dataset against the model and if a data point matches the estimated model, it is considered an inlier.
4. Check if the estimated model is sufficiently good through the number of inliers.
5. Evaluate the model by estimating the error of the inliers relative to the model.

This process is repeated a number of times, yielding either a model that is rejected because there aren't enough points identified as inliers or an improved model along with the accompanying error measure. If the improved model's error is smaller than the most recent stored model in the latter scenario, we maintain it. A set of observed data values, a method for fitting a model to the observations, and some confidence parameters make up the input



(a) A dataset of two dimensional points containing inliers and outliers.

(b) Fitted line (blue) and outliers (red).

Figure 3.1: Example of fitting a line to a set of points using RANSAC (retrieved from `https://en.wikipedia.org/wiki/Random_sample_consensus`).

to the RANSAC algorithm. Figure 3.1 shows an example of RANSAC used to fit a 2D line to a set of points.

In this dissertation, RANSAC is used in ground plane segmentation to obtain a robust estimation of the parameters for the ground plane, which can then be used to remove every point that does not belong to it. Along with the typical RANSAC parameters (number of iterations, confidence threshold, etc.), the implementation in this work also considers some restrictions in the plane's direction and normals. The plane needs to be roughly perpendicular to the z-axis of the LiDAR (which points up), and the normal direction of the points in the plane must be approximately similar to each other and to the plane's normal. This algorithm was implemented using the Point Cloud Library (PCL)[1].

3.2 Iterative Closest Point (ICP)

The challenge of correctly matching two or more point clouds, or collections of three-dimensional points, is known as 3D registration. In order to ensure that the overlapping areas between the point clouds match as closely as possible, registration determines the relative pose (position and orientation) between different point clouds in a global coordinate

[1] `https://pointclouds.org/`

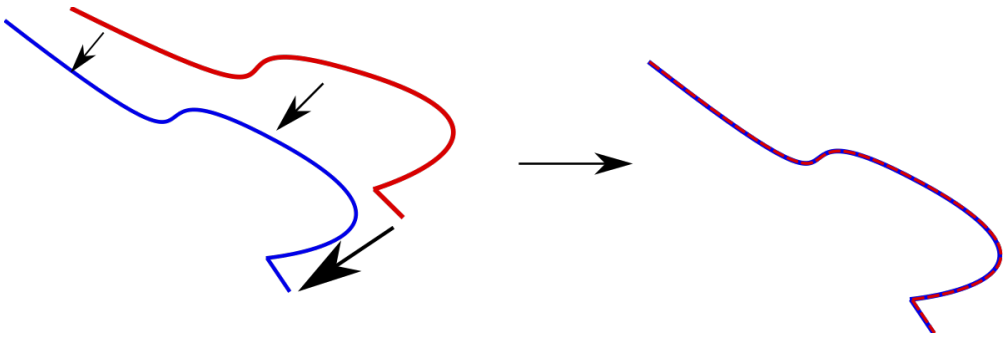


Figure 3.2: Example of the ICP algorithm (retrieved from `https://en.wikipedia.org/wiki/Iterative_closest_point`)

frame. Registration's main goal is to align disparate point clouds and combine them into a single point cloud so that processing operations like segmentation and object reconstruction can then be applied.

ICP is an iterative registration algorithm, meaning it performs a matching step to find the nearest points and then aligns the pairs of discovered points [8] (Figure 3.2). Iteratively perfecting the alignment of the source to the target point cloud, these two phases are repeated until convergence or until achieving another termination condition, such as a maximum number of iterations. There have been several implementations of this algorithm over the years. In this dissertation, the PCL implementation was used [22].

In its simplest form, the algorithm is implemented using the following steps [40]:

1. Match the closest point in the target point cloud to each point in the source point cloud.

2. Use a metric minimization technique based on the root mean square point-to-point distance to estimate the rotation and translation combination that will best align each source point with the match discovered in the previous stage.

3. Apply the obtained transformation to the source point.

4. Iterate by repeating from step 1.

The PCL implementation of this algorithm is more complex than the original one, introducing several steps to improve the final results. It is divided into the following stages:

1. Selection - Sampling Representative Subsets

2. Matching - Closest Points Correspondence Estimation

3. Rejecting and Filtering Correspondences

4. Alignment - Error Metrics and Transformation Estimation

5. Termination Criteria

The selection stage is optional and aims to reduce the computational cost of the algorithm by only registering a subset of the original point cloud. This is based on the assumption that, for the purpose of registration, data is frequently duplicated or overly detailed. In the matching stage, points from the source point cloud are matched to their closest neighbours in the target cloud. To reduce the computational cost of a greedy search, the PCL implementation uses various data structures, such as octrees [31] and K-d trees [3]. The goal of the third stage is to filter out invalid correspondences that may have a negative impact on the registration outcome in order to speed up the convergence of the transformation estimation procedure to the global minimum. There are several types of correspondence rejection methods, such as correspondence rejection based on distance, median distance, normal compatibility, RANSAC-based, etc. The alignment stage aims to minimize an error metric to obtain an estimation of the translation and rotation that transform the source point cloud into the target cloud. Like in the previous stage, there are multiple error metrics that can be used, such as the standard point-to-point error metric, the point-to-plane error metric, and linear least squares point-to-plane, among others. Finally, in the last stage, a termination criteria is defined, i.e., a condition that decides when the algorithm stops. The termination criteria may include the maximum number of iterations; an absolute transformation threshold; a relative transformation threshold; the maximum number of similar iterations; relative mean square error; and absolute mean square error. To get a more detailed explanation of the PCL implementation, see [22].

In this dissertation, the main use of ICP was to obtain the fitness score between the target and source point clouds as a similarity metric, in order to decide if the detected point clouds matched the corresponding nodes. Lower fitness scores correspond to similar point clouds. This matching process was used as a way to uniquely identify the trees, associate them with corresponding nodes and estimate the robot's location on the topological map.

3.3 Euclidean Clustering

Clustering is the task of segregating a dataset into groups with data points that share characteristics, designated clusters. In the case of 3D point clouds, a clustering method

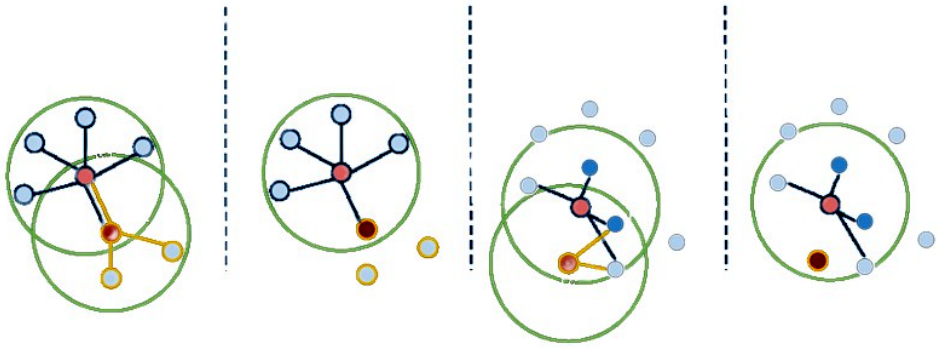


Figure 3.3: Example of visualization of euclidean clustering (adapted from [38]).

needs to divide a cloud into smaller groups with similar features. By adopting a 3D grid subdivision of the space with fixed-width boxes or, more generally, an octree data structure, a straightforward data clustering approach in the Euclidean sense can be performed. Assuming an initial point cloud dataset P and the use of a K-d tree structure for nearest neighbour search, in its simplest form, the algorithm can be implemented using the following steps (from [41]):

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 $p_i \in P$, perform the following steps:
   (a) Add p_i to the current queue Q;
   (b) For every point $p_i \in Q$ do:
      i. Search for the set P_i^k of k neighbouring points of p_i in a sphere with radius $r < d_{th}$;
      ii. For every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q;
   (c) 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 $p_i \in P$ have been processed and are now part of the list of point clusters C;

A visualization of euclidean clustering is shown in Figure 3.3.

For this dissertation, a conditional euclidean clustering algorithm is used where it is now possible to fully define the conditions that must be met in order for a neighbour to be merged into the existing cluster, in addition to the typical distance criterion. In this application, the custom condition requires a point to be similar to its neighbors in surface normal direction in order to be added to the final cluster. The user-defined criterion between points already inside the cluster and neighboring candidate points is evaluated as the cluster expands. In order to find the candidate points (nearest neighbors points) around each point in the cluster, an Euclidean radius search is used. The condition has to hold with at least one of a point's neighbors, but not with all of them, for that point to be included in the final cluster. The PCL was used for this implementation.

3.4 Multilateration

Multilateration is a method for estimating a target's position using range measurements to known points, i.e., the distances from the target to a set of n beacons. There are multiple solutions to this problem, including analytical and iterative ones. Analytical or direct algorithms estimate the target's position using only the range measurements, while iterative algorithms require a reasonably precise initial estimate of the target's position. The approach may not converge or may converge to an invalid solution if the initial estimate is not sufficiently close to the true solution. However, iterative solutions have several advantages, such as the use of redundant measurements; the incorporation of random errors in the distance measurements; and preventing ambiguous solutions that direct algorithms provide. For stationary beacons, the gradient descent multilateration algorithm is further detailed in [2]. Figure 3.4 shows an example of 2D multilateration with 3 beacons.

In this dissertation, multilateration is used to estimate the robot's position in the map's reference frame, where the beacons are the nodes of the topological map (trees), and the range measurements are the distances from the robot to each tree measured by the 3D LiDAR. This process generates a set of probable poses for the robot, which are then used alongside odometry and IMU readings to compute a final estimate of the robot's position.

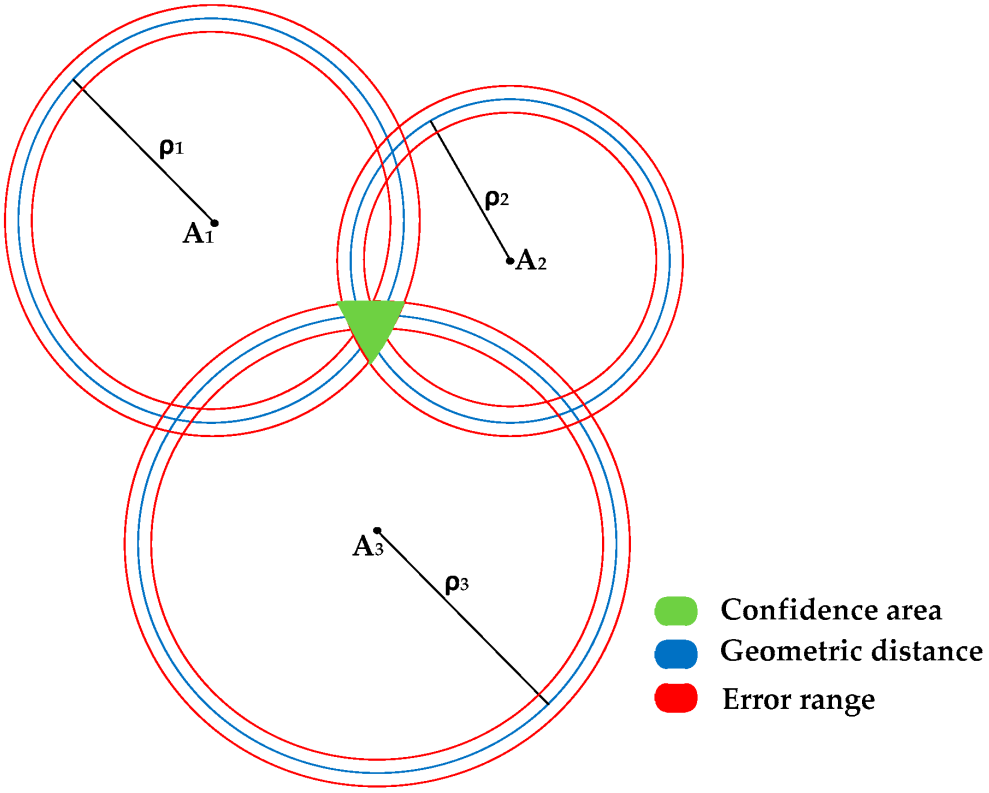


Figure 3.4: Example of 2D multilateration with 3 beacons (adapted from [26]).

4 Proposed Strategy for Topological Mapping and Localization

4.1 System Overview

The proposed system consists of a pipeline that can be split into four major stages, with the output of each stage being the input of the next one. Each stage can be further divided into multiple steps, which are better explained in the following sections. Figure 4.1 shows a general overview of the pipeline for the proposed strategy. The four main stages are the following:

- Filtering Stage
- Segmentation Stage
- Map Building Stage
- Localization Stage

The system takes point cloud data from the 3D LiDAR as input. It then applies a set of filtering steps to remove the ground plane and tree canopies in order to extract the trunks, followed by the segmentation stage that isolates individual tree trunks using euclidean clustering. The resultant set of clusters is then passed to either the localization stage or the map building stage, depending on the mode in which the system is operating. The localization stage matches detected clusters with tree trunks from a previously known map to obtain an estimate of the robot's location. The map building stage uses the detected clusters to build a topological map of the unstructured orchard environment and saves it to the disk for later use.

The proposed system has two modes of operation: the map building mode and the localization mode. In the first mode, the robot builds a topological map of the environment,

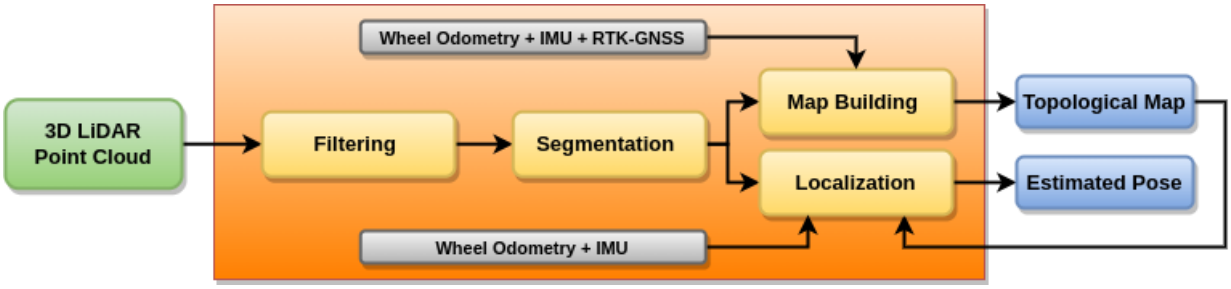


Figure 4.1: General overview of the proposed system.

while in the second mode, the robot tries to estimate its location on the previously built map. As seen in Figure 4.1, the first two stages of the system are common to both modes, meaning the input point cloud needs to be filtered and segmented into clusters to extract and isolate the tree trunks, which will be used to either build the topological map or estimate the robot's pose.

In the map building mode, a fusion of wheel odometry, IMU readings and RTK-GNSS data is used to obtain the robot's pose and build a precise topological map of the orchard. Due to the sub-centimeter accuracy of the RTK-GNSS system, we can obtain almost the exact positions of the trees in the orchard. In the localization mode, we assume that the RTK-GNSS system is not available and, as such, the robot's pose is obtained by fusing wheel odometry data, IMU readings and pose estimations from a measurement stage. Using both ICP and distance-based matching techniques, we are able to match detected clusters with the known tree trunks from the generated map and consequently obtain a measurement estimation of the robot's location in the previously built map using multilateration. This measurement estimation is then fused with data from wheel odometry and the IMU, allowing for a more precise final estimation of the robot's pose on the map.

4.2 Map Structure

A topological map M is given by

$$M = \{N,\ E\} \tag{4.1}$$

where N and E represent the set of nodes and edges, respectively, which can be described by

$$N = \{n_i\}, \tag{4.2}$$

$$E = \{e_{ij}\}, \quad i, j = 1,\ 2,\ \ldots,\ n_n \tag{4.3}$$

with n_n being the total number of nodes. In this representation, both nodes and edges can be seen as data structures that hold information about each tree and its relation with the neighbouring trees. A node representing tree i is given by

$$n_i = \{p_i,\ g_i,\ P_i,\ V_i\} \tag{4.4}$$

where $p_i = \{x_i,\ y_i,\ z_i\}$ represents the tree's position in a local fixed cartesian coordinate frame, $g_i = \{\phi_i,\ \lambda_i,\ h_i\}$ is the tree's position in GNSS coordinates (latitude, longitude and altitude), P_i is an array of point clouds that describe the tree from different viewpoints and V_i is the corresponding array of vectors that define those viewpoints, with each point cloud associated to a viewpoint. An array of point clouds and viewpoints is given by

$$P_i = \{P_{ik}\}, \tag{4.5}$$

$$V_i = \{V_{ik}\}, \quad k = 1,\ 2,\ \ldots,\ n_{vi} \tag{4.6}$$

where P_{ik} is the point cloud corresponding to viewpoint V_{ik} for node i, viewpoint k. The number of viewpoints for node i is given by n_{vi}. Each viewpoint indicates the direction from which the tree was seen and can be represented by a vector referenced in the global map frame given by

$$V_{ik} = \{v_x,\ v_y,\ v_z\} \tag{4.7}$$

where v_x, v_y and v_z represent the vector's components in the x, y and z directions, respectively.

An edge between tree i and tree j is given by

$$e_{ij} = \{d_{ij}\} \tag{4.8}$$

where d_{ij} is the edge's length, representing the distance between tree i and tree j. The set of edges is specified by an adjacency matrix A, where $A_{ij} = e_{ij}$. An example of the topological map structure is shown in Figure 4.2.

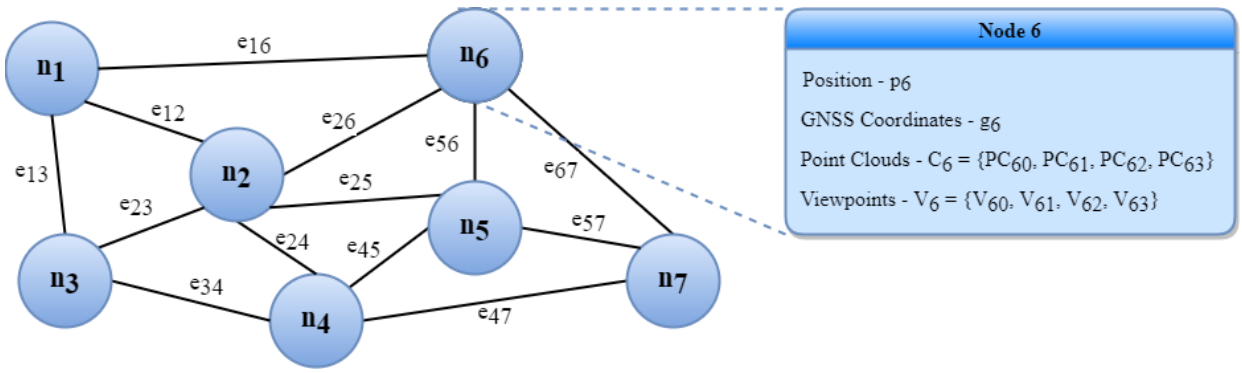


Figure 4.2: Example of the topological map structure.

4.3 Pipeline Stages

4.3.1 Filtering

The main goal of this stage is to extract the tree trunks from the original 3D point cloud of the orchard environment. As seen in Figure 4.3, this stage can be divided into the following steps:

1. Passthrough filtering (height and intensity)
2. Normal estimation
3. Ground plane segmentation
4. Outlier removal (statistical and radius based)

The filtering stage receives as input raw point cloud data from the 3D LiDAR given by

$$PC_{raw} = \{p_i\}, \quad i = 1, 2, \ldots, n_{raw} \tag{4.9}$$

with $p_i = \{x_i,\ y_i,\ z_i,\ i_i\}$, where n_{raw} is the number of points in the raw cloud and x_i, y_i, z_i and i_i correspond to the x, y, z and intensity values of point i, respectively. It then applies a set of passthrough filters to the point cloud, estimates the surface normal at each point, and removes the ground plane and outliers. This stage outputs a filtered point cloud, $PC_{filtered}$ given by

$$PC_{filtered} = \{p_i\}, \quad i = 1, 2, \ldots, n_{flt} \tag{4.10}$$

with $p_i = \{x_i,\ y_i,\ z_i,\ i_i,\ n_i\}$, where n_{flt} is the number of points in the filtered cloud and x_i, y_i, z_i and i_i correspond to the x, y, z and intensity values of point i, respectively, and n_i corresponds to the normal vector at point i.

Figure 4.3: General overview of the filtering stage.

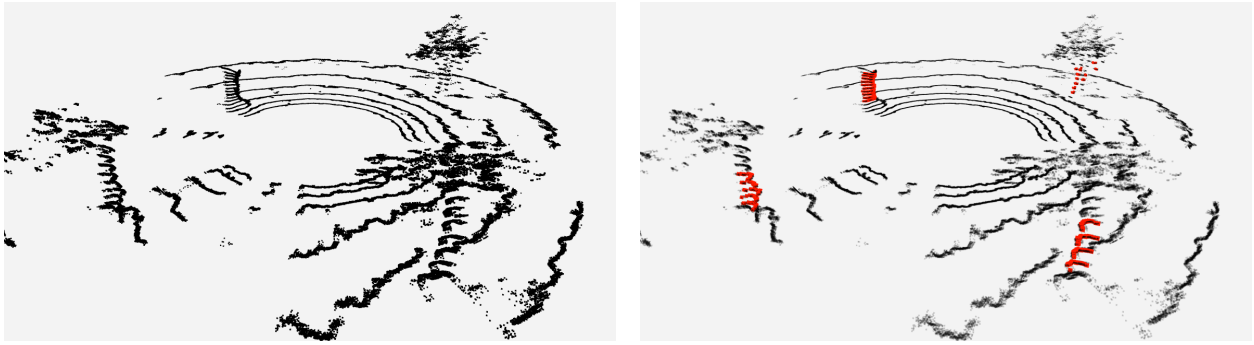
For the first step of the filtering stage, passthrough filters in both height and intensity are applied to the raw point cloud, essentially removing every point with height or intensity values that does not fall into a specific interval. The height corresponds to the z coordinate in the LiDAR's reference frame. As outlined in Algorithm 1, we traverse all points in PC_{raw} and remove every point p_i with z and i values outside a specific interval. This step is essential to significantly reduce the number of points in the cloud, decreasing the computation times for real-time processing.

In the second step, the surface normal is computed for every point in the filtered cloud. The problem of determining the normal at a point on the surface is approximated by the problem of estimating the normal of a plane tangent to the surface, which in turn becomes a least-square plane fitting estimation problem. The solution for estimating the surface normal is therefore reduced to an analysis of the eigenvectors and eigenvalues (Principal Component Analysis (PCA)) of a covariance matrix created from the nearest neighbors of the query point[1]. The estimated normals are then concatenated to $PC_{filtered}$, adding another field to each point (n_i). This step is required for the ground plane segmentation.

In the third step, a plane segmentation based on RANSAC is applied to the filtered cloud, to isolate and remove the ground plane. The implemented method for segmentation not only uses the previously estimated surface normals to extract a plane equation approximately perpendicular to the z-axis of the LiDAR's reference frame but also to decide if a point belongs to the extracted plane based on surface normal similarity between neighbouring points. In order to reduce computation times, the plane segmentation is only applied to points that belong to a specific interval, assuming the ground plane is always below the origin of the LiDAR's frame. A more detailed explanation of the RANSAC algorithm was detailed in Section 3.1.

Finally, in the fourth step, two filters are applied to the filtered cloud to eliminate any outliers that were not removed in the previous steps. The first one is a radius outlier removal filter, which removes all points in the cloud that do not have at least some number

[1]PCL Library - Estimating Surface Normals in a Point Cloud - `https://pointclouds.org/documentation/tutorials/normal_estimation.html`



(a) Original point cloud (b) Filtered point cloud (shown in red)

Figure 4.4: Example of results from the filtering stage.

of neighbours within a certain range. As described in Algorithm 1, we compute the number of neighbouring points (n_n) inside a specific radius (r) for each query point p_i in $PC_{filtered}$. If n_n is less than a certain value (N_n), the corresponding point p_i is removed. The second outlier removal filter performs a statistical analysis on each point's neighbourhood and trims those that do not meet a certain criteria. As seen in Algorithm 1, we compute the mean distance (d_i) from each query point p_i in $PC_{filtered}$ to its k nearest neighbours. By assuming this distribution is Gaussian, we then compute the mean (μ_d) and standard deviation (σ_d) of these distances. All points whose mean distance (d_i) is greater than a threshold ($D_{max} = \mu_d + \alpha \cdot \sigma_d$) are removed from the point cloud, where α is a tunable standard deviation multiplier. Figure 4.4 shows an example of the results from the filtering stage.

4.3.2 Segmentation

The goal of this stage is to isolate the tree trunks obtained from the previous stage into separate clusters, to be used either by the map building stage to generate a new topological map or by the localization stage to estimate the robot's pose on the map. Figure 4.5 shows a general overview of the segmentation stage.

This stage takes as input the filtered point cloud $PC_{filtered}$ from the previous stage given by Equation 4.10 and segments it into clusters to obtain distinct tree trunks, while also

Figure 4.5: General overview of the segmentation stage.

Algorithm 1: Tree trunk extraction from 3D point clouds.

Input: A raw 3D point cloud $PC_{raw} = \{p_i\}_{i=1...n_{raw}}$

Output: A filtered 3D point cloud $PC_{filtered} = \{p_i\}_{i=1...n_{flt}}$

$PC_{filtered} \leftarrow PC_{raw}$;

$D = []$;

`// ------- Passthrough filters -------`

for $p_i \in PC_{filtered}$ **do**

  if $p_i.z \notin [z\_min, z\_max]$ ***or*** $p_i.i \notin [i\_min, i\_max]$ **then**

    Remove p_i from $PC_{filtered}$

  end

end

`// ------- Normal estimation and ground plane removal -------`

$PC_{filtered} \leftarrow$ normalEstimation($PC_{filtered}$);

$PC_{filtered} \leftarrow$ removeGroundPlane($PC_{filtered}$);

`// -------- Radius outlier removal filter --------`

for $p_i \in PC_{filtered}$ **do**

  $n_n \leftarrow$ numNeighbours(p_i, r);

  if $n_n < N_n$ **then**

    Remove p_i from $PC_{filtered}$

  end

end

`// ------- Statistical outlier removal filter -------`

for $p_i \in PC_{filtered}$ **do**

  $d_i \leftarrow$ meanDistanceK(p_i, k);

  D.append(d_i)

end

$\mu_d \leftarrow$ mean(D);

$\sigma_d \leftarrow$ std(D);

$D_{max} \leftarrow \mu_d + \alpha \cdot \sigma_d$;

for $p_i \in PC_{filtered}$ **do**

  $d_i =$ meanDistanceK(p_i, k);

  if $d_i > D_{max}$ **then**

    Remove p_i from $PC_{filtered}$

  end

end

computing the clusters' centroids and number of points. It outputs an array of detected clusters given by

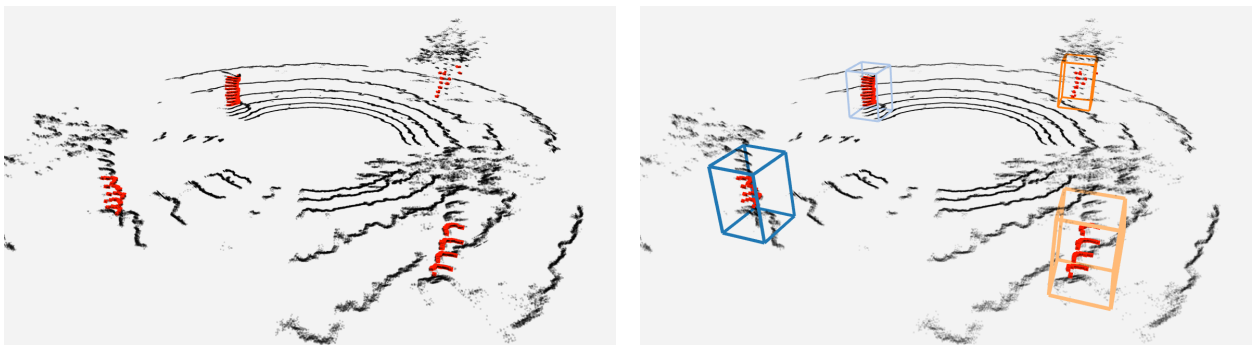
$$C_{detected} = \{C_i\}_{i=1...m} \tag{4.11}$$

where m is the number of detected clusters. Each cluster can be represented by a custom data structure that contains some relevant information about the cluster, given by

$$C_i = \{n_{pi},\ c_i,\ PC_i\} \tag{4.12}$$

where n_{pi}, $c_i = \{c_x,\ c_y,\ c_z\}$ and PC_i are the number of points, centroid position in a cartesian coordinate frame and 3D point cloud of cluster i, respectively. Similar to the map nodes' data structure, this can be easily expanded to include more information about the cluster.

An overview of this stage is outlined in Algorithm 2. To segment the point cloud into clusters, the Euclidean Clustering method is used together with a custom clustering condition that needs to hold for a point to be added to a cluster. This condition requires a point to be similar in surface normal direction to its neighbours in order to be added to a cluster. This custom condition removes any noisy points that do not belong to a trunk but are still close enough to be considered part of the cluster by the base Euclidean Clustering algorithm, like points belonging to leaves at lower heights. Clusters that are considered too small or too large, i.e., with a number of points below or above a specific threshold, are also removed. A more detailed explanation of the base euclidean clustering algorithm was described in Section 3.3. After extracting the clusters, their centroid (c) is computed and saved in the corresponding data structure, along with the number of points (n_p) and the cluster's point cloud coordinates (PC). All clusters are saved into an array to be used by the next stages. Figure 4.6 shows an example of the results from the segmentation stage.



(a) Filtered point cloud (shown in red) (b) Detected clusters (shown with boxes)

Figure 4.6: Example of segmentation results.

Algorithm 2: Segmentation of tree trunks into clusters.

Input: A 3D point cloud $PC_{filtered} = \{p_i\}_{i=1...n_{flt}}$

Output: An array of detected clusters $C_{detected} = \{C_i\}_{i=1...m}$

$C' \leftarrow$ ConditionalEuclideanClustering($PC_{filtered}$);

for $C_i \in C'$ **do**

$\bar{p} \leftarrow \text{mean}_{p \in C_i}(p)$;

$C.n_p \leftarrow C_i.points.size()$;

$C.c \leftarrow \bar{p}$;

$C.PC \leftarrow C_i.points$;

$C_{detected}$.append(C);

end

4.3.3 Map Building

The goal of this stage is to build the topological map using the detected clusters corresponding to different trees. With that purpose in mind, each cluster is matched against every known node and according to the distance between them, either a new node is added or an existing node is updated. Figure 4.7 shows a general overview of this stage, which can be split into the following steps:

1. Viewpoint computation
2. Cluster matching
3. Edges update

The map building stage receives as input the cluster array from the previous stage ($C_{detected}$), matches them against every known node and according to the matching result, it either adds a new node to the map or updates an existing one. It then outputs an updated version of the topological map, $M = \{N, E\}$.

As seen in Figure 4.7, the first step is to obtain the viewpoint from where the tree was detected. The viewpoint can be represented by a vector referenced in the global map

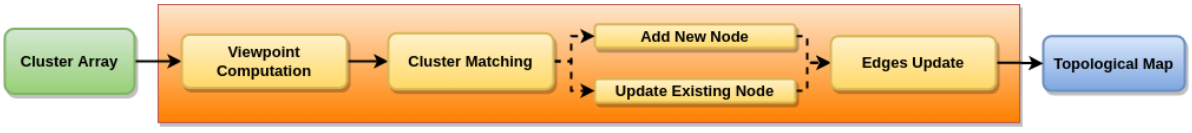


Figure 4.7: General overview of the map building stage.

frame as mentioned in Eq. 4.7 and is obtained by subtracting the cluster's centroid from the robot's position in the map's reference frame. The viewpoint is computed using the following equation

$$V_i = \bar{p}_{robot} - \bar{C}_i.c, \quad i = 1,\ 2,\ \ldots,\ m \tag{4.13}$$

where $\bar{p}_{robot}$ is the robot's position, $\bar{C}_i.c$ is the centroid of the detected cluster and m is the number of detected clusters. The resultant vector is then normalized. This is computed for every detected cluster.

In the second step, we traverse the topological map's known nodes (N) and match each cluster against them. This matching is solely based on position similarity, which means that if the distance between a cluster's centroid ($C_i.c$) and a node's position ($n_j.p$) is less than a certain threshold (d_{max}), we can assume the detected cluster i matches node j in the map. Otherwise, the detected cluster i is added as a new node to the map.

In the first case where the detected cluster matches an existing node, the matched node's position ($n_j.p$) is updated by computing the mean between the centroid of cluster i and the position of node j,

$$n_j.p = \frac{n_j.p + C_i.c}{2} \tag{4.14}$$

Then, the respective GNSS coordinates are computed by converting the updated position in the map's reference frame ($n_j.p$) to geodetic coordinates. If the computed viewpoint already existed in the database, then the corresponding point cloud ($n_j.C$) is updated by registering the detected cluster point cloud ($C_i.PC$) with it. Otherwise, a new viewpoint for node j is added. Since the viewpoints are represented by vectors, their similarity can be computed by verifying if the inner product between both vectors is below a certain threshold.

In the second case, when the detected cluster i does not match any existing node, it is added to the map as a new node, with the cluster's centroid as the new node's position, the GNSS coordinates computed the same way as in the previous case, and a new single viewpoint added along with the respective point cloud.

Finally, in the third step, the map's edges are updated, if new nodes were added. The updating process is solely based on node distance, which means that if the distance between nodes i and j is less than a certain threshold, the nodes are connected, and the distance is saved in the edge data structure (e_{ij}). With this strategy, the map can also be saved to a file to be analysed offline and loaded from a file when the robot's operation resumes, allowing the robot to start from a previously known map. The algorithm described above is applied to every detected cluster and it is outlined in Algorithm 3.

Algorithm 3: Topological map update.

Input: An array of detected clusters $C_{detected} = \{C_i\}_{i=1...m}$

Output: A 3D topological map $M = \{N, E\}$

```
for C_i ∈ C_detected do
    V_i ← computeViewpoint(C_i);
    for n_j ∈ N do
        d = distance(C_i.c, n_j.p);
        if d < d_max then
            n_j.p ← mean(n_j.p, C_i.c);
            n_j.P ← updatePointCloud(C_i.PC, V_i); // Does viewpoint comparison
            n_j.g ← updateGNSSCoords(n_j.p);
            node_exists ← true ;
            break ;
        end
    end
    if node_exists then
        continue ;
    else
        N.addNode(C_i);
        E.updateEdges();
    end
end
```

4.3.4 Localization

The goal of this stage is to match detected clusters with tree trunks from a previously known map to obtain an estimate of the robot's pose on the map. This is achieved by fusing a pose estimate from a prediction stage with a pose estimate from a measurement stage. Figure 4.8 shows a general overview of this stage, which can be divided into the following steps:

1. Local map matching
   (a) Cluster matching
      i. Distance-based matching
      ii. ICP-based matching
   (b) Consistency check
      i. Creation of node-cluster combinations
      ii. Removal of impossible combinations
2. Multilateration
3. Localization update

The localization stage takes as input the cluster array from the segmentation stage ($C_{detected}$), performs a cluster-node matching based on distance between clusters and based on the ICP algorithm, generating a list of possible node-cluster combinations. Using multilateration, this list is then used to compute a set of possible poses, based on distance measurements from the robot to each cluster, which are then fused with odometry data to obtain a final pose estimate.

Local Map Matching

The first step (cluster matching) can be subdivided into two phases. In the first phase, detected clusters (trees) are matched against known nodes based on the distance between

Figure 4.8: General overview of the localization stage.

clusters. As outlined in Algorithm 4, for a single cluster, the distance to every other cluster is computed and compared with known distances between nodes (E). If the difference between them is less than a certain threshold (D_{th}), the two corresponding nodes are added to a temporary list of nodes (a). This list is then pushed into an array (A) after removing any duplicate elements. After traversing through every cluster, the common elements between every element of A are computed, resulting in a preliminary list of candidate nodes (N_c) for that particular cluster. For example, assuming n clusters were detected, each one will have $n-1$ associated temporary lists of nodes (a) corresponding to the number of connections between that particular cluster and the other clusters. The common elements between the cluster's associated lists of nodes are computed to obtain the preliminary list of candidate nodes (N_c) for that particular cluster. This algorithm is repeated for every detected cluster. This phase is important to reduce the number of possible candidate nodes and therefore the number of times the ICP algorithm is applied, decreasing computation times.

In the second phase of cluster matching, an ICP-based matching process is used to obtain a reduced set of candidate nodes. This phase takes as input the preliminary list of candidate nodes generated by the previous stage and outputs a final set of candidate nodes for each cluster. As outlined in Algorithm 5, for a single cluster, its corresponding point cloud ($C_{query}.PC$) is registered against every cloud ($n_i.P$) of the nodes in the preliminary candidate node list (N_c). If the resulting fitness score is less than a specific threshold, the respective node is added to the final list of candidate nodes (N_{cf}). Similar to the previous one, this method is applied to every detected cluster.

The second step (consistency check) is also divided into two phases. The first one aims to produce all possible cluster-node combinations from the final lists of candidates nodes of each cluster. Figure 4.9 shows an example where 4 clusters were detected. Each one has a corresponding list of candidate nodes. For example, cluster 0 (C_0), may correspond to node 2, 3, or 11, and a likely cluster-node combination might be (2, 5, 4, 12), i.e., the first cluster corresponds to node 2, the second to node 5, the third to node 4, and the last cluster corresponds to node 12. The text in red represents the actual corresponding node (ground truth). A fragment of all possible combinations is presented in Table 4.1, where it can be seen that the final set of possible cluster-node combinations might contain some inconsistencies. For example, different clusters might be matched to the same node. Because of this, in the second stage of consistency check, all impossible combinations are removed by comparing the distances between the clusters in a particular combination with the actual distances in the known topological map. If there is any discrepancy between these values,

Algorithm 4: Distance-based matching for a single cluster.

Input: A query cluster c_{query} and array of detected cluster $C_{detected}$

Output: A list of candidate nodes for the query cluster N_c

$A \leftarrow [\,]$;

$a \leftarrow [\,]$;

for $C_i \in C_{detected}$ **do**

  $d \leftarrow$ computeDistance($C_i.c,\ c_{query}.c$);

  for $e_{jk} \in E$ **do**

    $D = \text{abs}(d - e_{jk})$;

    if $D < D_{th}$ **then**

      a.append(j);

      a.append(k);

    end

  end

  $a \leftarrow$ removeDuplicates(a);

  A.append(a);

end

$N_c =$ findCommonElements(A);

Algorithm 5: ICP-based matching for a single cluster.

Input: A query cluster c_{query} and preliminary list of candidate nodes N_c

Output: A final list of candidate nodes for the query cluster N_{cf}

for $n_i \in N_c$ **do**

  $score \leftarrow$ registerPointClouds($n_i.P,\ c_{query}.PC$);

  if $score < score\_th$ **then**

    N_{cf}.append(i);

  end

end

$N_{cf} \leftarrow$ findCommonElements(A);

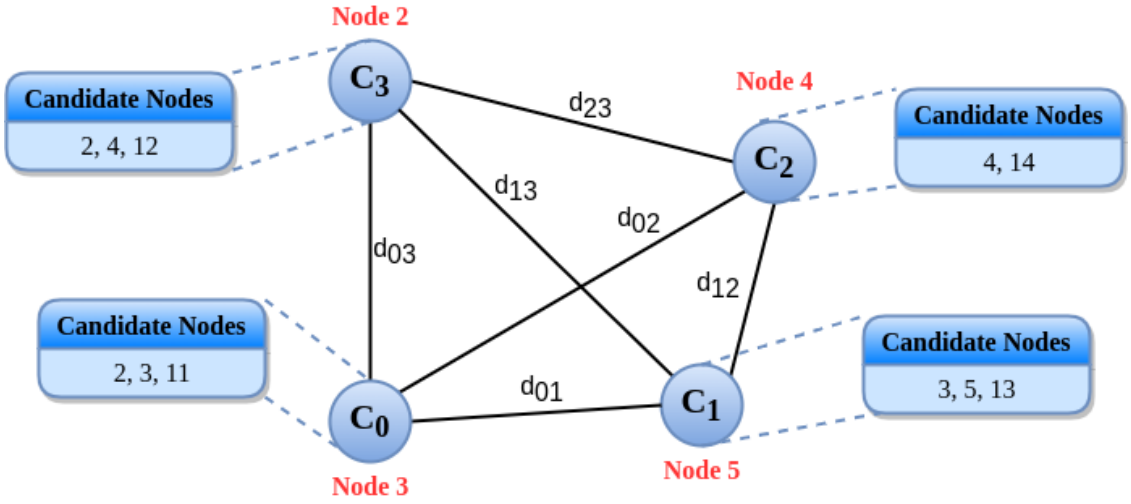


Figure 4.9: Example of the cluster matching process, where 4 cluster were detected.

Table 4.1: Example of possible cluster-node combinations.

Combination #	Nodes				Remove (yes/no)
	C_0	C_1	C_2	C_3	
1	2	3	4	2	Yes
2	2	3	4	4	Yes
3	2	3	4	12	Yes
4	2	5	4	2	Yes
...	...	...	...	...	...
...	3	5	4	2	No
...	...	...	...	...	...
n	11	13	14	12	No

the combination is removed. As such, the final output of the first step (local map matching) is an array of possible node-cluster combinations. This step only occurs if the number of detected trees is greater than one.

Multilateration

In the second step of the localization stage, multilateration is used to compute the robot's pose for each possible cluster-node combination from the previous step. This generates a set of probable poses for the robot. Because these poses result from LiDAR measurements, they are considered measured poses. The multilateration algorithm mentioned in Section 3.4 is applied to every possible node-cluster combination, generating a set of probable robot poses.

For example, for the combination shown in the previous step, nodes 2, 3, 4, and 5 would be the beacons in the multilateration algorithm, and using the measured distances from the robot to each of the clusters, a pose would be estimated. Details of the multilateration algorithm are explained in Section 3.4.

Position Update

Finally, in the last step, the robot's current pose is estimated by fusing the measured poses from multilateration with the predicted poses that result from previously estimated poses and odometry information. The system maintains a list of possible poses that is updated in each iteration, with the least probable ones being discarded. Assuming $\hat{x}$ is the state vector representing the robot's pose in 2D given by

$$\hat{x} = [x,\ y,\ \theta] \tag{4.15}$$

the list of probable poses at iteration k is given by

$$X_k = \{\hat{x}_1,\ \hat{x}_2,\ \ldots,\ \hat{x}_n\} \tag{4.16}$$

where n is the number of poses.

The pose update process consists of two parts. In the first one, the robot's current pose is predicted using data from wheel odometry, the IMU and previous estimated poses. In the second one, the current estimated pose is updated by fusing the predicted pose with the measured pose from the multilateration step. Assuming a pose at iteration k, $\hat{x}_k$, the current predicted pose is given by

$$\hat{x}_k^- = \hat{x}_{k-1} + \Delta\hat{x}_{k-1,\,k} \tag{4.17}$$

where $\hat{x}_{k-1}$ is the previous estimated pose and $\Delta\hat{x}_{k-1,\,k}$ is robot's displacement between iteration $k-1$ and k. This displacement is given by a fusion between data from the wheels odometry and IMU, provided by the `robot_localization` package from ROS. This package uses an EKF to obtain an estimate of the robot's pose. In the second part of this update process, the current estimated pose is computed

$$\hat{x}_k = \hat{x}_k^- \oplus \hat{z}_i^k \tag{4.18}$$

where $\hat{z}_i^k$ is the current measured pose from the multilateration step. This fusion is computed by averaging both poses. From the set of poses provided by the multilateration step, not all of them need to be fused with the current predicted pose. With that in mind, a strategy

based on range gates is used to determine which of the measured poses is fused with the current predicted pose. Only the ones within a certain distance of the current predicted pose are used in the update process.

The steps described above are applied to every pose maintained by the system. Since this is a multi-hypothesis process, some improbable poses need to be discarded over time. This is accomplished using a counter (cnt) associated with each pose, which keeps track of the number of times that particular pose was updated using only the wheels' odometry and IMU, i.e., when $\hat{x}_k = \hat{x}_k^-$. If that counter goes above a certain threshold (cnt_{th}), it can be assumed that the measurements do not confirm the predictions and, as such, the pose can be discarded. The localization update step is outlined in Algorithm 6.

The set of estimated poses is initialized equal to the first set of measured poses, i.e., given a random start point, the system cannot obtain an initial estimate of the robot's position until at least two trees are detected. When this occurs, the estimated poses are set equal to the measured poses computed using multilateration, and the standard position update process is resumed for the next iterations.

Algorithm 6: Robot's localization update

Input: List of previous estimated poses X_{k-1}, list of current measured poses Z_k

Output: List of current estimated poses X_k

$\Delta x_{k-1,\,k} \leftarrow$ computeOdomDelta(k);

for $\hat{x}_{k-1} \in X_{k-1}$ **do**

    // Discard improbable poses

    if $cnt_i > cnt_{th}$ **then**

        continue ;

    end

    // Predict pose

    $\hat{x}_k^- = \hat{x}_{k-1} + \Delta x_{k-1,\,k}$;

    // Update pose

    for $z_i^k \in Z_k$ **do**

        $d \leftarrow$ computeDistance($z_i^k,\ \hat{x}_k^-$);

        if $d < d_{max}$ **then**

            $\hat{x}_k \leftarrow$ fusePoses($z_i^k,\ \hat{x}_k^-$);

            X_k.append($\hat{x}_k$);

            $pose\_updated \leftarrow$ **true** ;

            break ;

        end

    end

    if $pose\_updated$ **then**

        continue ;

    else

        X_k.append($\hat{x}_k^-$);

        $cnt_i \leftarrow cnt_i + 1$;

    end

end

5 Experimental Work

5.1 Hardware Description

The mobile platform used in this work was a customized Clearpath Husky UGV[1] (Figure 5.1) equipped with a RTK-GNSS receiver in a dual antenna configuration, a 3D LiDAR, an Inertial Measurement Unit (IMU) and a WiFi Antenna. Figure 5.2 shows a diagram of the used hardware.

The on-board LiDAR is a Velodyne Puck[2] (formerly known as Velodyne VLP-16), a 16-channel LiDAR with a range of up to 100 m, 360° horizontal field of view, 30° vertical field of view ($-15°$ to 15°), typical range accuracy of ± 3 cm, vertical angular resolution of 2°, horizontal angular resolution of $0.1° - 0.4°$ and $5 - 20$ Hz rotation rate. It typically consumes 8 W and outputs up to $300,000$ points/sec through an Ethernet connection.

The GNSS system consists of a Septentrio Ruggedized Box containing an AsteRx4 multi-frequency dual antenna GNSS receiver[3] operating in RTK mode supported by a local GNSS base station. The receiver has 544 channels for tracking known GNSS signals for all major constellations (GPS, GLONASS, Galileo, BeiDou, QZSS, IRNSS and SBAS) on both antennas, horizontal accuracy of 0.6 cm, vertical accuracy of 1 cm and support for operation in RTK mode. The system also contains two PolaNt-x MF[4] multi-frequency antennas including L1/L2/L5 with L-Band correction services.

The IMU is a XSens MTi-300, an Attitude and Heading Reference System (AHRS) with a roll and pitch accuracy of 0.2° RMS, yaw accuracy of 1° RMS and 520 mW of power consumption. Its gyroscope has standard full range of $450°/s$, in-run bias stability of $10°/h$ and $0.01°/s/\sqrt{Hz}$ noise density. Its accelerometer has standard full range of 20 g, in-run

[1] See `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`

[2] See `https://velodynelidar.com/products/puck/`

[3] See `https://www.septentrio.com/en/products/gnss-receivers/oem-receiver-boards/asterx4-oem`

[4] See `https://www.septentrio.com/en/products/antennas/polant-x-mf`

Figure 5.1: Customized Husky robot used in the implementation.

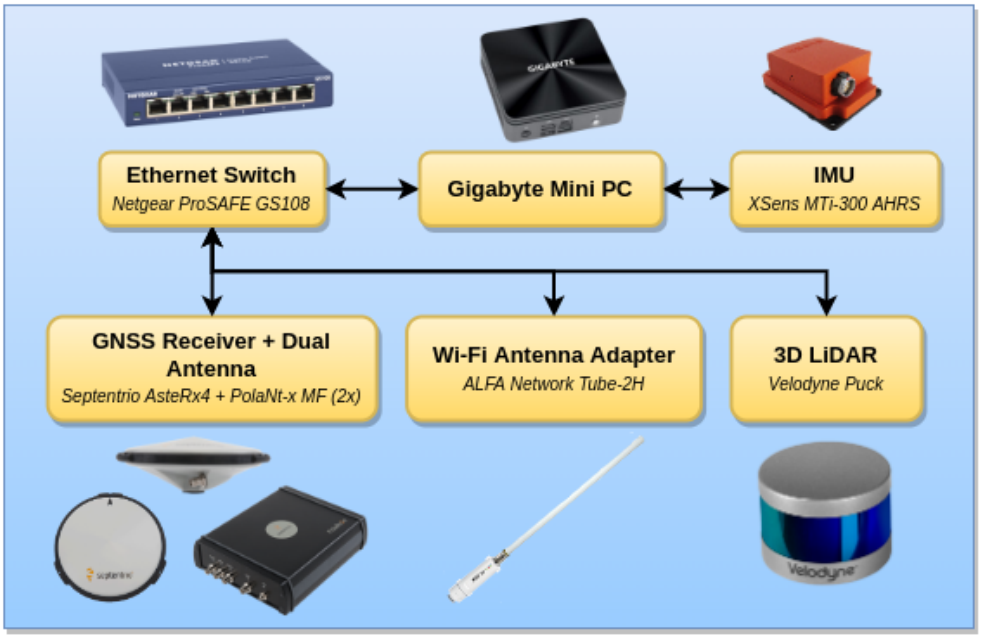


Figure 5.2: Schematic of the hardware.

bias stability of 15 µg and 60 µ$g/\sqrt{Hz}$ noise density.

Husky was also equipped with a computing platform running an Intel Core i7-3537U Central Processing Unit (CPU) (2 cores, up to 2.00 GHz) and 8 GB of Random Access Memory (RAM), as well as a pair of stereo cameras that were not used. Before starting to develop the software and implement the proposed strategy, Husky was equipped with the aforementioned sensors. For that purpose, some mounts for the IMU, LiDAR and on-board cameras were designed and 3D printed (See Appendix A).

5.2 Software

The proposed system was developed in C++ using the ROS[5] middleware suite due to its advantages in the development of software for robotics. ROS isn't computationally expensive and is supported by good simulation and visualization tools like Gazebo[6] and Rviz[7]. It also simplifies the data gathering process through the use of the `bag` files, allowing for the creation of datasets for future use, accelerating and simplifying the development process.

Before the data collection process, Husky had to be configured with the relevant software. The Operating System (OS) was upgraded to Ubuntu 20.04.4[8] running ROS Noetic. To interface with the hardware and fetch data in a readable form, the following off-the-shelf packages were installed and configured:

- 3D LiDAR - `velodyne` metapackage (includes the `velodyne_driver`, `velodyne_laserscan`, `velodyne_msgs` and `velodyne_pointcloud` packages)
- RTK-GNSS - `septentrio_gnss_driver`
- IMU - `ethzasl_xsens_driver`

The IMU package had to be migrated from ROS Melodic to ROS Noetic. To implement the proposed solution, a custom package **pcl_processing** was developed containing three separate nodes:

- **Filtering Node**

[5]See `https://www.ros.org/`
[6]See `https://gazebosim.org/home`
[7]See `http://wiki.ros.org/rviz`
[8]See `https://releases.ubuntu.com/20.04/`

  - Subscribed Topics:
    - `velodyne_points` (sensor_msgs/PointCloud2)
  - Published Topics:
    - `filtered_cloud` (sensor_msgs/PointCloud2)

- **Segmentation Node**
  - Subscribed Topics:
    - `filtered_cloud` (sensor_msgs/PointCloud2)
  - Published Topics:
    - `clusters` (pcl_processing/ClusterArray)

- **Mapping & Localization Node**
  - Subscribed Topics:
    - `clusters` (pcl_processing/ClusterArray)
  - Published Topics:
    - `topological_map` (visualization_msgs/MarkerArray)
    - `robot_poses` (geometry_msgs/PoseArray)

The filtering node subscribes to a point cloud, then performs a series of pre-filtering operations before publishing the filtered cloud. The segmentation node subscribes to the filtered cloud, divides it into clusters, each of which corresponds to a distinct tree, and sends a custom message containing information on the array of found clusters. Finally, the mapping node subscribes to the previously published cluster array, constructs the topological map, and estimates the robot's position, publishing a set of visualization markers containing the map as well as an array of all probable robot poses. This node also has a save/load capability, which allows the user to save the generated topological map into a file when the robot stops operating and load it when the robot resumes operation. Figure 5.3 shows a graph of the system's nodes and their relations.

The developed custom messages **pcl_processing/ClusterArray** and **pcl_processing/Cluster** are composed of the following fields:

- **pcl_processing/Cluster**

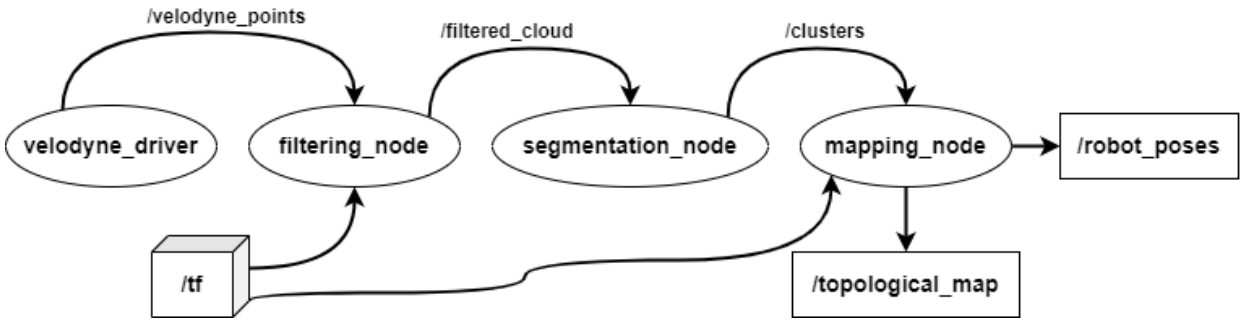


Figure 5.3: Graph of the system's ROS nodes and topics.

  - **uint32** num_points
  - **geometry_msgs/Point** centroid
  - **sensor_msgs/PointCloud2** cloud

- **pcl_processing/ClusterArray**
  - **std_msgs/Header** header
  - **pcl_processing/Cluster[]** clusters

5.3 Experimental Setup

The experiments were conducted in an olive orchard (Fig. 5.4) in Polo II, Coimbra, Portugal, during the summer season. The terrain was ramp-like with an uneven surface and a significant height difference between the lowest and the highest point. Additionally, most olive trees did not have straight, upright trunks. For practicality reasons and to speed up the development process, the data was collected in a ROS `bag` file which contained odometry, IMU and RTK-GNSS readings apart from the 3D point clouds.

To reduce odometry errors and obtain cleaner 3D point clouds, the robot was driven between rows and stopped roughly every 2 m to collect the point clouds. The robot's position was estimated by fusing data provided by wheel odometry, IMU and RTK-GNSS readings using the `robot_localization` package from ROS, which utilizes an EKF. This position was used as a ground truth for the tests presented in Section 5.4, due to the sub-centimeter accuracy of the RTK-GNSS readings during the data collection process.

A total of three datasets were collected, with the goal of obtaining different viewpoints of the same trees and testing the implemented strategy's robustness to different paths travelled by the robot. In the first run, the robot moved along the orchard rows downhill, turned

Figure 5.4: Olive orchard where the experiments were conducted.

back around a tree, and returned following an approximately parallel path to the downhill path. In the second run, the robot followed roughly the same path as in the first run but in the opposite direction. Finally, in the last run, the robot's path followed a zigzag pattern around the trees and returned to its initial position. Figure 5.5 depicts the travelled paths. During data collection, the robot was driven manually using a remote controller.

Figure 5.5: Different paths travelled by the robot for the data collection process.

5.4 Tests and Results

A set of preliminary tests were conducted in order to validate the proposed implementation. These tests were divided into three categories according to the different stages in the proposed pipeline. The filtering and segmentation tests were coupled together to evaluate the tree trunk extraction process. The second category of tests includes the quality assessment of the obtained topological map. Finally, the third category of tests entails the evaluation of the localization procedure.

5.4.1 Filtering and Segmentation

Since the main objective of the filtering and segmentation stages is to extract and isolate the tree trunks from the original 3D point cloud, both of these stages are inherently coupled together and, as such, their experimental tests. The primary goal of the filtering and segmentation tests is to evaluate the tree trunk extraction performance of the proposed strategy. The system should be able to correctly detect and extract only the trunks of each tree. With that purpose in mind, for each of the datasets mentioned above, the numbers of correct and incorrect tree detections were computed and are shown in Table 5.1. Correct detections occur when the system recognizes and extracts tree trunks, whereas wrong detections occur when the system detects other aspects of the environment as tree trunks, such as branches, leaves, ground, and so on. The datasets were manually analyzed, counting the number of times a detected cluster did not correspond to a tree (incorrect detection) as well as the number of successfully identified trees (correct detections). Examples of correct and incorrect detections are shown in Figure 5.6. This test allows for the detection of errors across all steps of the filtering and segmentation stages, such as the ground removal, trunk segmentation, etc.

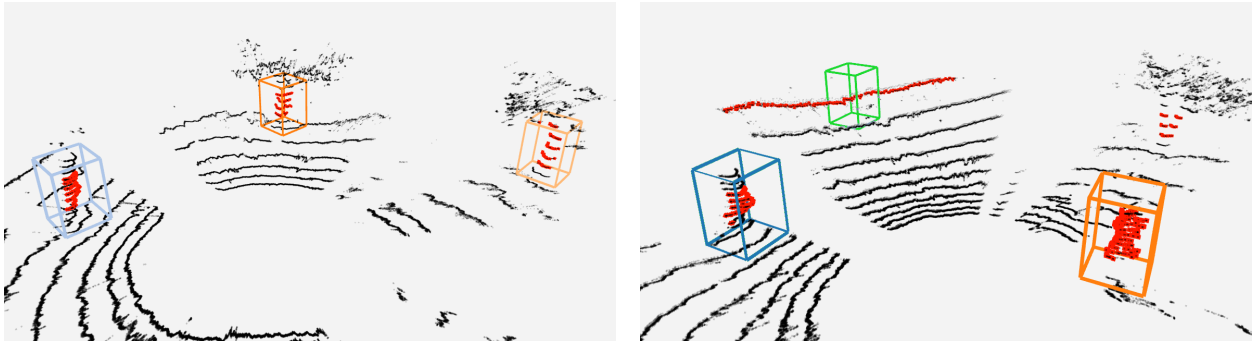
As seen in Table 5.1, the accuracy is within reasonable values, despite the potential for improvement. This can be accomplished with further fine tuning of the filtering and segmentation parameters. The accuracy is computed using the following formula,

$$Accuracy\ (\%) = \frac{No.\ of\ correct\ detections}{Total\ number\ of\ detections} \times 100 \tag{5.1}$$

The accuracy was minimum in run 3, as expected, since the robot travelled in a zigzag pattern, rotating more frequently, which caused the odometry readings to drift and accumulate more error. As such, the received point clouds contained some distortion, resulting

Table 5.1: Accuracy values of the filtering and segmentation stages.

Run #	No. of correct detections	No. of incorrect detections	Total No. of detections	**Accuracy (%)**
1	20	3	23	**86.96**
2	19	2	21	**90.48**
3	19	5	24	**79.17**
Mean	-	-	-	**85.54**



(a) Correct detections (shown with boxes). (b) Incorrect detection (shown with boxes).

Figure 5.6: Example of detections from the filtering/segmentation stage.

in more incorrect detections. Overall, the obtained accuracy values are considered to be acceptable.

5.4.2 Map Building

The main goal of this test was to assess the quality of the topological map constructed using the proposed strategy. In this implementation, nodes correspond to trees, and the main goal was to obtain a topological map in which the node locations matched the real-life locations of the trees as closely as possible. With that in mind, Figure 5.7 shows the obtained topological map overlaid on a satellite view of the orchard field. As expected, the nodes almost perfectly match the tree locations. The small errors are due to errors from the odometry and IMU readings that increase over time and increase even further as the robot rotates. Figure 5.8 shows the localization error between each node and its respective tree as well as the mean localization error overall. Figure 5.9 shows an histogram of the map nodes' localization errors. The distance error for a given tree/node pair is given by

$$error = |\bar{p}_{gt} - \bar{p}_{est}| \tag{5.2}$$

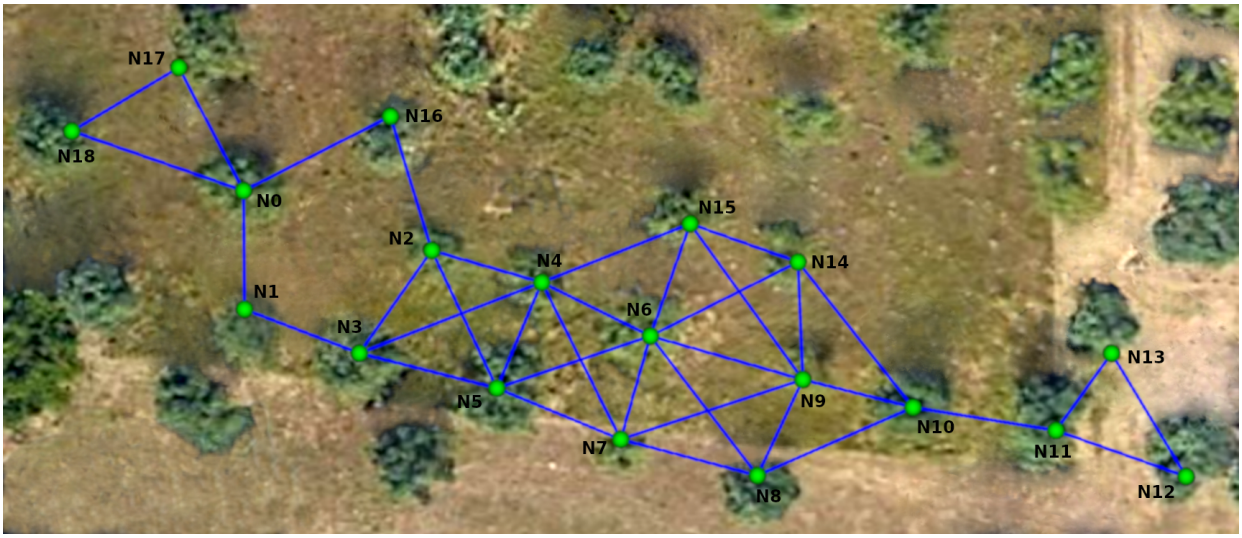


Figure 5.7: Obtained topological map overlaid on satellite view of the orchard field.

where $\bar{p}_{gt}$ is the ground truth position of the tree and $\bar{p}_{est}$ is the estimated position of the node. The ground truth position was obtained by manually retrieving the GNSS coordinates of the trees using Google Maps[9] and converting them to the fixed cartesian coordinate frame, in which the estimated node positions are referenced. The maximum and minimum errors were of 2.14 m and 0.095 m, respectively, while the average error was of 0.865 m. Overall, the results were acceptable, with room for improvement.

5.4.3 Localization

The primary goal of the localization test was to assess the effectiveness of the localization stage. The ground truth for the experiments was the pose provided by the EKF operating in ROS `robot_localization` node, which fused information from GNSS, IMU, and wheel odometry data. To evaluate the performance of this step, the robot's estimated and true positions, as well as the positions of the nodes, were plotted for runs #1, #2 and #3 in Fig. 5.10, Fig. 5.12 and 5.13, respectively.

The topological map was created using data from run #1 before executing the localization method. The resulting map was then utilized to evaluate the localization stage for all three runs, in order to assess the algorithm's robustness and repeatability against various scenarios and viewpoints. The purpose was to see if the robot could get an accurate estimate of its position by using a map built at a different time, in slightly different conditions, and from other viewpoints.

For run #1 (Fig. 5.10), the obtained results were rather decent, with the estimated

[9]See `https://www.google.com/maps/`

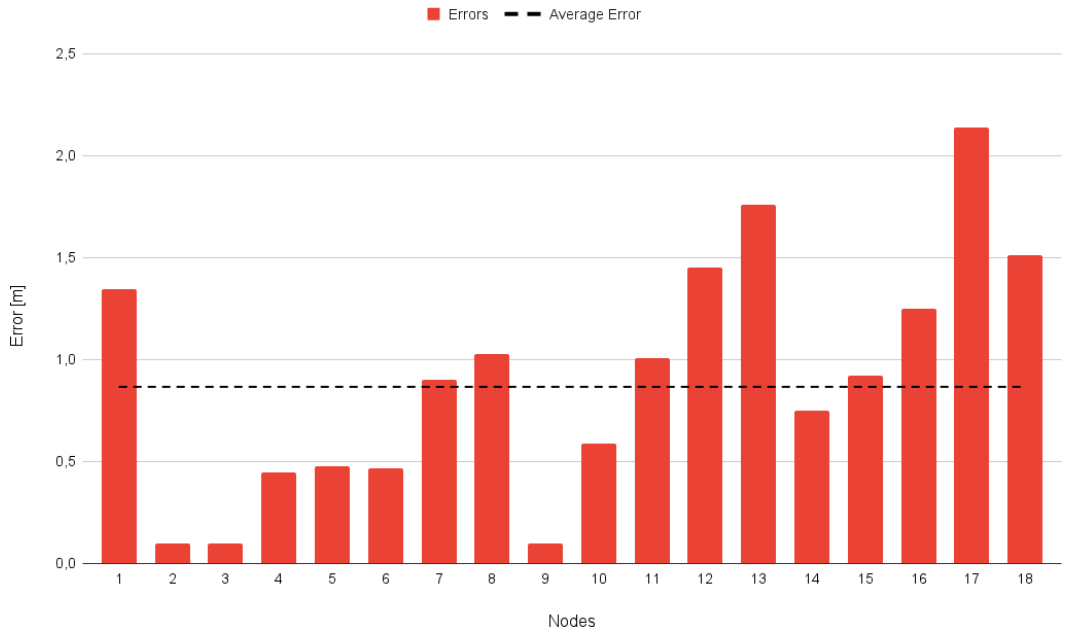


Figure 5.8: Localization errors of the topological map's nodes.

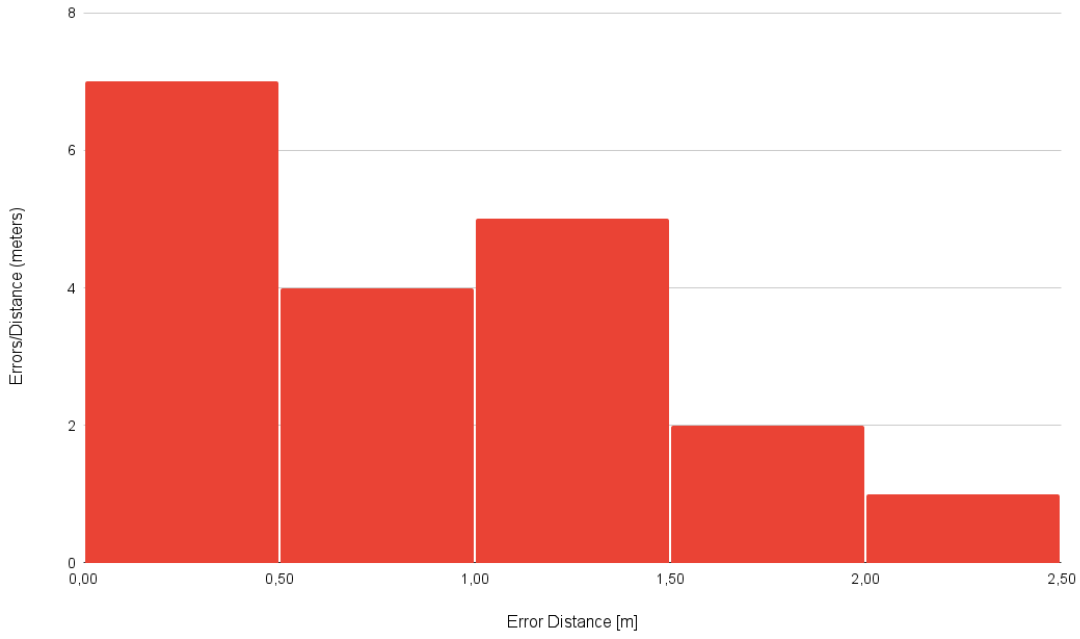


Figure 5.9: Histogram of the localization errors of the topological map.

position being able to approximately follow the ground truth. Occasionally, the estimated pose drifts slightly or even loses track of the real pose. However, the system is able to recover once it detects new clusters and matches them to the correct trees. Figure 5.11 shows the robot's position errors in the x and y directions, and the distance error between estimated and real poses over time, as well as their respective average errors. For this run, the average errors between the robot's estimated and real positions were of 1.6 m and 0.4 m in the x and y directions, respectively; the average distance error between the estimated position and the ground truth was 1.7 m. Because the robot lost track of the true position around the 40 m mark (position A in Fig. 5.10), the comparatively high error in the x direction had a substantial impact on the distance error. As a result, the robot followed the incorrect hypothesis given by the multilateration stage, and the error increased dramatically. Otherwise, when the poses estimated during the multilateration stage are within acceptable limits, the system can closely track the real position. The localization graph for this run additionally distinguishes between when the robot's position was determined using measurements (green in Fig. 5.10) and when only odometry was employed, because measurements were not available or the system lost track of the correct hypothesis (blue in Fig. 5.10). The peaks and valleys seen in Fig. 5.11 might be due to the fact the `robot_localization` node was running at a much higher rate than the RTK-GNSS driver node. Because of this, the EKF in the `robot_localization` node only fused wheel odometry and IMU data when GNSS readings were not available, leading to increased errors (peaks). When GNSS data was included in the fusion process, the error decreased (valleys).

The acquired results for run #2 (Fig. 5.12) were worse than for run #1, as expected, because the localization algorithm was executed with the map produced using run #1. However, the outcomes can be deemed acceptable. The average errors in the x and y directions were 1.7 m and 0.5 m, respectively; the average distance error between the predicted position and the ground truth was 1.9 m. Since the observed viewpoints from the trees differed significantly from the ones used to generate the map in run #3 (Fig. 5.13), the system was unable to correctly estimate the robot's position throughout various parts of the run. Despite this, the robot's position estimate still approximately followed the ground truth.

A convergence analysis was also performed to test how fast the estimation from the localization stage was able to converge to the robot's real position, meaning how quickly the wrong hypothesis were discarded. Since the data was collected in intervals in a start and stop approach, time wasn't used as a convergence measure. Instead the distance the robot travelled until the multiple hypothesis converged into the correct one was considered.

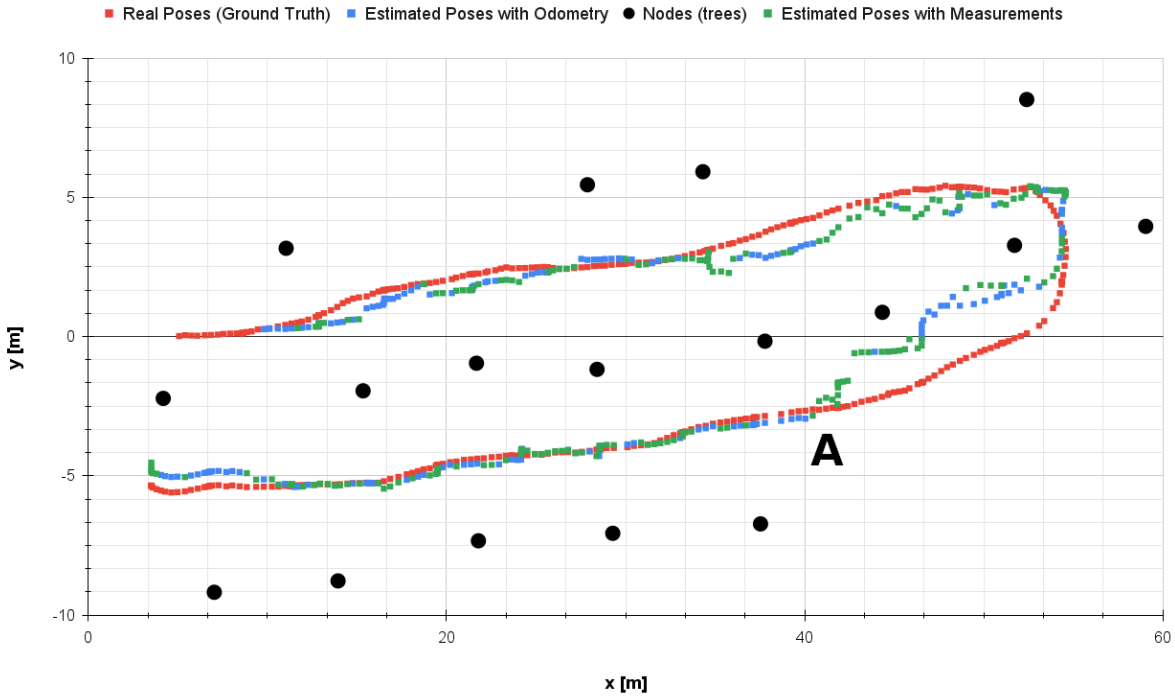


Figure 5.10: Localization plot for run #1.

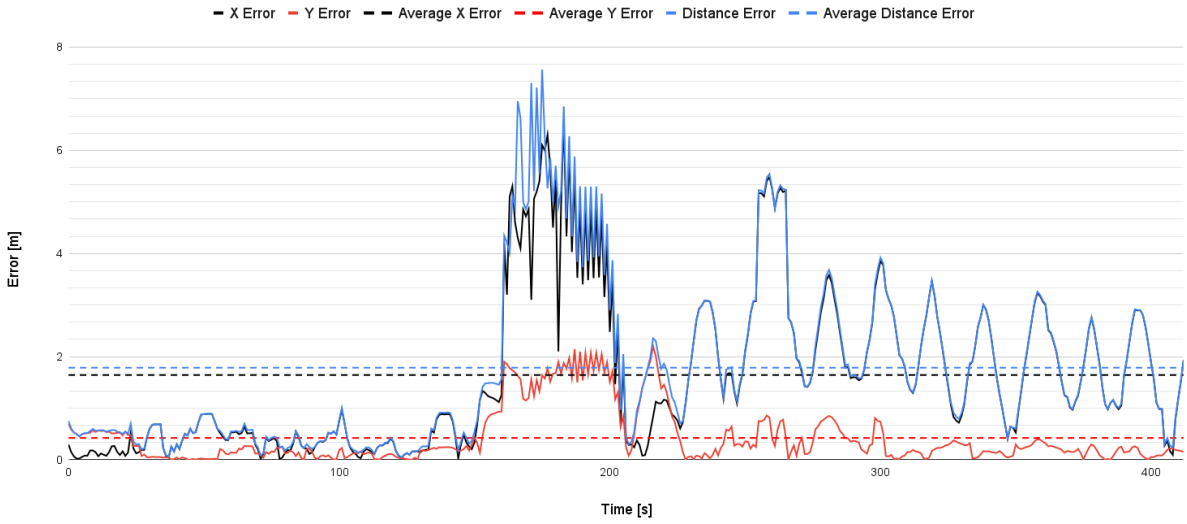


Figure 5.11: Robot's position errors over time for run #1.

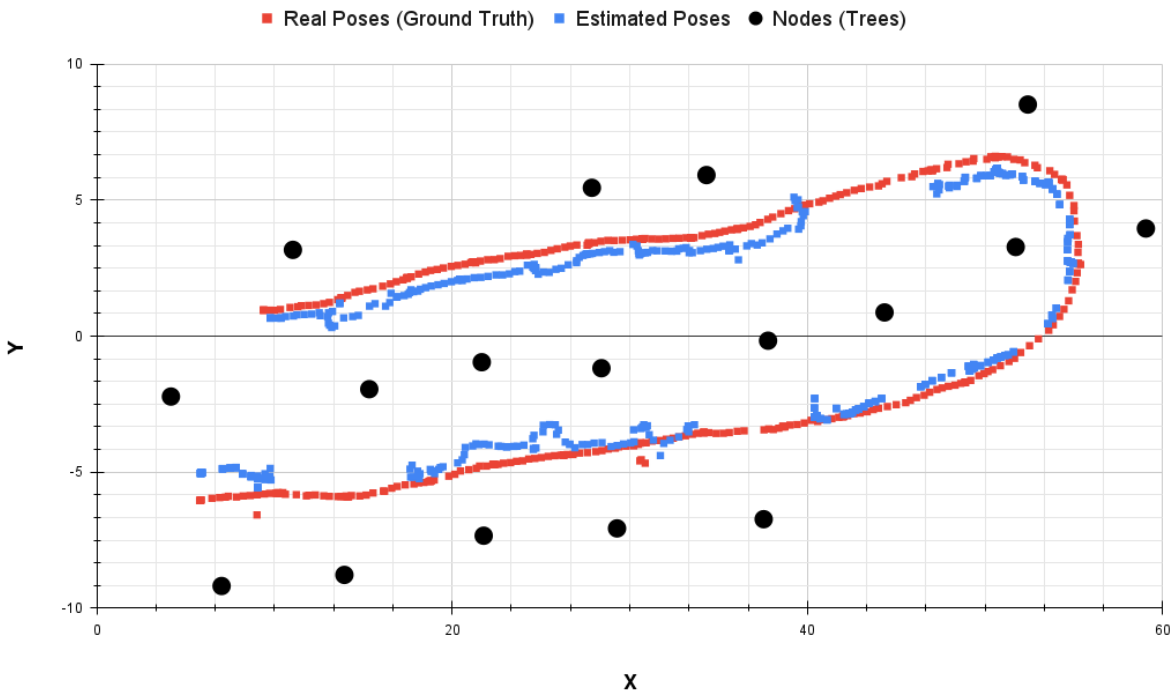


Figure 5.12: Localization plot for run #2.

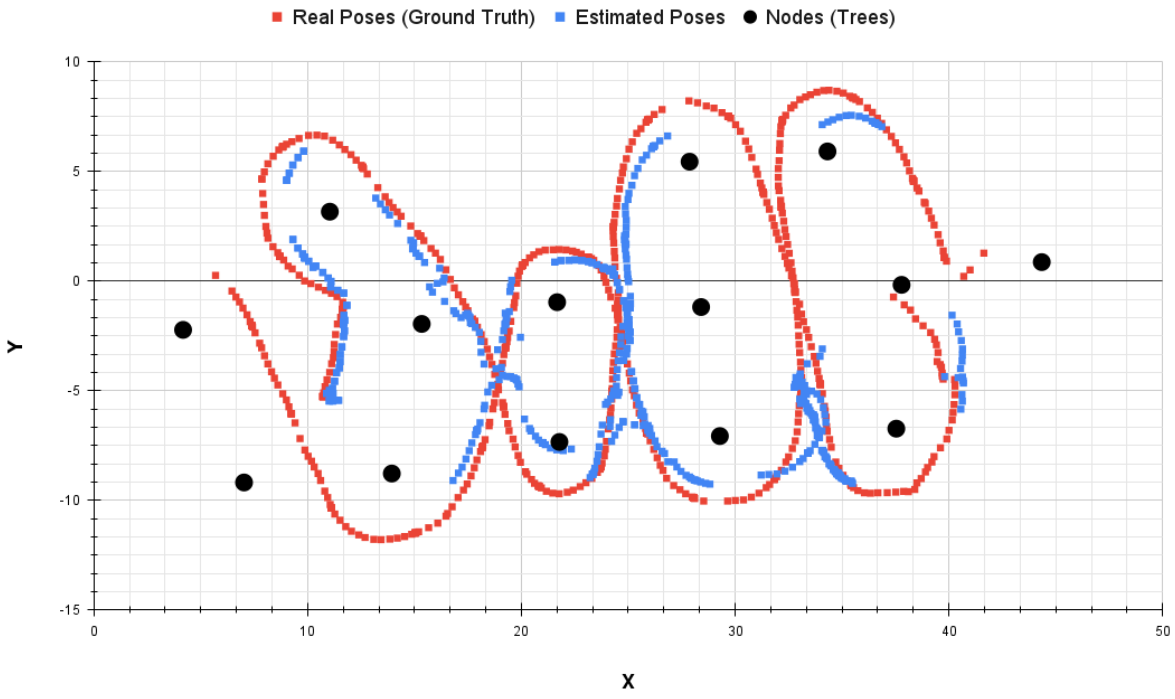


Figure 5.13: Localization plot for run #3.

For 16 different initial positions, the convergence distances were recorded and an histogram was obtained (Fig. 5.14). A normal distribution was also fitted to the collected data. The average convergence distance was approximately 6.1 m, the maximum distance was 12.9 m, the minimum distance was 1.0 m and the standard deviation was 3.0 m.

The convergence distance is not only affected by the initial robot's position but also by the number of trees it detects during the convergence process. Fig. 5.15 shows the average convergence distance for when the system detects 2, 3, or 4 trees while converging to a single hypothesis. When the robot identifies 2, 3 or 4 trees, the average convergence distances are 8.29 m, 5.12 m, and 5.19 m, respectively. As expected, the lower the number of detected trees, the greater the distance the robot has to travel until the system converges to a single hypothesis. The small difference between the average convergence distances for 3 and 4 detections (5.12 m and 5.19 m) is due to the fact that the hypothesis discarding algorithm waits for a fixed number of iterations before discarding an hypothesis. Because of this, if the system generates multiple hypotheses at any given instant, there will always be a minimum distance the robot has to travel before converging to a single hypothesis. Although not shown in the results, the uniqueness of the trees' descriptors also affects the convergence distance, which might explain the similar average distances when the system detects 3 or 4 trees. Overall, the outcomes produced were satisfactory, although there is definitely potential for improvement. The data gathering procedure can always be improved since the acquired datasets could have been of higher quality, allowing for better results.

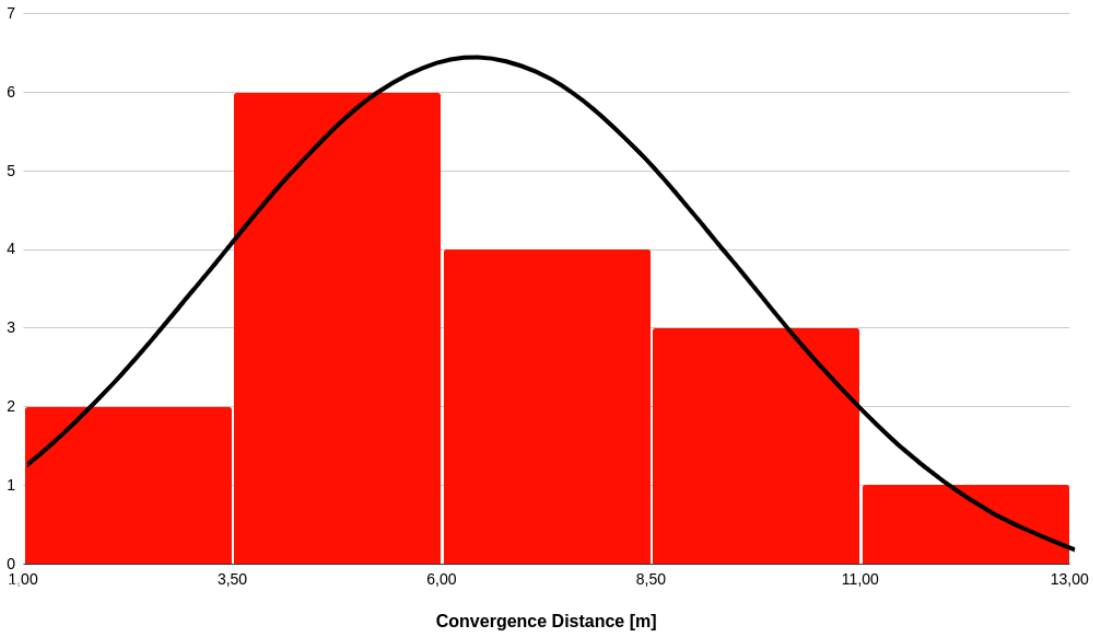


Figure 5.14: Convergence distance histogram (red) and normal distribution (black) for run #1.

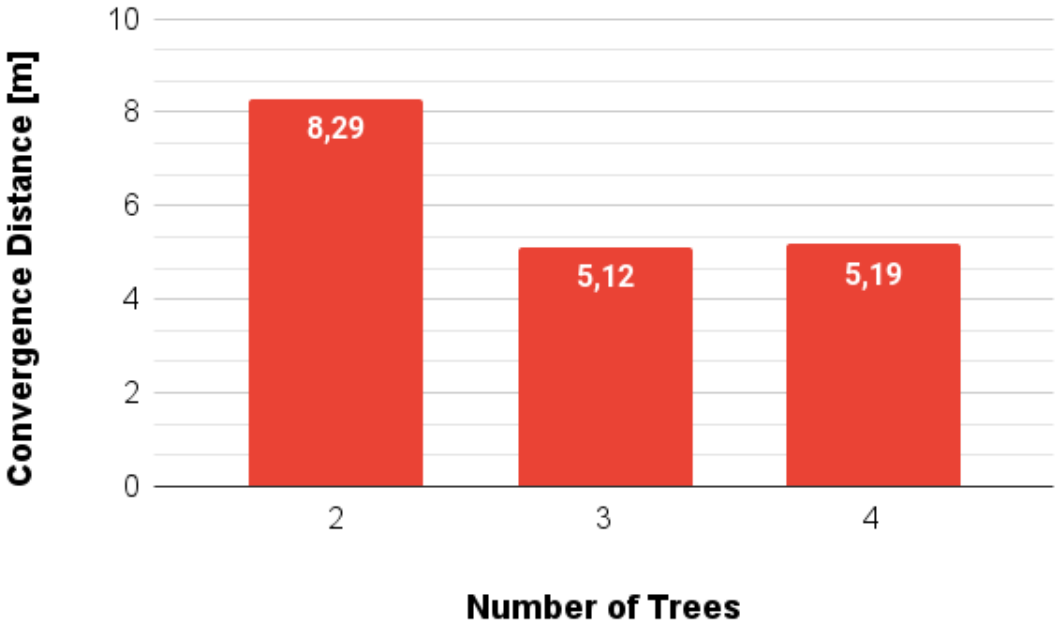


Figure 5.15: Average convergence distance for 2, 3 and 4 tree detections.

6 Conclusion and Future Work

In this dissertation, we proposed a new type of topological map to represent orchard environments and showed how LiDAR can be used to construct such a map. The mapping process used a node matching strategy based on distance and ICP to construct a topological map of an orchard environment, where the trees with their distinctive features serve as nodes and the neighbourhoods of the trees serve as the edges of the map. A preliminary localization strategy was proposed to estimate the robot's position on the map. ROS was used to implement a set of stages in a pipeline which filtered and segmented 3D point clouds to extract the tree trunks of the orchard, build a topological map, and obtain an estimate of the robot's position on the map. The proposed method favours modularity and scalability, allowing for additional information about each node and edge to be easily added to the map structure, increasing the overall robustness and utility of this strategy.

Three sets of validation tests were performed to assess the strategy's performance and demonstrate that it can generate a topological map of the orchard and locate the robot on it. Overall, the objectives defined for this dissertation were successfully accomplished, taking a step towards topological mapping of agricultural environments.

6.1 Future Work

Despite the acceptable results presented in Section 5.4, there are several improvements to be made as follow-up work. Future work will include the addition of more information to both the nodes and edges. Extra node information will include tree signatures, trunk descriptors from multiple views, height descriptors, among others. Additional edge information will incorporate its traversability. This would allow for less memory expensive and overall better descriptors for the topological map's nodes from different viewpoints. In spite of being a preliminary strategy used as a way to quickly validate the map building process, the localization approach can still be improved upon by integrating prediction and measurement

errors into the estimation procedure. It would also be interesting to use other approaches like a Kalman Filter or a particle filter. The whole strategy can eventually be validated in a full Simultaneous Localization and Mapping (SLAM) scenario in an orchard, allowing the robot to build the map while localizing itself as well as update an already constructed map during navigation. Other future work would also include the integration of the proposed topological map structure with typical navigation and path planning algorithms to allow full autonomous navigation. Finally, the implementation of the proposed strategy can be further optimized to improve computation times and memory efficiency, allowing for real-time use in real-world scenarios.

7 References

[1] André Aguiar, Armando Sousa, Filipe Santos, J. Cunha, and Heber Sobreira. Localization and mapping for robots in agriculture and forestry: A survey. *Robotics*, 9, Nov 2020.

[2] Nuha A.S. Alwan and Zahir M. Hussain. Gradient descent localization in wireless sensor networks. In Philip Sallis, editor, *Wireless Sensor Networks*, chapter 3. IntechOpen, Rijeka, 2017.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sep 1975.

[4] Fabian Blöchliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Roland Y. Siegwart. Topomap: Topological mapping and navigation based on visual SLAM maps. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9, 2018.

[5] Pieter M. Blok, Koen van Boheemen, Frits K. van Evert, Joris IJsselmuiden, and Gook-Hwan Kim. Robot navigation in orchards with localization based on particle filter and Kalman filter. *Computers and Electronics in Agriculture*, 157:261–269, 2019.

[6] O. Booij, B. Terwijn, Z. Zivkovic, and B. Krose. Navigation using an appearance based topological map. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3927–3932, 2007.

[7] Pär Buschka. *An investigation of hybrid maps for mobile robots.* PhD thesis, Örebro universitetsbibliotek, 2005.

[8] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. Range Image Understanding.

[9] Young-Ho Choi, Tae-Kyeong Lee, and Se-Young Oh. A line feature based slam with low grade range sensors using geometric constraints and active exploration for mobile robot. *Autonomous Robots*, 24:13–27, 2008.

[10] William R. Cline. Global warming and agriculture. *Finance & Development*, 0045(001):A007, 2008.

[11] Adrien Danton, Jean-Christophe Roux, Benoit Dance, Christophe Cariou, and Roland Lenain. Development of a spraying robot for precision agriculture: An edge following approach. In *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pages 267–272, 2020.

[12] Tom Duckett and Alessandro Saffiotti. Building globally consistent gridmaps from topologies. *IFAC Proceedings Volumes*, 33(27):405–410, 2000. 6th IFAC Symposium on Robot Control (SYROCO 2000), Vienna, Austria, 21-23 September 2000.

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

[14] L. Emmi, Emile Le Flécher, Viviane Cadenat, and M. Devy. A hybrid representation of the environment to improve autonomous navigation of mobile robots in agriculture. *Precision Agriculture*, 22:1–26, Apr 2021.

[15] Gianluca Falco, Mario Nicola, Marco Pini, Gianluca Marucco, Wim De Wilde, Alexander Popugaev, Paolo Gay, and Davide Ricauda Aimonino. Investigation of performance of GNSS-based devices for precise positioning in harsh agriculture environments. In *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pages 1–6, 2019.

[16] David Fernandez-Chaves, Jose-Raul Ruiz-Sarmiento, Nicolai Petkov, and Javier Gonzalez-Jimenez. From object detection to room categorization in robotics. In *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, APPIS 2020, New York, NY, USA, 2020. Association for Computing Machinery.

[17] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, Jun 1981.

[18] Spyros Fountas, Nikos Mylonas, Ioannis Malounas, Efthymios Rodias, Christoph Hellmann Santos, and Erik Pekkeriet. Agricultural robotics for field operations. *Sensors*, 20(9), 2020.

[19] Gustavo Freitas, Ji Zhang, Bradley Hamner, Marcel Bergerman, and George Kantor. A low-cost, practical localization system for agricultural vehicles. In Chun-Yi Su, Subhash Rakheja, and Honghai Liu, editors, *Intelligent Robotics and Applications*, pages 365–375, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[20] Axel Großmann and Riccardo Poli. Robust mobile robot localisation from sparse and noisy proximity readings using hough transform and probability grids. *Robotics and Autonomous Systems*, 37(1):1–18, 2001.

[21] Karoline Heiwolt, Willow Mandil, Grzegorz Cielniak, and Marc Hanheide. Automated topological mapping for agricultural robots. In *UKRAS20 Conference: "Robots into the real world" Proceedings*, pages 27–29, May 2020.

[22] Dirk Holz, Alexandru Ichim, Federico Tombari, Radu Rusu, and Sven Behnke. Registration with the point cloud library - a modular framework for aligning in 3-D. *IEEE Robotics & Automation Magazine*, 22:110–124, Dec 2015.

[23] Andrew Howard, Lynne Parker, and Gaurav Sukhatme. The SDR experience: Experiments with a large-scale heterogeneous mobile robot team. 21, Jun 2004.

[24] J. Jessup, S. N. Givigi, and A. Beaulieu. Robust and efficient multi-robot 3D mapping with octree based occupancy grids. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3996–4001, 2014.

[25] Jin-Woo Jung, Jung-Soo Park, Tae-Won Kang, Jin-Gu Kang, and Hyun-Wook Kang. Mobile robot path planning using a laser range finder for environments with transparent obstacles. *Applied Sciences*, 10:2799, Apr 2020.

[26] Rohan Kapoor, Subramanian Ramasamy, Alessandro Gardi, Chad Bieber, Larry Silverberg, and Roberto Sabatini. A novel 3D multilateration sensor using distributed ultrasonic beacons for indoor navigation. *Sensors*, 16(10), 2016.

[27] Qingqing Li, Paavo Nevalainen, Jorge Peña Queralta, Jukka Heikkonen, and Tomi Westerlund. Localization in unstructured environments: Towards autonomous robots in forests with delaunay triangulation. *Remote Sensing*, 12(11), 2020.

[28] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J. Leonard, David Cox, Peter Corke, and Michael J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[29] Ren C. Luo and Wei-Ting Shih. Autonomous mobile robot intrinsic navigation based on visual topological map. *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, pages 541–546, 2018.

[30] Dimitri Marinakis and Gregory Dudek. Pure topological mapping in mobile robotics. *IEEE Transactions on Robotics*, 26(6):1051–1064, 2010.

[31] Donald Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-D objects by computer. Technical report, Rensselaer Polytechnic Institute, Image Processing Laboratory, Oct 1980.

[32] Rasoul Mojtahedzadeh. Robot obstacle avoidance using the kinect. *Master of Science Thesis Stockholm, Sweden*, 2011.

[33] Andreas Nüchter and Joachim Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[34] United Nations Department of Economic and Social Affairs, Population Division (2022). World population prospects 2022: Summary of results. 2022. UN DESA/POP/2022/TR/NO. 3.

[35] Francis J Pierce and Peter Nowak. Aspects of precision agriculture. *Advances in agronomy*, 67:1–85, 1999.

[36] A. Poncela, E.J. Perez, A. Bandera, C. Urdiales, and F. Sandoval. Efficient integration of metric and topological maps for directed exploration of unknown environments. *Robotics and Autonomous Systems*, 41(1):21–39, 2002.

[37] Vaishali Puranik, Sharmila, Ankit Ranjan, and Anamika Kumari. Automation in agriculture and IoT. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6, 2019.

[38] Pablo Ramon, Robert Bevec, Begoña Arrue, Aleš Ude, and Anibal Ollero. Extracting objects for aerial manipulation on UAVs using low cost stereo sensors. *Sensors*, 16:700, May 2016.

[39] Ankit A. Ravankar, Abhijeet Ravankar, Takanori Emaru, and Yukinori Kobayashi. A hybrid topological mapping and navigation method for large area robot mapping. In *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 1104–1107, 2017.

[40] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[41] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, Oct 2009.

[42] Luís Santos, Filipe N. Santos, Sandro Magalhães, Pedro Costa, and Ricardo Reis. Path planning approach with the extraction of topological maps from occupancy grid maps in steep slope vineyards. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–7, 2019.

[43] Luís Carlos Santos, André Silva Aguiar, Filipe Neves Santos, António Valente, and Marcelo Petry. Occupancy grid and topological maps extraction from satellite images for path planning in agricultural robots. *Robotics*, 9(4), 2020.

[44] Uferah Shafi, Rafia Mumtaz, José García-Nieto, Syed Ali Hassan, Syed Ali Raza Zaidi, and Naveed Iqbal. Precision agriculture techniques and practices: From considerations to applications. *Sensors*, 19(17), 2019.

[45] Redmond Shamshiri, Cornelia Weltzien, Ibrahim Hameed, Ian Yule, Tony Grift, Siva Balasundram, Lenka Pitonakova, Desa Ahmad, and Girish Chowdhary. Research and development in agricultural robotics: A perspective of digital farming. *International Journal of Agricultural and Biological Engineering*, 11:1–14, Jul 2018.

[46] Rafid Siddiqui. *On Fundamental Elements of Visual Navigation Systems*. PhD thesis, Blekinge Institute of Technology, 2014.

[47] S. Simhon and G. Dudek. A global topological map formed by local metric maps. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications*, volume 3, pages 1708–1714, 1998.

[48] Cyrill Stachniss. *Robotic mapping and exploration.* Springer, 2009.

[49] Cyrill Stachniss, Oscar Martinez Mozos, Axel Rottmann, and Wolfram Burgard. Semantic labeling of places. In *International Symposium of Robotics Research*, San Francisco, CA, USA, Oct 2005.

[50] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Ben Talbot, Ruth Schulz, Peter Corke, Gordon Wyeth, Ben Upcroft, and Michael Milford. Place categorization and semantic mapping on a mobile robot. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5729–5736, 2016.

[51] Juan Tardos, José Neira, Paul Newman, and John Leonard. Robust mapping and localization in indoor environments using sonar data. *The International Journal of Robotics Research*, 21, Apr 2002.

[52] André Teixeira, Sedat Dogru, and Lino Marques. LiDAR-based topological mapping of orchard environments. In *Proc. of the 5th Iberian Robotics Conference*, Lecture Notes in Networks and Systems, Zaragoza, Spain. Springer. (Accepted).

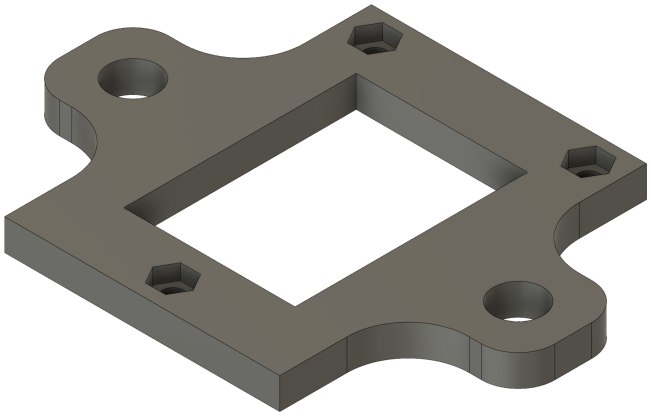
[53] Jozef Vörös. Simple path-planning algorithm for mobile robots using quadtrees. *IFAC Proceedings Volumes*, 28(20):175–180, 1995.

[54] Ulrich Weiss and Peter Biber. Plant detection and mapping for agricultural robots using a 3D lidar sensor. *Robotics and Autonomous Systems*, 59(5):265–273, 2011. Special Issue ECMR 2009.

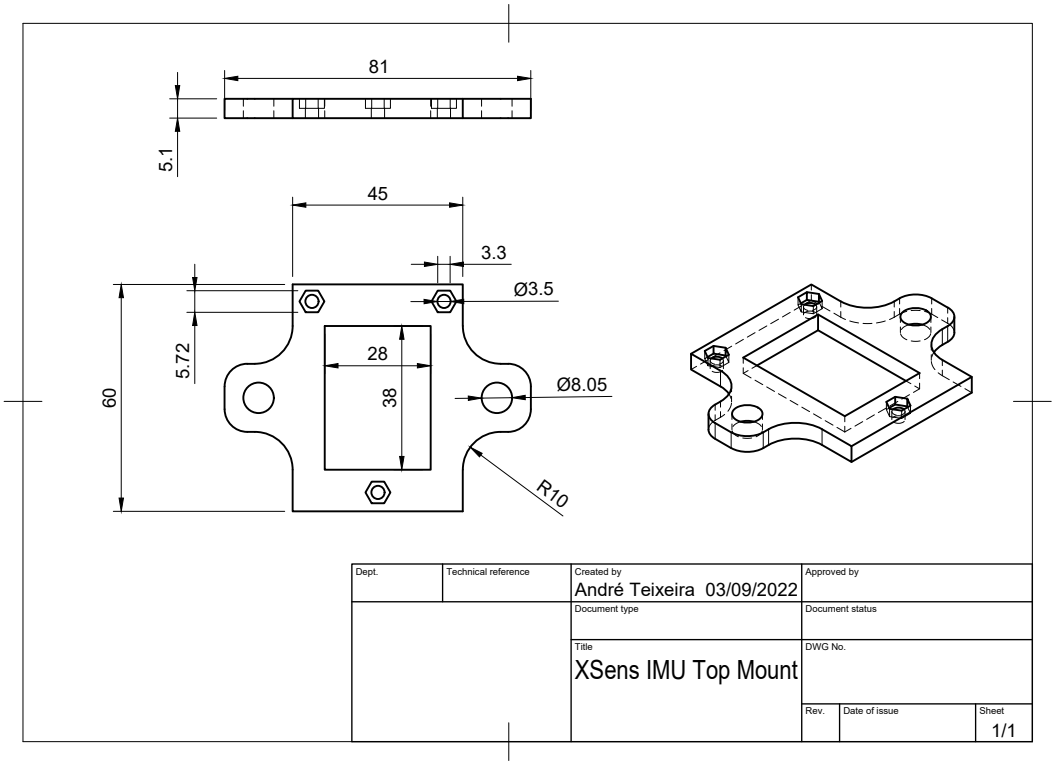
[55] Wera Winterhalter, Freya Veronika Fleckenstein, Christian Dornhege, and Wolfram Burgard. Localization for precision navigation in agricultural fields—beyond crop row following. *Journal of Field Robotics*, 38:429 – 451, 2021.

Appendix A: 3D CAD

This appendix includes the 3D designs and schematics of the IMU mount, the 3D LiDAR mount, the camera mount, and the camera lens cover.

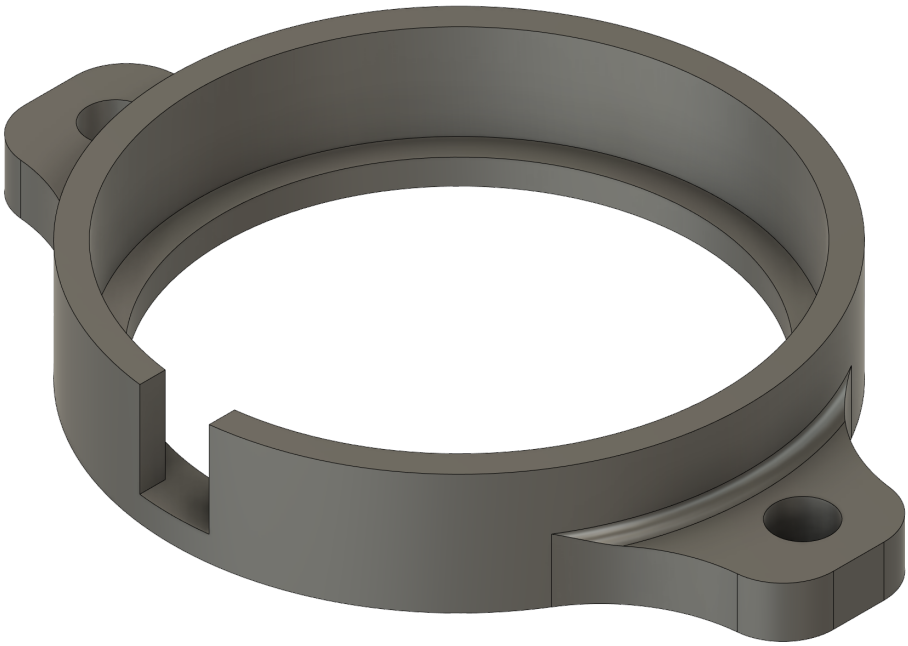


(a) 3D design.

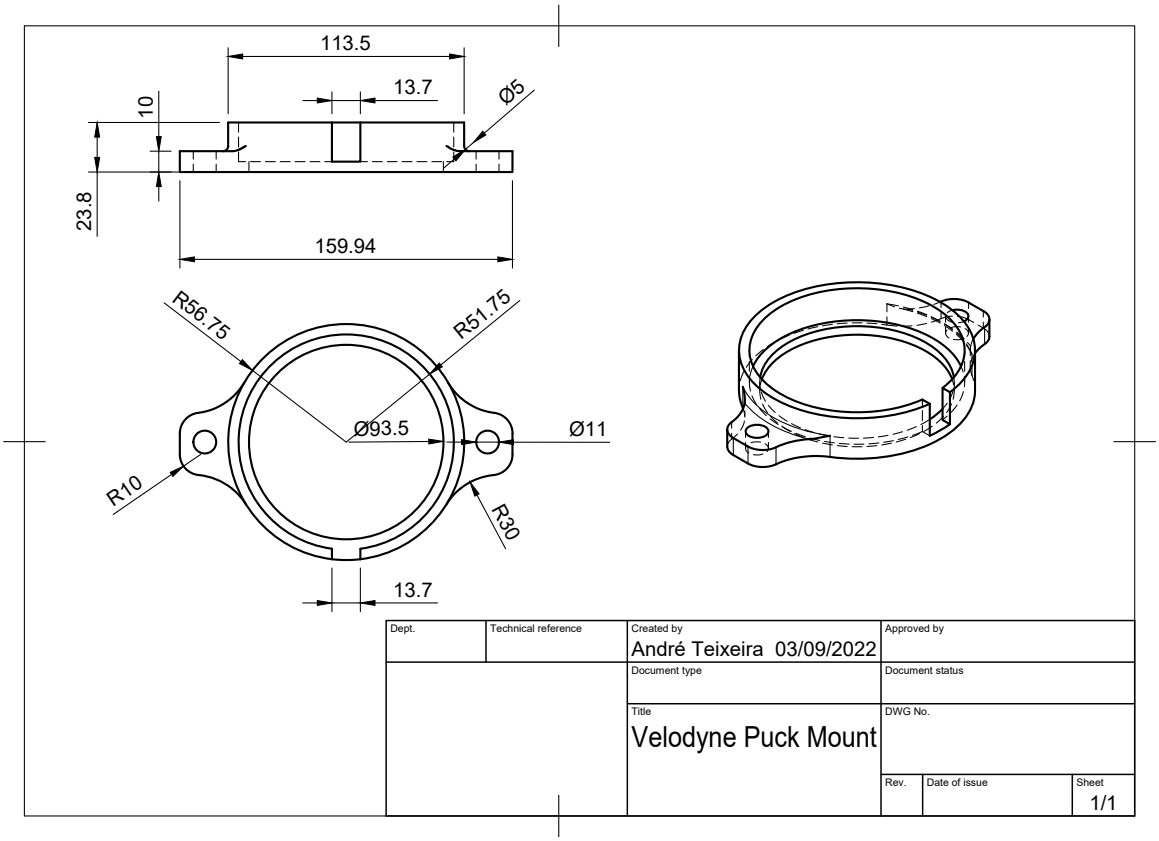


(b) Design schematic (units in mm).

Figure A.1: 3D design and schematic of the IMU mount.

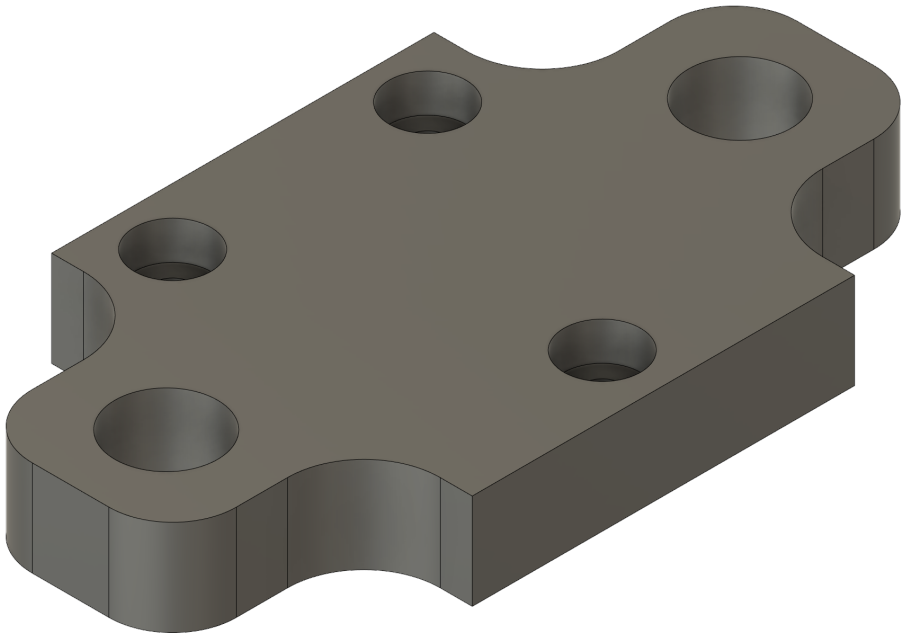


(a) 3D design.

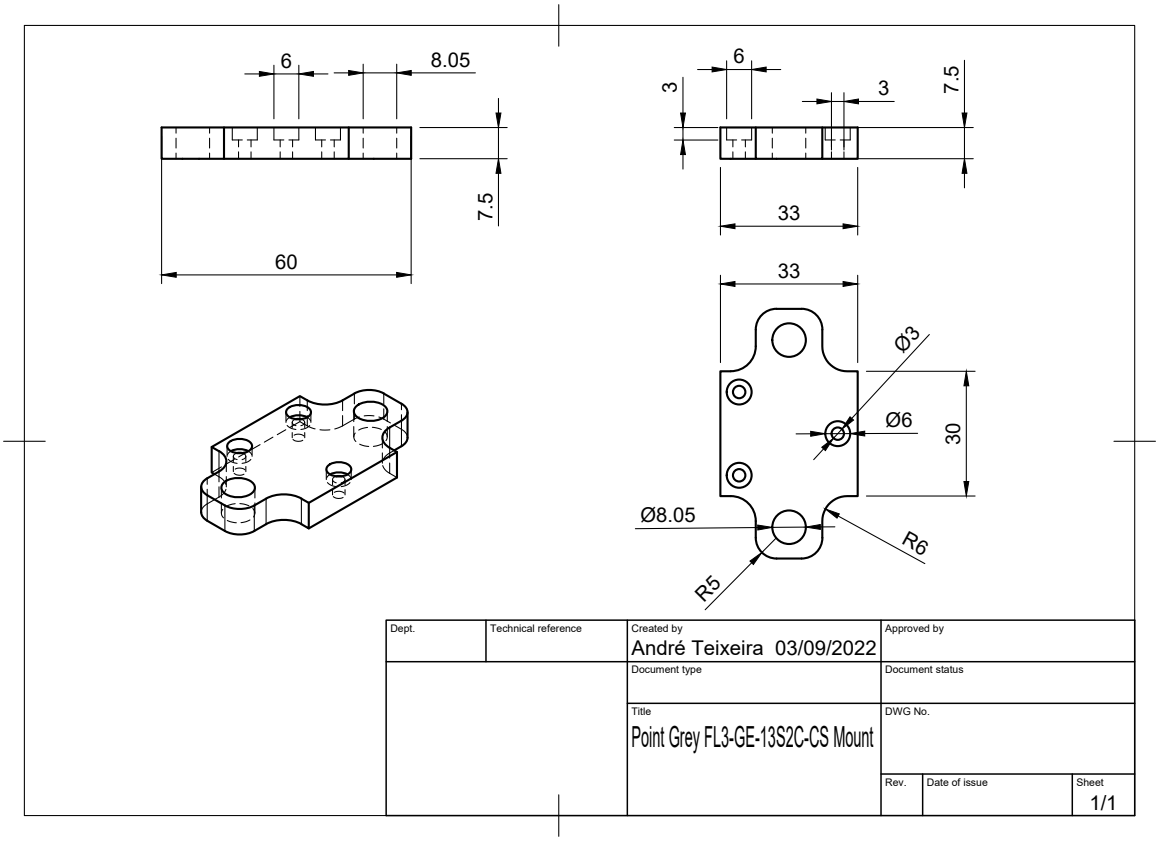


(b) Design schematic (units in mm).

Figure A.2: 3D design and schematic of the 3D LiDAR mount.

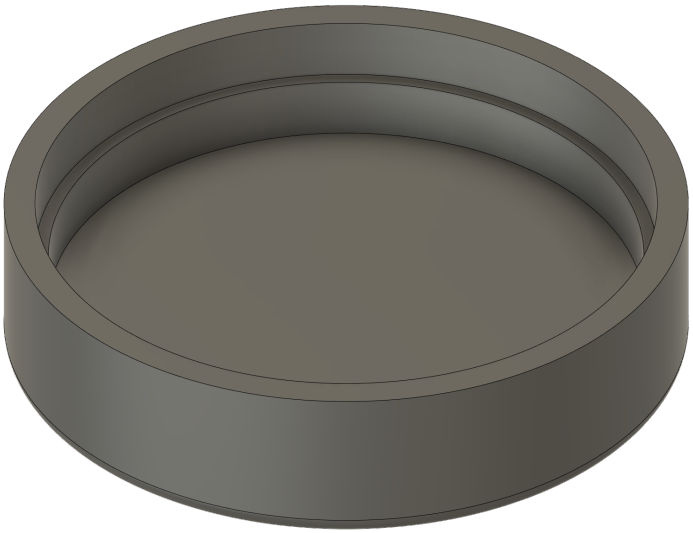


(a) 3D design.

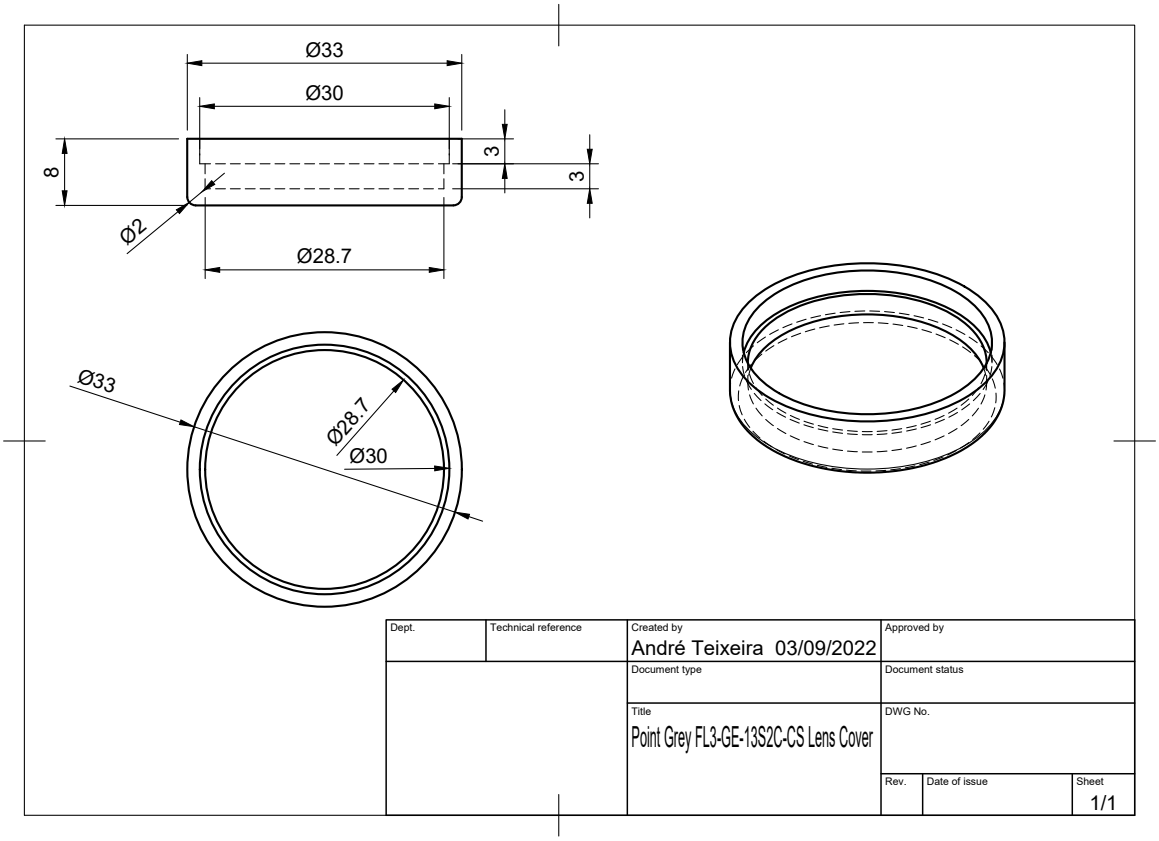


(b) Design schematic (units in mm).

Figure A.3: 3D design and schematic of the camera mount.



(a) 3D design.



(b) Design schematic (units in mm).

Figure A.4: 3D design and schematic of the camera lens cover.