POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An EKF-SLAM approach to plant counting

Tesi di Laurea Magistrale in
Computer Science and Engineering -
Ingegneria Informatica

Author: **Erica Ceriotti**

Student ID: 962792
Advisor: Prof. Matteo Matteucci
Co-advisors: Paolo Cudrano, Simone Mentasti
Academic Year: 2022-2023

Abstract

The expected growth in the global formulation and in food demand bring several challenges for the environment. Precision agriculture is a strategy invented to address the efficient use of resources and to support managerial decisions through the collection of data from fields. It finds its application in agricultural robotics, a discipline which researches solutions for the automation of repetitive tasks and which has seen recent developments due to labour shortages. Regarding the activity of plant monitoring, the need for the reassessment of the asset status is particularly felt in the context of plant nurseries, where management decisions depend on the availability of an up-to-date estimate of the stock availability. Different solutions have been suggested in the past years to for the automation of plant counting. Outdoor environments, however, pose natural challenges to the application of camera-based or infrared sensors, because of continuous changes in the light conditions and irregularities in the terrain. Also, in an orchard context, the presence of tree crowns might render difficult the access to the GPS signal, thus hindering the possibility of creating a map of the field. This thesis proposes an approach to plant counting by using a 3D LiDAR sensor and a Simultaneous Localisation and Mappping (SLAM) algorithm, EKF-SLAM, for the parallel estimation of the map of an unknown environment and the trajectory followed by the robot. In particular, we have formulated a process for plant recognition starting from the point cloud returned by the sensor and we used the position of the detections as landmarks, i.e. points of interest, to correct the location computed by the robot. The system evaluation has been conducted both on a simulated and on real world case.

Keywords: Plant counting, crop monitoring, EKF-SLAM, SLAM, ROS

Abstract in lingua italiana

L'attesa crescita nella popolazione mondiale e nella richiesta di alimenti comportano diverse sfide per l'ambiente. L'agricolutira di precisione è una strategia creata per far fronte ad un uso efficiente delle risorse e per il supporto di decisioni a livello di management sfruttando la raccolta di dati dai campi. Questa tecnica trova applicazione nella robotica per l'agricoltura, una disciplina che ricerca soluzioni per l'automazione di lavori ripetitivi e che ha visto recenti sviluppi per via della mancanza di lavoratori. Per quanto riguarda l'attività del monitoraggio delle piante, i vivai, nello specifico, necessitano di stimare spesso la quantità di prodotto disponibile, in quanto le decisioni per la gestione della domanda dipendono da quanto la stima è aggiornata. Allo scopo di automatizzare il processo di conteggio, differenti soluzioni sono state proposte negli anni passati. Tuttavia, gli ambienti esterni impongono delle difficoltà nell'applicazione di sensori come camere e infrarossi, per via dei continui cambiamenti nelle condizioni di luminosità e delle irregolarità che il terreno presenta. Inoltre, nel contesto di un frutteto, la presenza delle chiome degli alberi può rendere arduo l'accesso al segnale GPS, rendendo poco possibile la creazione di una mappa del campo. Questa tesi propone un approccio al contenggio delle piante tramite l'uso di un sensore di tipo 3D LiDAR e di un algoritmo di SLAM, EKF-SLAM, per la stima simultanea della mappa di un luogo sconosciuto e della traiettoria percorsa dal robot. In particolare, abbiamo formulato una processo per il riconoscimento di piante a partire dalla nuvola di punti ritornata dal sensore e utilizzato le posizioni delle rilevazioni come punti di riferimento per correggere la posizione calcolata dal robot. La valutazione del sistema è stata condotta sia nel caso di un ambiente simulato che reale.

Parole chiave: Conteggio piante, monitoraggio colture, EKF-SLAM, SLAM, ROS

Contents

Introduction

Agricultural robotics is the field regarding the construction of robots for agricultural purposes. The shortage in labour and the need to perform operations in an efficient way has led to the development of different type of systems, which can be divided into stand-alone robots and in robotics implements of existing agricultural machines [20]. The ones belonging to the former category are the most employed by farmers and include both general purpose self-driving platforms and autonomous robots specialised activities such as spraying and harvesting. Also, some smaller systems have been designed for data collection purposes and weeding. The latter class, instead, refers to tractors equipped with external devices to perform tasks such as transplanting, lettuce thinning, fruit harvesting and vine pruning.

In particular, we are interested in the automation of plant monitoring for the assessment of the number of plants in a field. The temporal and economical cost required by manual counting is felt is especially in the context of plant nurseries, where the management decisions depend on the amount of crop available. Indeed, because of natural factors, small plants are subject to a premature perish and therefore the stock monitoring needs to be re-executed several times.

To research this topic, we started by reviewing the existing approaches (Chapter 1), we studied some algorithms and tools which are at the core of our work (Chapter 2), we proposed our solution to the problem (Chapter 3) and finally we performed an evaluation of the system (Chapter 4).

1 State of the art

1.1. Motivation

According to UN, the world population is expected to reach more than 9 billion people by 2050 [10]. To meet the food demand, the crop production would need to increase by 25-70%, while still maintaining environmental sustainability [6]. Indeed, the intensification of land use requires a growing amount of fertilizers, of which nitrogen is one of the most used and most dangerous when lost to waterways [6][18].
To counter these challenges, scientific advancement has led to the technique known as precision agriculture, i.e. "a management strategy that gathers, processes and analyzes temporal, spatial and individual data and combines it with other information to support management decisions according to estimated variability for improved resource use efficiency, productivity, quality, profitability and sustainability of agricultural production." [8]. The need for manual labour required by such practice, however, clashes against the shortage of workers [19], and thus robotic platforms have been developed to help farmers. Generally speaking, robotics have been introduced into a variety of agricultural tasks such as weeding, seeding, harvesting, spraying including crop monitoring, which well fits the description of precision agriculture. In particular, it comprehends two different tasks, i.e. phenotyping and plant vigor monitoring [4]. While the former regards the collection of plant traits, such as morphology, the latter addresses the retrieval of information about the plant status. These problems have been assessed in the analysis of cereal crops and orchards, but plant nurseries could benefit from their usage as well, since their economical decisions directly depend upon the health and availability of stock. In particular, the acceptance, or rejection, of a customer order, is a choice which needs an up-to-date estimate of the asset to be taken, and thus the counting of plants has to be repeated many times, since seedlings can be subject to premature death and diseases. Especially in big realities, the manual counting would not be feasible, as it represents a time-expensive, repetitive and costly task. The automation of such job could therefore decrease the retail price and increment the nursery profit.

1.2. Literature Review

The presented project aims at creating a stand-alone system for plant counting while performing other existing agricultural operations, meaning that it operates at plant level. For this reason, the literature about Unmanned Aerial Vehicles (UAV) has been excluded, in favor to Unmanned Ground Vehicles (UGV). In addition, our context of operation is that of an orchard nursery whilst many of the works realized so far are related to crops such as maize, sorghum and corn. Nonetheless, they have been included in this review to present a variety of approaches that have been taken to perform plant counting. The analysed literature hereby presented is intended as a survey of different sensors and algorithm developed for the task at hand. It has been divided per type of sensor, resulting in three main categories: Mechanical and Infrared, Camera and LIDARs. Some of the articles might cover more than one type of sensor, in which case they have been assigned to their prevalent class.

1.2.1. Mechanical and Infrared Sensing

Mechanical and infrareds sensors work according to different physical principles but share some common characteristics. Indeed, both of them generate a variation in voltage output when triggered which then can be converted into digital signals. Moreover, for the results to be accurate, they might need to be mounted at sufficient heights because the type of information returned makes the distinction between plants and weeds very difficult. In some cases they might also be needed to surround the row by the two sides, i.e. back and front of trees, a cumbersome requirement for adults trees especially.

Starting from mechanical sensors, one of the firsts designed for plant counting consisted in a spring-loaded rod attached to a rotary potentiometer [1]. The deflection of the rod causes the output voltage to change with respect to the rest state. The aim of the paper was to estimate the corn plant population and the results showed an average difference over the manual count of less than 5%.

A more recent application [9] explored the possibility of using an active InfraRed (IR) Sensor. Differently from a mechanical approach, it is designed to measure the distance from the target, with a working range from few centimeters to 1-5m depending on the type of sensor. To detect an object, two components are being used: a Light Emitting Diode (LED) transmitting an IR pulse and an IR receiver sensitive to the emitted signal wavelength. The voltage returned by the sensor depends on the intensity of the light return and a calibration might be needed to assign the right voltage to the sensed dis-

Figure 1.1: Sensor and reflective plate placement during field testing (taken from [9])

tance. Although they are more informative than mechanical sensors, infrareds capture quality can degrade with sunlight, which renders them less suitable to be used in outdoor environments.

An application of an IR based system is proposed in [9], where the sensor was coupled with a solid plate to exclude readings from other rows, as shown in Figure Figure 1.1. Since the sensor is only able to detect the presence, or absence, of a subject, it was mounted at a precise height to avoid weed interference. The collected data was filtered to exclude the background plate and other irrelevant measurements, for example those nearer than the location of the row. Stalks were then visually identified by comparing the signals to the estimated stalk width at the positioning height of the infrared. The tests were conducted on corn fields and demonstrated an accuracy similar to the one of the aforementioned work.

In [5] a study was conducted to establish the performance of a light curtain sensor against a LiDAR for recognition and classification of alive and dead trees. Light curtains creates an invisible barrier by combining a set of LEDs or lasers emitters, and a set of photoelectric receivers, parallel to them. These sensors are usually employed to delineate safety regions around factory machineries. When the light beams are interrupted, a signal is sent so that the active equipment can stop and allow the human operator to approach it without being in danger.
In the analysed case, a custom, hand-driven cart was equipped with a 2D LiDAR and an IR light curtain vertically aligned as shown in 1.2 and driven upon the plant rows of an almond nursery. Also, a GNSS receiver was added for ground truth recording and a rotary encoder wheel with a timing chain was used for odometry, in the recognition process and

for triggering the registration of light curtain returns.
The light curtain was constituted by 4 IR emitters and receivers distributed perpendicularly with respect to the terrain to cover a determined height. The recognition process started with the detection of a presence by the first, lowest sensor, with the subsequent check of the status of the above infrareds, within an encoder window. If all the first tree sensors were triggered, the detection was considered as valid and registered as long as its medium encoder value was far enough from the previously registered one. The distinction regarding the aliveness of the tree was based on the number of readings inside an encoder window centred in the encoder value of the activation of the lowest positioned detector. The value was then compared against a threshold to decide upon the state of the tree.
The LiDAR sensing, instead, posed some challenges relative to the small shape of the trees. To cope with the noise produced by the sensor, the authors needed to perform filtering at different levels, and add up to 6 hand-tuned parameters for the recognition only. The results showed that the simplest sensor, i.e. the light curtain, performed better in both identification and classification tasks.

Figure 1.2: The cart used by [5] with highlights of the mounted sensors. On the top left, the encoder coupled with a timing chain, mounted on the wheel. On bottom left, the tree stem detection. On the right the arrangement of the LiDAR and the light curtain. (taken from [5])

The practical application of all the presented systems is subject to the positioning of the sensor. In particular, they had to be placed at a certain distance from the plants to perform a correct detection and at a certain height to avoid ground, leaves and weeds. However, irregularities in the terrain can cause the sensors to move vertically and thus hinder the sensing of plants. Especially in [5], a custom built cart was tested but the manual movement does not take into account the vibration which might occur during with the use of actual agricultural vehicles. Also, the identification process heavily relied

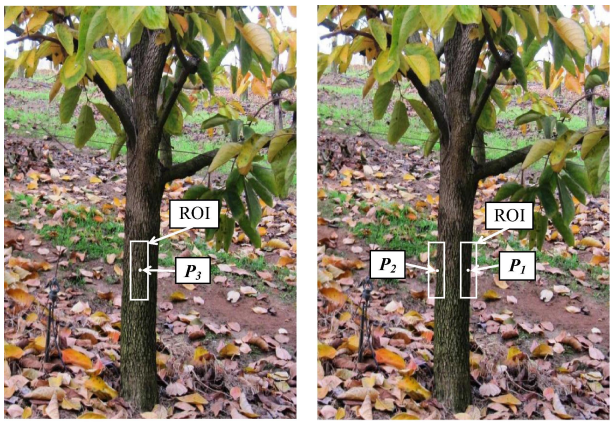
on the use of an encoder for the identification, which can easily fail in case of a slippery terrain.
Moreover, both of the IR works involved the displacement of the sensor from the top of plants which might become more difficult to maintain with their growth.

1.2.2. RGB Camera Sensing

Compared to the precedent class, RGB cameras are a more complex type of sensor. Indeed, they can capture the lights, colours and shapes producing a 2D representation of the environment in the form of a pixel array. Their diffusion in electronics for mass production rendered them cheaper, with a very low cost with respect to LiDARs. Digital images acquired with such sensors cover an important part in our everyday life and are becoming more and more important in informatics, especially in the artificial intelligence field, where they are at the object of Convolutional Neural Networks (CNNs) training. The colour and contour information are some of the core features used by models for differentiating between classes of objects in recognition tasks. However, different light conditions might affect the quality of the subjects representation in images produced by cameras and thus pose an important challenge in the training of classification models.

The authors of [7] presented a light-weight robot under the name "TerraSentia" , which was equipped with a consumer-grade camera for real-time corn plants detection and counting. In order to reduce the computation cost and achieve a fast processing rate, a CNN of the MobileNet family was employed. Moreover, since a network require a dataset of thousands of images for training by scratch, a pre-trained network on the ImageNet Large Scale Visual Recongnition Competition (ILSVRC) [2] dataset was used as base. Transfer learning was performed to adapt the network for corn identification. The last fully-connected layer of the network was substituted with a Support Vector Machine (SVM) classifier to further reduce the need of new images. The project aimed at counting corn plants in different growth stages, therefore a total of 4 datasets were collected, each with less than a thousand images. Differently from Infrared and mechanical sensors which produced one or more near impulses per detection, with cameras the same plant can be recorded by multiple frames and multiple plants might be visible at the same time. To cope with this issue, a region of interest (ROI) mechanism was proposed, which, in case of multiple plants in the ROI, computes the effective number dividing the translation between two consecutive frames marked as positive by the width of the ROI. The tests resulted in an R index, also named Pearson Correlation Coefficient, equal to 0.96, which indicates an high correlation between the automatic and manual count with.

A different approach to image analysis was taken in [15], where the possibilities of fusing 2D LiDAR data with camera data was explored. The study was conducted in a persimmon orchard where trees were interleaved with objects such as poles and tree supports which were treated as separate classes with respect to trees. The methodology followed for discerning them was composed by various stages. First of all, the trunk edges were detected by comparing the LiDAR points against the expected trunk width distribution, which was formulated according to field observations. The visible edges were projected in the camera frame and used as centre for two ROIs. A third ROI was created by taking the middle point between the previous two and used to extract the trunk colour as in Figure 1.3. To reduce the effect of light changes, the Hue value was used as a distinctive characteristic and a distribution was built around the medium hue extracted from the taken samples. Both the width and the colour were tested simultaneously and a positive result indicated the presence of a tree. Based on the width, then, the trunk could be classified as thin or regular. An element passing the width test but not the color one was labelled as a small post. The other two ROIs, instead, were employed in the recognition of potential large posts, which failed both the previous set conditions, but which are straight, from unknown objects. All the threshold values applied in the three tests were empirically determined, by collecting data from 100 trees for the first parameter and 100 for the other two. The hue mean was evinced by photos taken during sunny and cloudy days and the results demonstrated that the union of the two distributions was the best performing. In the test phase, the sensors were mounted on a CoroWare Explorer (CoroWare Inc, USA) and the LiDAR was placed at such an height so to avoid weeds, fallen leaves and other sources of disturbance on the ground. The errors occurring in the classification were mostly due to a fail of the hue thresholding, which showed to be more reliable when the value for the misclassified elements was included in the distribution. Indeed, the accuracy after the forth test showed an increase from 91.60% to 96.64% with a reduction of the number of trees classified as objects.



(a) ROI for colour detection. (b) ROI for edge detection.

Figure 1.3: ROIs involved in the trunk recognition process (taken from[15])

The same authors employed the proposed detection algorithm for building a map of the orchard and perform EKF localisation [16], using the trees as natural landmarks to guide the robot without a GPS signal. Indeed, the availability of precise positioning data is not always assured in orchards context, since the tree canopies might occlude the satellites vision, which happened during the record of ground truth data in the work itself.

This last presented paper did not perform real-time detection of plants, but it was still taken into consideration as an interesting alternative to CNNs which require a great amount of computational power to operate. Indeed, even if it is not directly stated in the paper, the TerraSentia robot is marketed as capable of functioning for more than three hours [11], which is not so much when applied in large fields counting. Nevertheless, the color threshold used in [15] demonstrated to be not very resilient to change in lights and required the inclusion of the values for missed trees for an increase in performance.

1.2.3. LiDAR Sensing

LiDARs (Light Detection And Ranging) are Time-Of-Flight (TOF) sensors composed by a light emitter and a receiver as in Infrareds, with the advantage that they can detect objects at a greater distance and offer a greater measurement quality. The sensor returns a representation of the elements in the environment in the form of point clouds, where each point brings information about the distance and displacement of the subjects with respect to the sensor. The location of the reflecting point is computed based on the time that it takes for the emitted light pulse to return back to the sensor.
There exist different types of LiDARs depending on the spatial dimensionality of the points collected. Starting from the simplest one, 1D LiDARs use a single and fixed beam to return the distance from an object. 2D LiDARs, instead, make the light rotate so

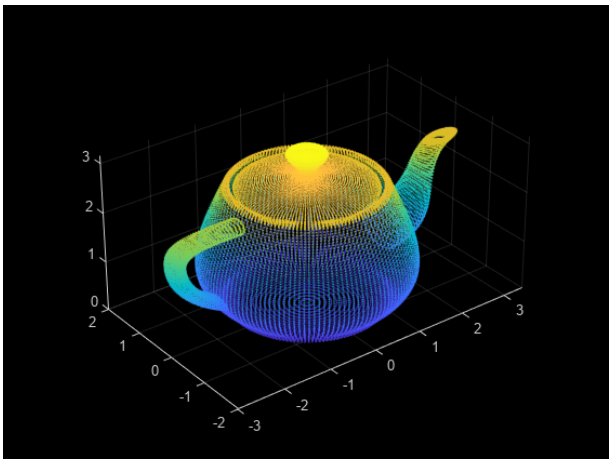


Figure 1.4: Example of a three dimensional point cloud representing a teapot[1]

Figure 1.5: The custom cart built in [22]. The LiDAR is placed on the left of the cart and inclined of 20° with respect to the ground.

that they can detect also the bearing, or angle from the sensor. Finally, 3D LiDARs are of a more advanced type as they stack many rotating beams vertically to also map the z coordinate. The point cloud registered by the sensor is tree dimensional, as the one showed in Figure 1.4. Their resolution depends on the number of planes involved, which usually go from 16 up to 128.

As an example of the second category, in [22] the authors mounted a 2D LiDAR on the side of a four wheeled cart, as depicted in Figure 1.5, along with a shaft encoder employed in localisation, for the exploration of two corn rows at different growth stages. The identification of stalks was performed within a fixed height window to reduce at minimum the weeds appearance. A slightly modified version of DBSCAN (see Algorithm 2.1) was applied to the filtered points, where a variable radius distance for neighbours

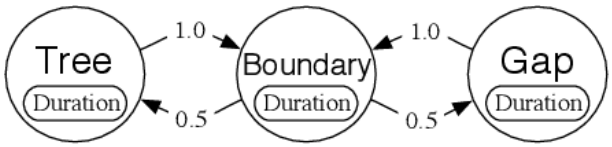


Figure 1.6: The Hidden Semi-Markov Model (taken from [22])

search was used and core points were not necessarily surrounded by a minimum number of neighbours. This last characteristic was introduced as most of the sensor returns were constituted by border points, so it was preferred to avoid marking them as noise to then provide a later reconsideration of class. The formed clusters were discarded if they did not meet a minimum and a maximum number of points.
In this study too, we assist to the problem of recognizing the same plant through different scans. The solution consisted in the projection of clusters position back through past detections by subtracting the distance between the current and the precedent encoder reading of recorded scans. If a cluster was found within ± 6cm of the computed previous location, then it was considered to be of the same plant. As it was conceived, this method required the memorization of scans up to 30 previous steps. At the end of the matching if a cluster was not found it was considered as noise, unless it was at the beginning of the sensor's field of view in which case it was marked as new. The final location of plants was found as the mean of their stored coordinates.
The main source of misdetections occurring in the results was due to leaves being detected as part of the plant clusters as they made the number of points exceed the maximum expected threshold. False positives were caused by weeds presence, which showed the system to not be reliable, especially on uneven terrain where bumps of the cart can cause more elements, aside from stalks, to appear in the analysed window. Localisation inaccuracies, instead, were caused by the uneven ground surface, the pre-processing step of the encoder data and the merging of leaves and stalk clusters.

Passing to 3D LiDARs, a more probabilistic methodology was at the core of [22], where such sensor (referred as Ladar) was used to detect the volume of trees canopies and adjust the flow of a sprayer tractor in an orchard of adult citrus trees. The secondary objective of the work was to count and generate a tree inventory for operations like yield prediction. The proposed solution implied the accumulation of LiDAR data with GPS-INS information to form a 3D map of the environment. The spray application was meant for tree canopies only, hence, in order to exclude weeds, a Markov Random Field (MRF) was used to obtain a consistent estimation of close ground areas by combining the lowest part of the map with a prior distribution of terrain smoothness. The map was divided into voxels, i.e. volumetric 3D pixels, and the number of hits and pass-through was computed for each

of them to measure the density of regions. The released spray flow was then adjusted to match the canopies density to reduce the chemicals usage.
Instead of looking at more than one tree at a time, the authors focused on the various stages visible through an orchard row, i.e. trees, borders and gaps. Specifically, they assigned to each 20cm slice in the collected LiDAR data one of the above classes and modelled the permanency and transitions between them as a Hidden Semi-Markov Model (HSMM) as in Figure 1.6. For computing the Gaussian distributions of the duration characterizing each state, they relied on the expected tree distance in the rows given by the location of the robot and a pre-existing GIS map of the orchard. The fusion of prior knowledge with the modelled probability of states given the observations allowed the system to perform well even in case of walls of trees and without the need to assess any fixed threshold. Newly replanted trees however were not detected since they were classified as weeds because of their height. Large replants areas caused the overestimation error to increase up to 15%.

Last, but not least, we present the study [21] that inspired our work the most. It was produced as part of the BoniRob project and its goal was to detect and map plants for navigation purposes. A 3D LiDAR was mounted in front of the robot directed towards the ground and was coupled with a GPS sensor to mark the plant location. The RANSAC algorithm (see Section 2.1.2 for details) was employed for the estimation of the ground plane and the found equation was refined with least square. To cope with the possibility of bad, or absent, plane detection, the authors propagated the plane by using a Kalman filter (Section 2.2.2). Once the estimated terrain surface was filtered out, the remaining points were clustered with a region growing algorithm. To distinguish the plant clusters, they formulated a statistical model based on a Gaussian distribution using the expected width, height, and number of points characterizing plant clusters and the average distance μ_d between two consecutive plants. The probability of a cluster to be a plant was found by comparing it to the expected box size and number points. In the case two clusters were found to be nearer than the estimated standard deviation between centres they were merged.
To avoid the same plant to be detected multiple times and to improve plant localization, the authors associated each positive detection to a different tracker. As in [17] they matched the trackers based on a threshold: if the distance of a cluster was found to be less than $\frac{1}{2}\mu_d$ from a tracker, it was considered to belong to the same plant and its position was updated. The trackers were deleted once the associated plant was not visible anymore or no new update was made in a period of time. Using this system, they managed to improve the detection rate from 60% in field trials to 80-90% with the majority of the

problems given by plants not corresponding to the created model.

Despite being more costly than cameras, LiDARs have demonstrated to be a better alternative for outdoor environments, as they are less sensitive to light changes. During the seasons, the alternation of sunny and cloudy days can not be overlooked in the choice of a sensor for the task at hand. The crucial part of the recognition process is shifted toward the identification of plants from point clouds, which, with respect to cameras, doesn't necessarily require the collection of a dataset of images from the field, a cumbersome operation to be repeated for adapting the same system to different types of crops.

2 | Background and Tools

It is common for robotic system to access two pieces of information to properly complete a task: the first is the external elements in their proximity, the second is the platform location within the environment. If we think, for example, of cleaning a room, some type of robotic vacuums need to know whether there is an obstacle in front of them to avoid it. Also, they might need an estimate of their position in the house to plan for their actions, like choosing in which room to move next. The understanding of the scene and the displacement of the robot within it constitute two different types of problems which are researched by different branches in informatics. The former one goes under the name of Computer Vision when the representation of the surrounding world is sensed through sensors such as cameras or LiDARs. The latter, instead, can be addressed in parallel to the creation of a map of the environment, by Simultaneous Localization and Mapping (SLAM). The algorithms used in this work for both of the cited categories are presented in the following section. Subsequently, the software which enables the robot to manage the different processes, i.e. the Robotic Operating System (ROS), and a tool for the management of point cloud data are described.

2.1. Computer Vision algorithms

Computer Vision represents a fundamental part of autonomous systems whenever the decision making process requires an understanding of the subjects in the scene under analysis. In robots based on cameras, this job can be executed by Neural Networks which are trained to perform object detection or instance segmentation. When talking about point clouds, instead, the elements can not be directly discerned as they might appear as a unique object when the points belonging to them are very close. The activity of point cloud segmentation tries to shade a light on the elements currently seen by the robot. Model estimation algorithms, like RANSAC, perform the recognition of various types of geometrical shapes, such as cylinders, lines and planes, while clustering algorithm, like DBSCAN, can help to discern of objects whose points are expected to be near each other.

2.1.1. DBSCAN

The Density-Based Spatial Cluster Algorithm (DBSCAN) [3] is a density clustering algorithm, i.e. it separate the points into groups according to their nearness and concentration in a dataset. Its general input parameters are a threshold ϵ describing the maximum distance within two points to assign them to the same cluster c and the minimum number of points defining the density regions $minPts$. The pseudocode is presented in algorithm 2.1[14], where $RangeQuery$ is a function returning the neighbour points within distance ϵ, computed with a specified function $distFunc$, of a query point P.

DBSCAN forms new clusters starting from a *core point*, i.e. a point P surrounded by at least $minPts$ points within a distance ϵ. At the beginning, it selects one by testing random data points against this requirement. Those discarded are considered as noise. Once the first core point has been found, it is assigned to a cluster along with its neighbours which are added to a queue S. For each of them, the area within ϵ distance is inspected and the included points are marked as belonging to their same cluster. If these points meet the $minPts$ threshold, they are added to S as well for later retrieval of other potential neighbours. In the case they do not, the points from which the test started becomes *border points* and their neighborhood is still assigned to their same cluster, but not anymore tested to expand the cluster. These operations are repeated until the queue is empty and, once it is, a new core point is searched to form a new cluster for as long as every point is not associated to one or labelled as noise.

The algorithm presents some important advantages with respect to other clustering alternatives. First of all, it can detect outliers, i.e. anomalous points which are different from the others observed. Indeed, at the end of the execution, those points that do not belong to any cluster are considered as noisy and discarded. In addition, it does not require the number of clusters present in the points to be specified a priori and it is not bounded to find a predetermined shape assumed by the groups of points. However, the result of the algorithm is not deterministic since points lying on the border between two cluster can be assigned to any of the two depending on which one is formed first. Also, the quality of the result depends on the distance measure in $distFunc$ and ϵ which are difficult to be selected especially when dealing with high-dimensional data. Lastly, since $minPts$ and ϵ characterize the density regions, DBSCAN can not be used in datasets where there is a large difference in point concentration.

Algorithm 2.1 DBSCAN Algorithm

Input: DB: Database
Input: ϵ: Radius
Input: $minPts$: Density threshold
Input: $dist$: Distance function
Input: $label$: Point labels, initially $undefined$

```
for each point p in database DB do
    if label(p) ≠ undefined then continue
    Neighbors N := RangeQuery(DB, distFunc, P, eps)
    if |N| < minPts then
        label(p) := Noise
        continue
    c ← next cluster label
    label(p) ← c
    SeedSetS ← N \ {p}
    for each q in S do
        if label(q) = Noise then
            label(q) ← C
        if label(q) ≠ undefined then continue
        NeighborsN ← RangeQuery(DB, distFunc, Q, eps)
        label(q) ← c
        if |N| < minPts then continue
        S ← S ∪ N
    endfor
endfor
endprocedure
```

2.1.2. RANSAC

The RANdom SAmple Consesus (RANSAC) is an iterative algorithm used for the estimation of the parameters of a specified mathematical model expected to be found in a set of points. It is based on the idea that data can be separated into inliers, i.e. points which fit the model according to some parameters, and outliers, i.e. those who do not. The inputs to RANSAC are the dataset D, the model to be searched, the maximum number of iterations to be performed k, the minimum number of data points that has to fit the model for the estimation to be successful $minN$, a threshold t to decide whether the inspected point can be considered as generated by the model and finally the least amount of inliers $minC$, also called consensus set, for the model to be considered a good one.

The algorithm can be described by the following pseudocode:

Algorithm 2.2 RANSAC Algorithm

```
1: procedure RANSAC(D, k, minN, t, minC)
2:     iterations := 0
3:     bestfit := null
4:     bestError := ∞
5:     while iterations < k do
6:         startInliers := selectRandomInliers(D)
7:         model := fitModel(startInliers)
8:         modelInliers := ∅
9:         for each point p in D do
10:            if error(model, p) < t then
11:                finalInliers := finalInliers ∪ p
12:        endfor
13:        if size(finalInliers) > minC then
14:            model := fitModel(finalInliers)
15:            modelError := totalError(model, p)
16:            if modelError < bestError then
17:                bestFit := model
18:                bestError := modelError
19:    endWhile
20: endprocedure
```

The working principle behind RANSAC can be divided into different steps. First of all, a number of random data points is selected from the dataset by $selectRandomInliers$ and a model is fitted to those points through $fitModel$. Then, the whole dataset is compared against it and the points which are close enough, i.e. which produce an error smaller than ϵ, are added to the consensus set. If the resulting number of inliers is greater than the given threshold $minC$, the model total error is computed and, in case it is less than the best error found so far, its parameters are saved along with the new value for the error. The refinement of the model continues until the maximum number of iterations is exhausted.

This algorithm is good for the estimation of models even in presence of a large number of outliers. The parameters quality, however, is affected by the maximum number of iterations allowed that, if too small, may bring to suboptimal solutions. Moreover, it is not suited for datasets in which more than an instance of the model can be found, where other algorithms such as the Hough Transform are preferable. However, this eventuality

is not present in the problem at hand, thus RANSAC is sufficient.

2.2. State estimation

As previously stated, a robot might need to know its location within the environment in order to act. Its current state can be represented, for example, by its (x, y, θ) coordinates with respect to a map. This information, however, is not always directly observable by a sensor such as GPS, therefore it has to be estimated over time and by employing a probabilistic approach.

In particular, the knowledge about the robot location can be represented as a probability distribution $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$, where the current location, x_t, depends on all the past states $x_{0:t-1}$, all the past measurements $z_{1:t-1}$, and all the control actions, i.e. all the movements commands given in input to the robot $u_{1:t}$.
To simplify the problem, we make the Markov assumption that the state transition model, i.e. the probability distribution over the next states, depends only to the current state and the last action:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.1}$$

The same assumptions can be applied to the sensor model, stating that whatever is sensed at time t only depends on the state at time t:

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.2}$$

2.2.1. Bayes Filter

We ha said that the knowledge of the robot is represented by a probability distribution. In the context of bayesian filtering, it is also known as belief $bel(x_t)$ and it is computed according to Bayes' theorem:

$$posterior = \frac{likelihood \times prior}{normalization} \tag{2.3}$$

so that:

$$\overline{bel(x_t)} = \int P(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{2.4}$$

$$bel(x_t) = \eta \underbrace{P(z_t|x_t, m)}_{likelihood} \underbrace{\overline{bel(x_t)}}_{prior} \tag{2.5}$$

where η is the normalization term and the posterior belief, $bel(x_t)$, can be obtained by applying two consecutive steps.

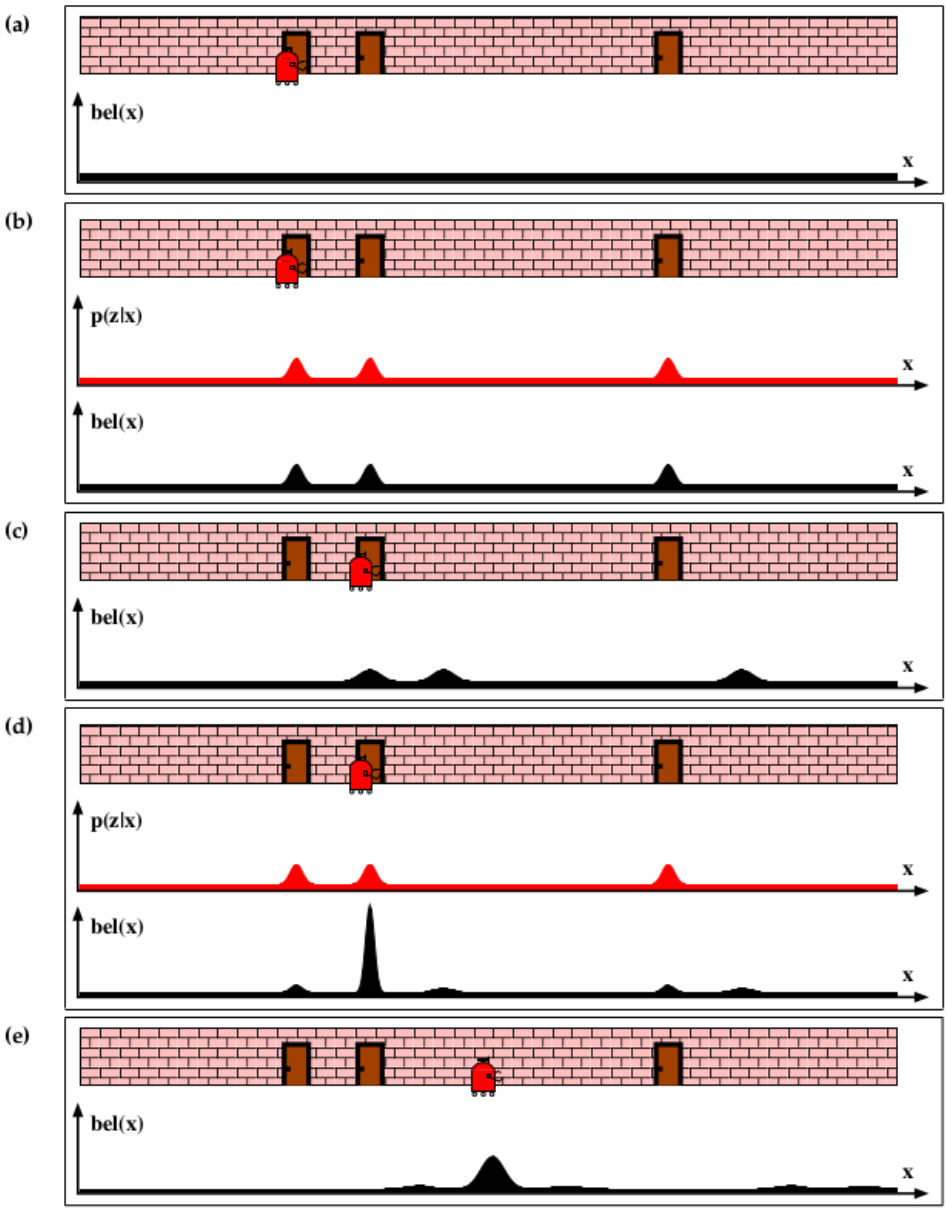


Figure 2.1: A robot moving down an hallway with the representation about the belief of its state. The transitioning from an image to another shows the effect of the prediction and the update step of the algorithm (Figure 1.1 in [?])

The first step, also known as prediction (Equation 2.4), resolves the prior by applying the state transition model to each of the previous possible states in order to predict the new probability distribution over the states after the robot has performed an action. This step involves for each state a time t an integration over each state at time $t-1$, therefore its complexity is $\mathcal{O}(n^2)$ being n the number of states. Since it is an expensive computation, the state representation plays an important role on the applicability of this algorithm to

reality. The prediction step, however, exploits a model for the robot movement which does not completely reflects the reality, and thus its execution reduces the uncertainty over the knowledge of the location. An example of this effect is shown in Figure 2.1, where the transition of the robot from the state in image b to c is coupled with the flattening of the belief. This indicates that the robot is less certain about its location. Without any clue about its surroundings, the estimate of the state would become shallower and shallower, until it "fades away".
Here, the second step, or correction step (Equation 2.5), has the job to update the estimate of the state through the sensors returns z_t. The posterior belief $bel(x_t)$ is obtained for each state by combining the likelihood of this measurement by the predicted belief $\overline{bel}(x_t)$. The introduction of new information about elements in the environment renders the robot more confident in its estimate, as it can be observed in the step from image c to d. In this case, the sensing of the second door after the first one makes the locations in its proximity more probable with respect to the others.

At last, a normalization term η is added to the equation to obtain a probability distribution for which, by definition, the integral over the probability density function has to sum up to 1. The computation of η involves an integral over the belief of each states as well, making it another critical operation in terms of time.

2.2.2. Kalman Filter

The Kalman Filter (KF) is an implementation of the Bayesian filter to the state estimation where the state space is continuous. At each time step t, the belief over the robot location and orientation is given by a multivariate Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$. While μ_t represents the point estimate of the variables composing the state, Σ_t is the covariance matrix depicting the uncertainty over the estimate. This representation overcomes the computational difficulties required by the integrations described in the Bayesian filter, as the multiplication and sum of Gaussian probability distributions is straightforward and does not require any cycle.

The KF can be applied to problems where the state transition model and the sensor model are both linear in the variables and both the noise in the motion and sensing is considered to be Gaussian:

$$\mathbf{x}_t = A_t\mathbf{x}_{t-1} + B_t\mathbf{u}_t + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, R_t) \tag{2.6}$$

$$\mathbf{z}_t = C_t\mathbf{x}_t + \delta_t \qquad \delta_t \sim \mathcal{N}(0, Q_t) \tag{2.7}$$

where (2.6) is the state transition model and (2.7) is the sensor model defined in (2.1) and (2.2). A_t and B_t are the matrices realizing the linear state transition, while C_t accounts for the linear sensor measurement. Since we do not know the real transition model, the prediction of our filter is not perfect, therefore we take into consideration such uncertainty by adding the process covariance matrix, R_t. The measurements that we get from the sensor is noisy as well, so we introduce Q_t (or sensor noise covariance matrix), to take care of it.
For the filter to work, the initial state estimate $bel(x_0)$ has to follow a Gaussian distribution. From that point on, the filter only performs summation and multiplication of Gaussians, which still results in a Gaussian distribution, hence the posterior is Gaussian as well (at least up to a normalizing constant) and the prior property need for the filter functioning is preserved.

Algorithm 2.3 Kalman Filter Algorithm

1: $\bar{\mu}_t = A_t\mu_{t-1} + B_t\mu_t$
2: $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
3: $K_t = \bar{\Sigma}_t C_t^T(C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
5: $\Sigma_t = (I - K_tC_t)\bar{\Sigma}_t$
6: return μ_t, Σ_t

In 2.3, the algorithm behind the filter is illustrated. Instructions 1 and 2 execute the prediction step, where the filter predicts the next state and, in doing so, increases the covariance over the state by summing the matrix R_t. The Instruction 3 computes the Kalman Gain, i.e. the weighting factor which decides how much correction has to be applied to the state given the received measurement z_t. Instruction 4 and 5 update the predicted state and the covariance matrix which reflects the reduction in uncertainty due to the incorporation of another measurement into the filter.

This filtering process requires the assumption of the linearity in the state transition and the sensor model. In reality, however, not all the models are linear. The Extended Kalman Filter is presented as a solution to this problem.

2.2.3. Extended Kalman Filter

The Extended Kalman Filter (EKF) defines the models as:

$$\begin{aligned} \mathbf{x}_t &= g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \epsilon_t & \epsilon_t &\sim \mathcal{N}(0, R_t) \\ \mathbf{z}_t &= h(\mathbf{x}_t) + \delta_t & \delta_t &\sim \mathcal{N}(0, Q_t) \end{aligned}$$

where g and h are non-linear functions representing the state transition and sensor models. The filterings still works with linear functions, therefore a linearization step needs to take place. This operation is carried on by resorting to the first order Tailor expansion of g and h:

$$\begin{aligned} g(\mu_t, u_t) &= g(\mu_t, u_t) + G_t(\mu_t) \\ h(\mu_t) &= h(\mu_t) + H_t(\mu_t) \end{aligned}$$

where G_t and H_t, are the so called Jacobians, and are defined as following:

$$G_t = \frac{\partial g(\mu_{t-1}, u_t)}{\partial \mu_{t-1}} \tag{2.8}$$

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial \bar{\mu}_t} \tag{2.9}$$

The EKF algorithm differs from KF as it substitute matrices A_t, B_t and C_t with their linearized version:

Algorithm 2.4 EKF Algorithm

1: $\bar{\mu}_t = g(u_t, \mu_{t-1})$
2: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
6: return μ_t, Σ_t

2.3. Simultaneous Localization And Mapping

2.3.1. Problem formulation

In some circumstances we might not have a map of the environment where the robot is placed, aside from not knowing its position and orientation (x, y, θ) with respect to it. This can be the case of outdoor places, where a GPS signal is not always available because of occlusions caused, for example, by tree canopies. The robot, then, can only access a rough estimate of its positioning through a sensor and a set of observations $z_{1:t}$.
Depending on the application, we may need to either compute the full trajectory of the robot or keep only the latest map and robot pose. The first problem is known as Full SLAM and its goal is to find $p(x_{1:t}, m|z_{1:t}, u_{1:t})$. It can be solved with various algorithm, such as Fast SLAM. The second one is instead called Online SLAM and aims at estimating $p(x_t, m|z_{1:t}, u_{1:t})$. It does so by continuously integrating over time the previous robot poses:

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int \int \dots \int p(x_{1:t}, m|z_{1:t}) dx_1 dx_2 \dots dx_{t-1} \tag{2.10}$$

One of the approaches to find the solution to the Online SLAM problem is EKF-SLAM, which is based on the EKF algorithm, but it has some key differences which is explained in the following Section.

2.3.2. EKF SLAM

In EKF-SLAM, the EKF algorithm is used to simultaneously estimate the last pose of the robot and a map of landmarks. Landmarks are point of interest detected in the environment and are usually stored in the form of a set of (x_l, y_l) coordinates that are used by the robot to correct its pose and map estimates. Differently from EKF, the state now comprises both the robot variables, x, y and θ and the observed landmarks. It is represented by a multivariate Gaussian $\mathcal{N}(\mu_t, \Sigma_t)$:

$$\mu_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ l_1 \\ l_2 \\ \vdots \\ l_N \end{bmatrix} \quad \Sigma_t = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl_1} & \sigma_{xl_2} & \dots & \sigma_{xl_N} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\theta} & \sigma_{yl_1} & \sigma_{yl_2} & \dots & \sigma_{yl_N} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_\theta^2 & \sigma_{\theta l_1} & \sigma_{\theta l_2} & \dots & \sigma_{\theta l_N} \\ \sigma_{xl_1} & \sigma_{yl_1} & \sigma_{\theta l_1} & \sigma_{l_1}^2 & \sigma_{l_1 l_2} & \dots & \sigma_{l_1 l_N} \\ \sigma_{xl_2} & \sigma_{yl_2} & \sigma_{\theta l_2} & \sigma_{l_1 l_2} & \sigma_{l_2}^2 & \dots & \sigma_{l_2 l_N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{xl_N} & \sigma_{yl_N} & \sigma_{\theta l_N} & \sigma_{l_1 l_N}^2 & \sigma_{l_2 l_2} & \dots & \sigma_{l_N}^2 \end{bmatrix}$$

where μ_t is the mean estimate for the state, while Σ_t is the covariance matrix which takes into account all the uncertainties about the locations and the correlations among them. The EKF SLAM works under the assumption that the motion and the sensing of the robot are affected by a small Gaussian noise, as a large uncertainty in the posterior would introduce increasing errors in the linearization process [**?**]. Also, only a limited amount of landmarks can be managed as the update of the algorithm is quadratic in the number of landmarks, and therefore computationally inefficient.

During the prediction step, EKF SLAM only involves modifications in the robot state mean and covariances, hence the matrix F_x is introduced as a multiplier factor for the matrix G_t to leave the landmark map unmodified:

$$F_x = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

where the number of trailing zeros to the right is equal to the number of landmarks times the dimensionality of the landmark vector. The matrix R_t is still a 3×3 and only includes the covariances for the robot state.

On the other hand, the correction step requires an additional step of data matching to understand whether the detected landmarks have already been stored or not. If a landmark is not new, their location estimate is retrieved and used for the correction. In the case it is, it is added to the mean vector and its position can be initialized with an estimate considering the incoming measurements of the sensor and the current pose of the robot.
Since the update step following Algorithm 2.4 would require the μ vector and the Σ_t matrix to be modified for each landmark in the detection range, a batch update can be performed by computing the $H_{i,t}$ matrix for each landmark separately and then stacking them to form a H_t matrix and continue with the instructions of Algorithm 2.4. The $H_{i,t}$ matrices should pertain only the landmarks i, therefore, another mapping is needed as follows:

$$F_{x,i} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & \underbrace{0}_{l_i} & 1 & 0 & \dots & 0 \end{bmatrix}$$

2.4. Software Tools

Robots are made by different types of sensors and actuators, which produce many types of information. Programming drivers at a low end would become difficult in presence of multiple products from various companies, since each component would require its own instruction set. The Robot Operating System (ROS), which is also used in this work, was created to overcome this issue. Apart from its application over real robotic system, ROS can manage also simulated ones in conjuction with Gazebo, a software which has been employed for the simulation of a real world environment. In the specific case of data received by a LiDAR sensor, the library known as PointCloud Library (PCL) has been exploited for the management of point clouds and the execution of segmentation activities.

2.4.1. Robot Operating System

ROS is an open-source middleware for software development in robotics. It has been built with modularity in mind and it offers a set of tools for allowing different applications to communicate, thus promoting the sharing and reusability of code.
Inside ROS, a program can be distributed as a collection of one or more packages written in the same or different programming languages, in particular C++, Python and Lisp. Each package contains one or multiple nodes, i.e. executables, performing some type of computation. In a navigation package, for instance, we can find a node responsible for mapping and another for localization. In the following, some of the ROS core features are briefly introduced.

Nodes interactions types Nodes can exchange messages between them through Topics, Services and Actions, depending on the type of communication needed:

- **Topics**: unidirectional channel bounded to a name for the identifying it and a message format, specifying the rules to send information over the channel. There are many message types which can vary from float and string only data, to more structured ones, such as robot poses, camera images or point clouds. In the context of topics, the nodes sending messages are called publishers, while all the others listening to the topic are named subscribers. A node can subscribe and publish to multiple topics but as a publisher it can never receive an answer back, since the communication is one-way;
- **Services**: two-way synchronous form of interaction. To initiate the exchange of data, a node sends a request message to a server and waits until it receives a response;

- **Actions**: unlike Services, Actions allow a node to continue its flow of execution after a request is made and to optionally check for the progress state of the demanded job. It is also possible for the client to issue cancellation commands to the server. Once the action is completed, the result is returned to the requesting node.

ROS Master For any type of communication to happen, an instance of a ROS master needs to be running. This component is essential as it provides a registration and naming service, so that the nodes can locate each other over the ROS network before starting to exchange messages. The master keeps also a record of all the publishers and subscribers for any topic and services offered by the various nodes. Moreover, it manages the parameter server, a global dictionary where each node can get access to and retrieve or modify information. It is usually employed to define configuration parameters so that they can be shared across multiple nodes thus reducing the eventuality of bugs due to rewriting the same parameters in different places. A parameter is usually composed of a name, a type and a value and can be added at runtime through the terminal, within a node or inside the launch file, a special kind of file which allows to easily launch multiple nodes with their configuration.

ROS Bags Aside from the real-time acquisition and manipulation of data coming from sensors, ROS offers the possibility to save all the topics and messages exchanged during a session to a file known as ROS bag. The recordings can then be played as many times as needed to test the functionalities of nodes.

2.4.2. Gazebo

Gazebo is a 3D simulation software used for robotics applications. It allows the construction of real world scenarios starting from tree-dimensional representations of objects and environments and the reproduction of their physical properties, such as surfaces friction. In particular, it can simulate robotic systems, which can move within these worlds and collect data about their surrounding elements through a variety of sensors, including LiDARs and cameras. Different parameters, like the resolution of sensors and the noise disturbing their detections, can be manually adjusted to better model the reality.

2.4.3. PointCloud Library

PCL [13] is a C++, open-source library for 3D point cloud processing. It implements several state-of-the-art algorithms for the manipulation of point cloud data, such as filtering for the isolation of specific points, object recognition, model fitting, e.g. RANSAC,

surface reconstruction, segmentation and so on.
All the algorithms provided are based on the same interface which is composed by four main steps: creation of the processing object, like filters or segmentation; call to the *setInputCloud* method to pass the cloud to the processing object; parameters setting through the use of the object methods and finally the call to the function to run the operations.
Since the point clouds have to be passed to different objects, its better to avoid to copy them more than needed as they occupy a lot of memory and therefore can slow down the computation. PCL solves this issue by using shared pointers, a data type included in the Boost libraries set. This is not the only library required for PCL functioning, as it relies also on Eigen for performing linear algebra operations, on the Fast Library for Approximate Nearest Neighbors (FLANN) for fast k-nearest neighbour search operations and on VTK for point clouds and surface visualization data through the PCL Visualization library.
Another feature offered by the Point Cloud Library is the possibility to save and retrieve point cloud data thanks to its input-output module in the form of PCD files, allowing an easy integration of existing point clouds into the projects.

3 | Implementation

Plant counting is an expensive task when executed physically in the field, since it requires time for the human operator, agricultural machinery or self-contained robotics unit to complete an full exploration. To reduce its cost, we developed a system which can be potentially mounted on nursery's vehicles thus parallelizing the detection of plant with existing procedures, such as weeding. Another advantage of this setting is that it does not request the farmers to buy a specific robotic platform, so there are less expenses involved in the automation.
The core decision for this work was about the type of sensor to be employed, of which three were presented in the State of The Art (section 1.2). Mechanical and infrareds are easy to be operated but they offer a very limited source of information about the subject detected and they have to be accurately positioned to be used effectively. Passing to more technologically advanced sensors, cameras represent an inexpensive mean for plant counting, but the recognition phase requires a dataset to be collected from the field and it is sensitive to changes in light conditions. Therefore, taking into consideration the outdoor context in which an orchard nursery is located, our choice falls on a 3D LiDAR.

The acquisition and manipulation of the point cloud by the sensor was managed through ROS Noetic, run on Ubuntu 20.04. Various nodes have been created to support the analysis operations.

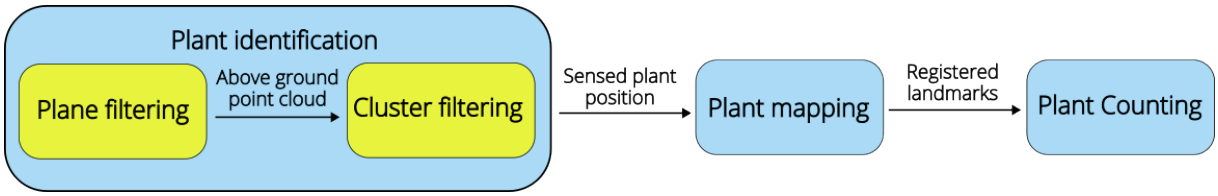


Figure 3.1: Overview of the main steps of plant counting

In the following, the proposed process at the base of this work is described. An overview containing the three main steps in which it can be divided shown in Figure 3.1:

- Plant detection: identification of plant elements from the point cloud. It compre-

hends the removal of the ground plane and the clustering involved in plant recognition;

- Plant mapping: map and trajectory estimation;
- Plant counting: count of the plants registered in the map.

This final phase is performed in post-processing, at the end of the execution. The interactions among nodes and their specific details can be found in section A.1.

3.1. Plant identification

Because of the sensor position, the ground is also included in the field of view and it has to be removed to avoid interference in plants recognition. Fixing a threshold above which only the plants are expected to be visible is not a viable solution. Indeed, as it might happen in the case of sensors mounted at a certain height, irregularities in the terrain can cause oscillations during the acquisition and the ground point might be incorrectly detected as plants.
After the elimination of the ground from the scene, the single plant elements have to be identified. The procedure taken for addressing these two problems are illustrated in Figure 3.2.

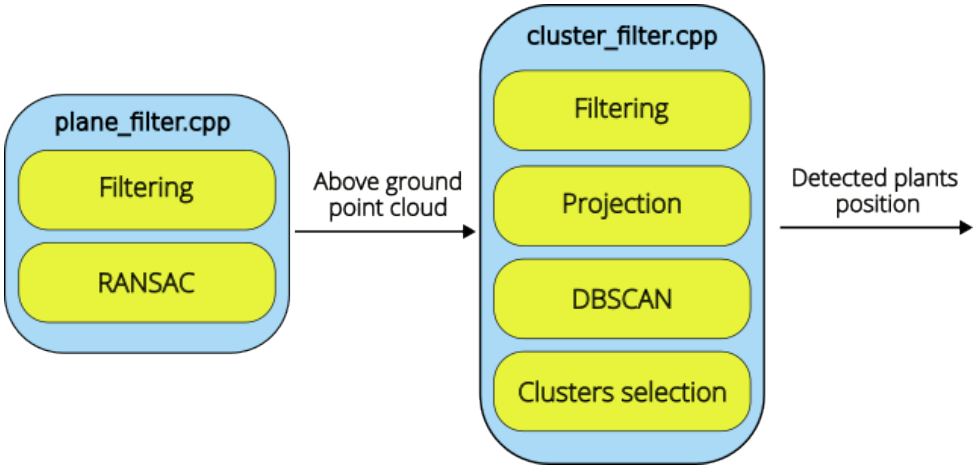


Figure 3.2: Main procedures included in plant filtering and cluster filtering

3.1.1. Plane filtering

Before any type of segmentation takes place, we have narrowed the area over which to perform the operations by using a filter over the three dimensions through the PCL `CropBox` object. This allowed us to discard farer points that might not be representative

of the ground currently under inspection and to decrease the computational time required by the algorithm.
To simplify the identification of the terrain, we decided to model it as a plane, as made in [21]. In this way, we reduced the problem to the estimation of the four coefficients of the plane in the point cloud, according to the equation:

$$ax + by + c = d \tag{3.1}$$

Given at least three points from the point cloud, namely $p1 = (x1, y1, z1)$, $p2 = (x2, y2, z2)$ and $p3 = (x3, y3, z3)$, we can compute the normal $\mathbf{n} = (a, b, c)$ of the plane fitting them. To do so, we firstly calculate the elements of vectors $\mathbf{r}$ and $\mathbf{s}$:

$$\mathbf{r} = p2 - p1 \tag{3.2}$$

$$\mathbf{s} = p3 - p1 \tag{3.3}$$

As a result, they lie on the plane surface. Then, since the cross product of two vectors returns a vector perpendicular to both of them, $\mathbf{n}$ can be found by performing:

$$\mathbf{n} = \mathbf{r} \times \mathbf{s} \tag{3.4}$$

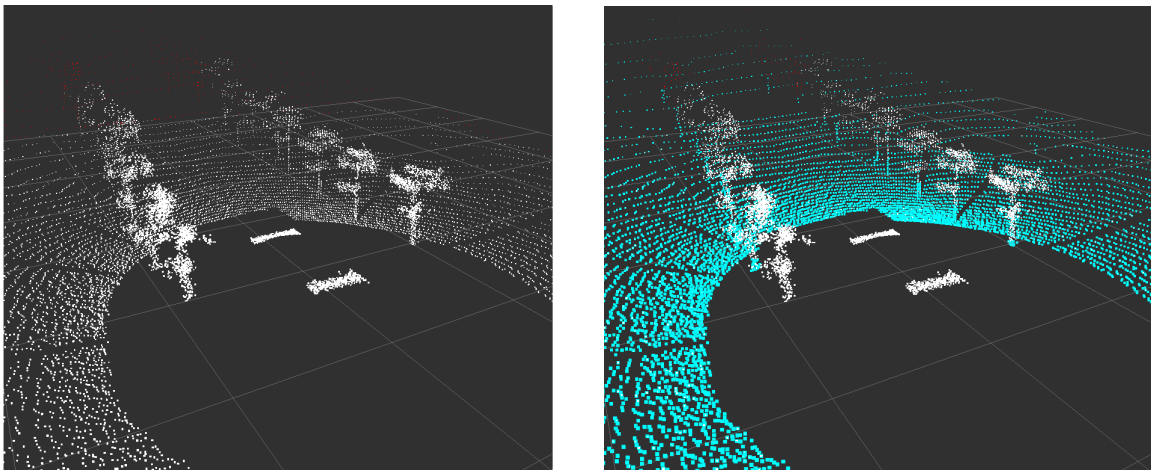
The last part consists in the normalization of $\mathbf{n}$ and the calculation of d.

To automatize and efficiently execute this research for the plane, we employed RANSAC (see Section 2.1.2) from the library PCL. The algorithm repeatedly extracts three random points from the point cloud, it computes the plane coefficients, counts the number of inliers, evaluates the model error and saves the plane equation if it results to be less than the best one found so far. In the case of a plane model, the inliers are found by using the following distance measure referred to a generic point i and by comparing the result to a threshold t:

$$d_i = \frac{ax_i + by_i + cy_i + d}{\sqrt{a^2 + b^2 + c^2}} \tag{3.5}$$

An example of the plane identified with this methodology is shown in Figure 3.3.

The described process accounts for a fixed point cloud, whereas the one returned by the sensor continuously changes. It is reasonable to assume that from the time step t to $t + 1$ the ground would be approximately in the same position. Therefore, to include



(a) (b)

Figure 3.3: Demonstration of plane segmentation with RANSAC. On the left, the point cloud in input to RANSAC and on the right the inliers to the plane are highlighted

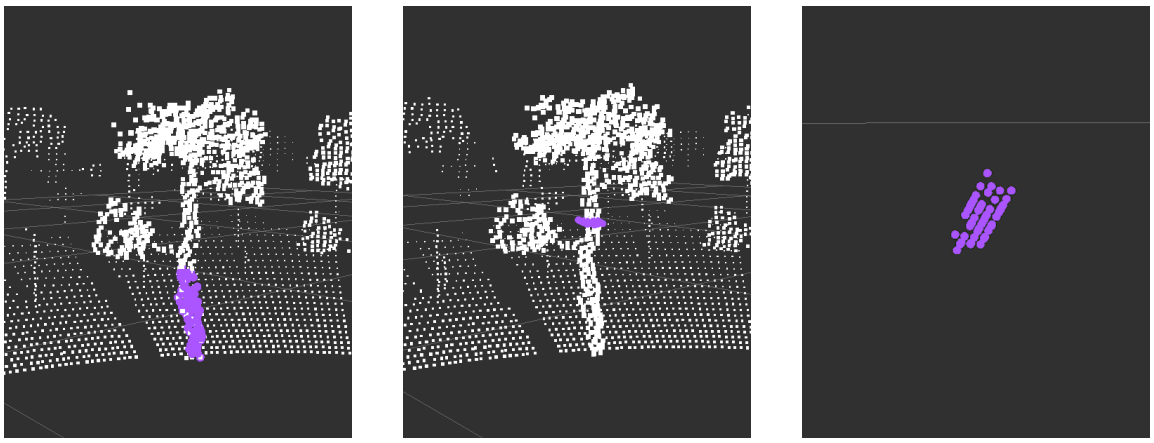
consistency in the estimation, we save the coefficients found at time t and we compute the inliers at time $t+1$ by using those coefficients. The new inliers become the base point cloud from on which RANSAC is re-executed to find the plane equation for $t+1$.
During the various iterations, the plane models are tested to see whether their intersection point with the z axis is at a certain height. This additional check is optional and has been inserted to try to drive the search toward the ground and avoid the detection of wrong planes which, in the point cloud, span across the tree crowns. This eventuality has been considered, since the point cloud density is greater where the foliage is located and thus RANSAC might incorrectly identify planes located at that level instead of the ground. In such case, the found model is discarded along with its inliers and the search is repeated until a number of maximum iterations is met.

3.1.2. Cluster filtering

After the elimination of the ground points, this second step addresses the recognition of plants. Another filter, of the same type of the one used in the precedent phase, is employed to leave only the point cloud nearby the robot for the analysis, as the farer parts are later explored as the movement proceeds. Then, the robot points are removed by applying a PCL `PassThrough` filter to focus the search only on the sides of the platform.

We chose the DBSCAN (subsection 2.1.1) algorithm in order to distinguish plant elements from the cloud. Although it is computationally expensive, it was considered as a better alternative to other methods, such as Euclidean Clustering, as it groups points according

to density. This last characteristic can be particularly helpful, as it reduces the possibility of joining two different plants together, a case which might happen when they are very close because, for example, of the presence of leaves. However, we did not give the algorithm the full 3D point cloud. We exploited the knowledge that the shape of plants develops vertically, hence a greater concentration of points is present where the trunk is located. Therefore, we projected the point cloud over the xy plane as it is shown in Figure 3.4.



(a) (b) (c)

Figure 3.4: Result of the projection of the point cloud on the xy plane. On the left the point cloud of a plant obtained through the filtering step, on the center the projection of the point cloud visible from the side and on the right the same projection from above.

A further advantage of this operation is that we do not have to manage the different densities which characterizing plants clusters along the height which might cause the same plant to be split in multiple groups.
Even if the DBSCAN parameters can be manually adjusted to achieve the desired level of performance, there still exist the possibility of various parts of the plants to be detected separately. Also, even after the plane filtering operation, some points belonging to the ground might mix in during the plant identification phase. We tried to reduce as much as possible these eventualities by characterizing the plants clusters in terms of width, height and number of points. Since the above mentioned type of interference usually comes in the form of short and small groups of points in the first case, and short and wide in the second, we imposed three threshold over the clusters returned by DBSCAN, i.e. maximum width, minimum height and maximum number of points. Still, we can not tell apart objects similar to plants which might be present on the field, such as poles or other support structures, in which case they have been marked as plants.
For the remaining clusters, the barycenter is computed over x, y and z by averaging the

location of the N points assigned to them as:

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i \quad y_c = \frac{1}{N}\sum_{i=1}^{N} y_i \quad z_c = \frac{1}{N}\sum_{i=1}^{N} z_i \tag{3.6}$$

The result of the identification phase is illustrated in Figure 3.5, where the squares over the plants indicate the position of the barycenter.

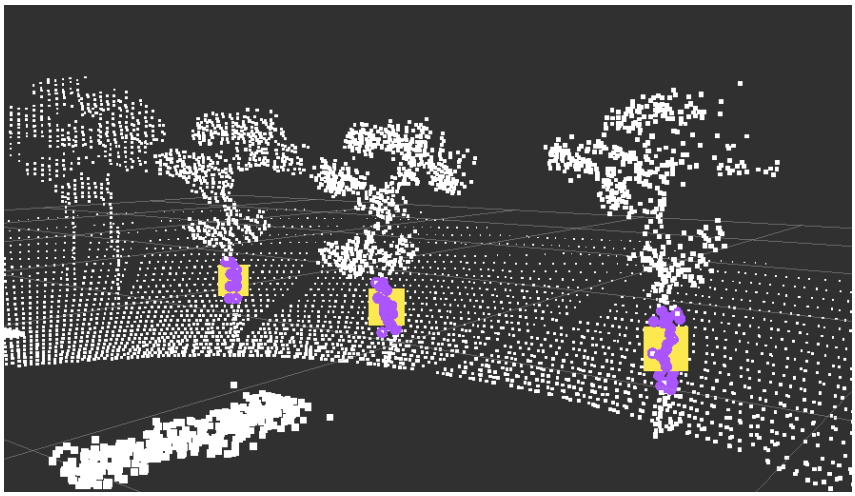


Figure 3.5: The resulting clusters position after the cluster filtering process has been executed

3.2. Plant mapping

We plan our robot to be working in orchards, where a GPS (or RTK-GPS) signal might not always be available because of trees crowns. The information source accessible to us would then be the odometry, i.e. a rough estimate of the robot location produced, for example, by the integration of signals coming from encoders. Wheel encoders are a type of sensor attached to the wheels of the robot which return, with a given frequency, how much the wheel has rotated with respect to the previously registered measurement. However, when applied to the practice, especially on slippery terrains, this system does not give a good estimate of the robot position, as some movements may not be tracked by the encoder. The integration of bad measurements over time would eventually make the odometry drift and the robot to think of being somewhere in the environment when, actually, it might be meters away from its estimate.

Aside from this problem, our primary objective is to count plants in a field, which raises the question on how to avoid double-counting them. Indeed, the robot continuously moves

to explore the environment, therefore the same plants are detected at different positions by the sensor.

To overcome these challenges, we decided to perform trajectory correction and mapping with an EKF-SLAM approach, as presented in subsection 2.2.3. The input to the algorithm are two: the odometry and the plants position (x_s, y_s) in the sensor coordinate frame detected through the previous step. The final output is a map of landmarks, where each landmark corresponds to the (x_m, y_m) coordinates of the plants in the environment. In the next sections the motion model and the sensing of the robot are presented, along with the characteristics of our implementation and the problem of data association.

3.2.1. Prediction step

As previously stated, the algorithm is composed by two steps: prediction and update. To accomplish the first one, we need to define how the state variables evolve from x_t to x_{t+1}. In this case, they are composed by the robot position and orientation (x_R, y_R, θ_R) and we assume the robot to be transitioning between one state and the other according to the odometry motion model presented in [**?**].

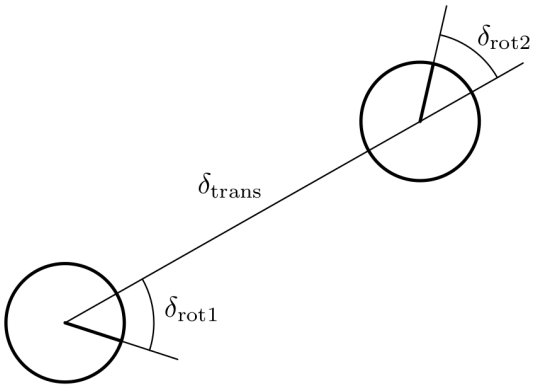


Figure 3.6: The tree type of movement composing the odometry motion model: a rotation δ_{rot1}, a transition δ_{trans} and a final rotation δ_{rot2} (Figure 5.7 in [**?**])

This model approximates any movement with three steps: a rotation δ_{rot1} from the original robot orientation θ_R^t towards the direction of the translation, a translation δ_{trans} from the current position to the final one and a last rotation δ_{rot2} to match the final angle θ_R^{t+1}. A graphical representation of this partitioning can be seen in Figure 3.6.
From the EKF-SLAM perspective, the inputs are in the form of odometry poses, which we can write as $(x_{odom}^t, y_{odom}^t, \theta_{odom}^t)$, and assume to be control commands u_t received by

the robot. Therefore, we can directly apply the motion model starting from them:

$$\begin{aligned} \delta_x &= x_{odom}^{t+1} - x_{odom}^{t} \\ \delta_y &= y_{odom}^{t+1} - y_{odom}^{t} \\ \delta_{rot1} &= atan2(\delta y, \delta x) - \theta_{odom}^{t} \\ \delta_{trans} &= \sqrt{{\delta_x}^2 + {\delta_y}^2} \\ \delta_{rot2} &= \theta_{odom}^{t+1} - \theta_{odom}^{t} \end{aligned} \tag{3.7}$$

In the original version of the model, a Gaussian noise is added to δ_{trans}, δ_{rot1} and δ_{rot2} to account for the differences between the perfect execution of these three movements and the reality, where the robot, for instance, might slightly translate during a rotation:

$$\begin{aligned} \hat{\delta}_{rot1} &= \delta_{rot1} - \mathcal{N}(0, \alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2) \\ \hat{\delta}_{trans} &= \delta_{trans} - \mathcal{N}(0, \alpha_3\delta_{trans}^2 + \alpha_4\delta_{rot2}^2) \\ \hat{\delta}_{rot2} &= \delta_{rot2} - \mathcal{N}(0, \alpha_1\delta_{rot2}^2 + \alpha_2\delta_{trans}^2) \end{aligned} \tag{3.8}$$

The computation of the final pose $(x_{t+1}, y_{t+1}, \theta_{t+1})$ is then given by:

$$\begin{aligned} x_{t+1} &= x_t + \hat{\delta}_{trans} cos(\theta^t + \hat{\delta}_{rot1}) \\ y_{t+1} &= y_t + \hat{\delta}_{trans} sin(\theta^t + \hat{\delta}_{rot1}) \\ \theta_{t+1} &= \theta_t + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \end{aligned} \tag{3.9}$$

The filter formulation requires us to separate the update of the state from the update of the noise. Hence, for the first, we compute the next state by using the same equations as in 3.9 but substituting the perturbed movements with the original ones:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \delta_{trans} cos(\theta_t + \delta_{rot1}) \\ \delta_{trans} sin(\theta_t + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix} \tag{3.10}$$

Since the transition is not linear in the state variables, in order to update the covariance matrix Σ_t, we need to compute the Jacobian of the motion model G_t:

$$G_t = I + \begin{bmatrix} 0 & 0 & -\delta_{trans} sin(\theta_t + \delta_{rot1}) \\ 0 & 0 & \delta_{trans} cos(\theta_t + \delta_{rot1}) \\ 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

Differently from the formulation of the model given by the book, we define the process noise R_t as a diagonal matrix:

$$R_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{3.12}$$

where σ_x^2, σ_y^2 and σ_θ^2 are the variances describing the uncertainty in the computation of x, y and θ respectively. At the end, the prediction for the state covariance Σ_t results in:

$$\Sigma_{t+1} = G_t \Sigma_t G_t^T + R_t \tag{3.13}$$

3.2.2. Update step

As already discussed, whenever a prediction is made in the EKF algorithm, the uncertainty about the state estimate increases. Without the recognition of some sort of reference points in the environment, the robot would end up in a situation similar to the one depicted in Figure 3.7, where the covariance over the position becomes bigger and bigger, until the robot does not know anymore where it can be located.

The plant positions given by the cluster filtering phase, however, are not directly usable by the algorithm to perform a correction, since they are expressed in the sensor frame, which is different with respect to the map one. As we can see in Figure 3.8, a transformation composed by a rotation and a translation bounds would be required to go from a coordinate system to the other. Luckily, the EKF-SLAM algorithm already estimates the robot position (x_R, y_R) and its orientation, therefore, it can take care of this conversion by exploiting the sensor model and the inverse sensor model.

Whenever the position of a new cluster is received from the plant identification phase, its coordinates in the map are computed as:

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} cos(\theta_t) & -sin(\theta_t) \\ sin(\theta_t) & cos(\theta_t) \end{bmatrix} \begin{bmatrix} x_s \\ y_s \end{bmatrix} + \begin{bmatrix} x_R \\ y_R \end{bmatrix} \tag{3.14}$$

and checked against all the registered landmarks which are stored as (x, y) coordinates inside the filter state vector. If no correspondence is found, the new landmark is added to the map and its position is initialized with the one resulting from the application of the inverse sensor model. Otherwise, the matching entry needs to be converted back to the

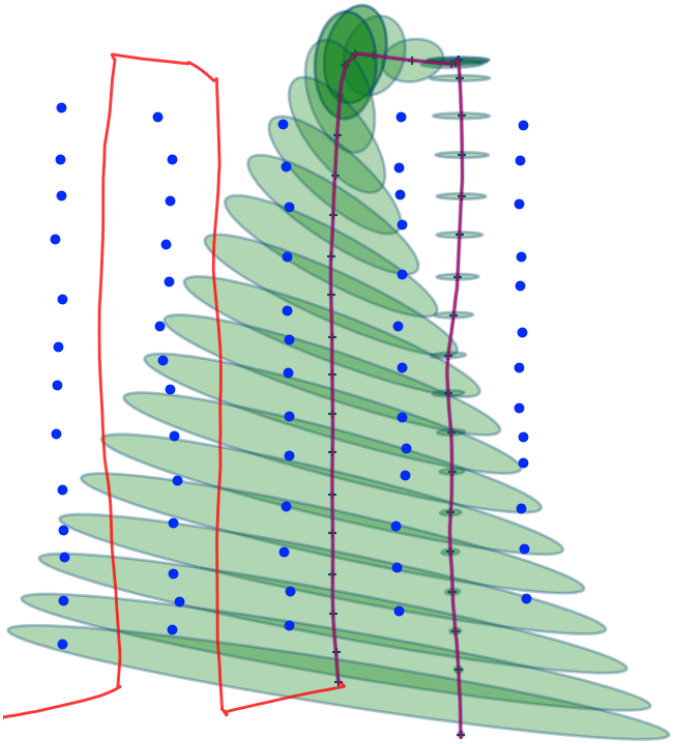


Figure 3.7: State covariance updates during the movement without any correction. The straight red line represents the path followed by the robot and the green ellipses the covariance over its position x_R, y_R

sensor frame in order to perform a comparison with the received measurement, therefore the sensor model is used:

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} cos(\theta_t) & sin(\theta_t) \\ -sin(\theta_t) & cos(\theta_t) \end{bmatrix} \left(\begin{bmatrix} x_m \\ y_m \end{bmatrix} - \begin{bmatrix} x_t \\ y_t \end{bmatrix} \right) \tag{3.15}$$

Both this projection in the sensor coordinate system and the obtained measurement are then involved in the computation of the Kalman gain matrix K_t which is be used to correct the state vector.

Back to the EKF-SLAM algorithm, our sensor model is not linear as it happens in the case of the motion model. Therefore, the Jacobian matrix $H_{i,t}$ becomes necessary to perform the update. Since the measurements almost always comprise more than one detection per time, we can compute the matrix H_t as the stack of different matrix $H_{i,t}$, each one relative to the landmark l_i. The matrix $F_{x,i}$ defined in subsection 2.3.2 can be applied to this case, so that only the derivatives of the sensor model for l_i with respect to the robot state and its position in the state vector are taken into account:

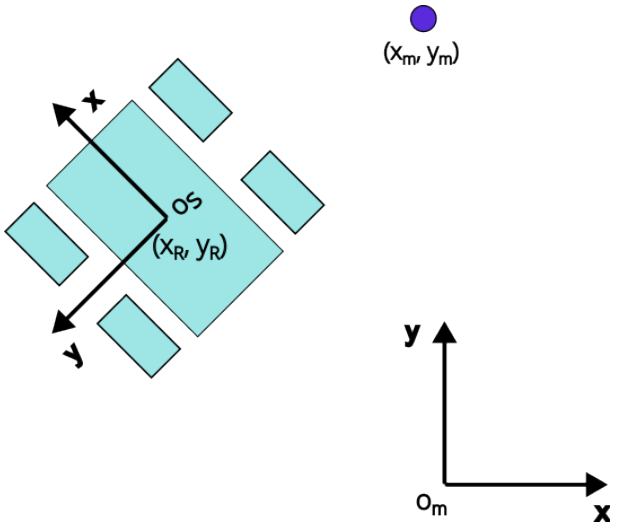


Figure 3.8: The sensor reference frame denoted by S, located within the map. The robot is rotated and translated with respect to the map frame. Its position is denoted by (x_R, y_R), while the plane one by (x_m, y_m)

$$H_{i,t} = \begin{bmatrix} \frac{\partial h(z)}{\partial x} & \frac{\partial h(z)}{\partial y} & \frac{\partial h(z)}{\partial \theta} & 0 & \dots & \frac{\partial h(z)}{\partial l_{i,x}} & \frac{\partial h(z)}{\partial l_{i,y}} & \dots & 0 \end{bmatrix} \tag{3.16}$$

where the partial derivatives with respect to the robot state μ_R are the following:

$$\frac{\partial h(z_i)}{\partial \mu_R} = \begin{bmatrix} -cos(\theta_t) & -sin(\theta_t) & -sin(\theta_t)(x_m - x_t) + cos(\theta_t)(y_m - y_t) \\ sin(\theta_t) & -cos(\theta_t) & -cos(\theta_t)(x_m - x_t) - sin(\theta_t)(y_m - y_t) \end{bmatrix} \tag{3.17}$$

whereas the ones with respect to the landmark itself are defined as:

$$\frac{\partial h(z_i)}{\partial l_i} = \begin{bmatrix} cos(\theta_t) & sin(\theta_t) \\ -sin(\theta_t) & cos(\theta_t) \end{bmatrix} \tag{3.18}$$

The update step then can be concluded with the calculation of K_t and the correction of μ_t, Σ_t as in Algorithm 2.4. The effects of the reduction in the uncertainty realized by this step can be appreciated from Figure 3.9.

3.2.3. Landmark association problem

As previously mentioned, whenever a measurement is given in input to the algorithm, a test has to be run over all the memorized landmarks to find which one is being sighted. This part is essential for the proper functioning of a SLAM algorithm in general, since a wrong linkage might cause the filter to diverge and produce a bad estimate of both the

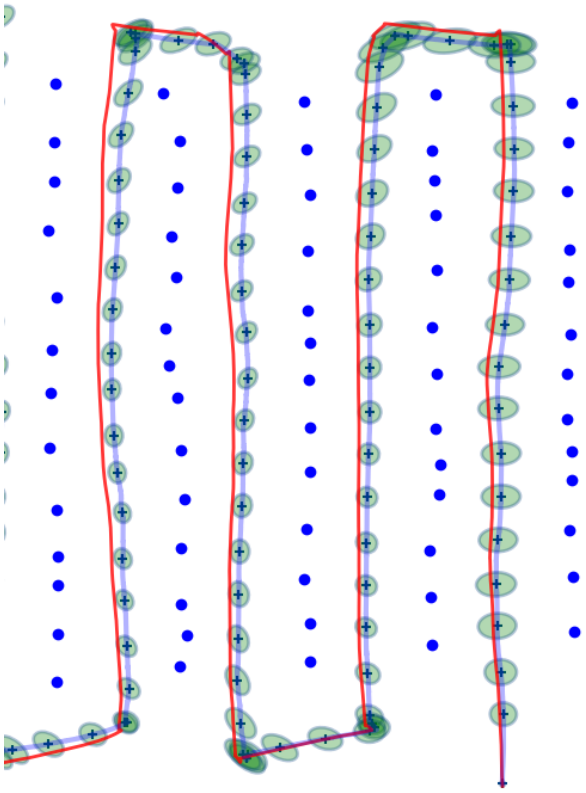


Figure 3.9: Covariance evolution during the robot motion with the introduction of the update step.

map and the trajectory.
The use of a simple check about the Euclidean distance between the detected value and the estimated one is not sufficient, since it would not take into account the uncertainty in the estimation. Indeed, the landmark i is stored as a multivariate Gaussian distribution, whose position μ_i is placed in the state mean vector and whose covariance Σ_i is located starting from row and column i in the filter covariance matrix Σ:

$$\mathcal{N}(\mu_i, \Sigma_i) = \left(\begin{bmatrix} x_i \\ y_i \end{bmatrix}, \begin{bmatrix} \sigma_{i,x}^2 & \sigma_{i,xy} \\ \sigma_{i,xy} & \sigma_{i,y}^2 \end{bmatrix} \right) \tag{3.19}$$

Hence, to fully consider the landmark estimation in the comparison, we chose to perform an outlier test based on the Mahalanobis distance. This type of measure scales the distance of a point x to the center of a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ by the covariance along the multivariate components, according to the formula:

$$D(x, \mu) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)} \tag{3.20}$$

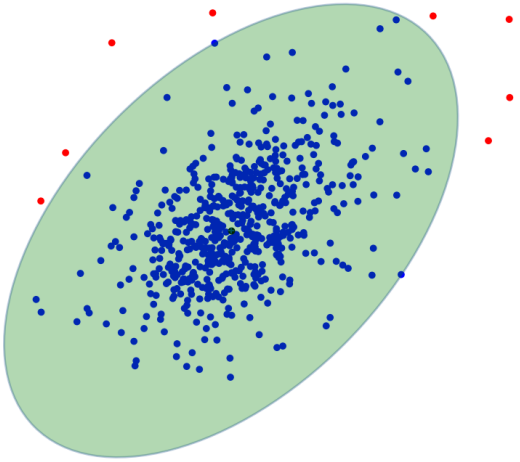


Figure 3.10: Chi-squared test performed over a set of points with $N = 2$ and at $\alpha = 0.01$. The green ellipse is the threshold out of which the points are considered outliers.

Since the Mahalanobis distance of samples drawn from a multivariate Gaussian distribution of N components follows χ^2 distribution of N degrees of freedom, we can perform a χ^2 test to assess whether or not the point x comes from this distribution.

Generally speaking, the χ^2 test is a statistical test that can be used to understand the behaviour of the observed data with respect to the expectations. It is usually applied to establish whether two different categorical variables, for example the profession and the preferred style of clothes of a set of citizens, are correlated or not. In our context, we are interested to the goodness-of-fit test, i.e. how well the observed data point x fits the multivariate Gaussian $\mathcal{N}(\mu, \Sigma)$. Therefore, we formulate the null hypothesis H_0 that the point belongs to the distribution. Then, we impose a confidence level, for example, 90% ($\alpha = 0.01$) and we compare $D(x, \mu)$ against the critical value extracted by looking at a χ^2 with N degrees of freedom, in this case two. In Figure 3.10, an instance of this test is demonstrated an instance of this test, where the points are drawn by two different Gaussian distributions and the ellipse models the threshold for the Mahalanobis distance. If the resulting distance is less than the critical value, as it happens for the points inside the depicted area, we do not refuse the null hypothesis H_0, which, in the landmark test, translates as a positive matching. Instead, if the distance is greater, as in the case of points outside the marked perimeter, we pass to the next landmark for continuing the comparison.

3.2.4. Implementation details

In this section, some details specific to our implementation are introduced, in particular about the initialization procedure for the filter and some additional steps taken in prediction and the update phase.

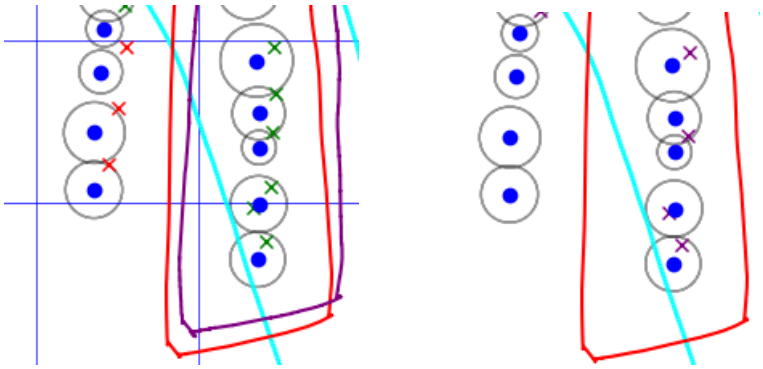
Initialization The filter stores the state μ in a vector of elements, initially 100, which is expanded only when needed, so that to avoid the reallocation of elements each time a new landmark is discovered. The same holds for the covariance matrix. At the beginning, the state of the robot is initialized to the first received pose or a starting custom pose if specified. The start covariance for the pose and landmarks are filled with given values for the process and sensor noise respectively.

Prediction and update steps During the prediction step, not all the odometry updates are integrated in the filter as the oversampling during turns might register slight translations of the robot which result a wrong estimate for δ_{rot1}. Therefore, the movements have been integrated only when surpassing a given threshold for translation. Regarding the update, the recently seen landmarks are stored in a vector along with some data describing the landmark which are used for both understanding the importance of the landmark and to avoid continuous queries to the state vector. In particular, these data are composed by: a trust value, which counts the number of times the landmark has been seen; its lives, i.e. a counter decremented whenever the landmark is not detected in the current scan and which is restored when it is seen again; the landmark index inside the state vector. The first value is employed to know whether the landmark has been seen enough times to be considered as belonging to a plant and is tuned according to the specific necessities. In the case it is not, then it is removed by the filter. The second one gives us an insight on how old the landmark is: if it has not been seen for quite some time, it would be below zero. At this point the landmark position is checked against the robot one and if the landmark is farer than a specified threshold with respect to the robot, the landmark is removed from the local landmark vector.

3.3. Plant counting

The last part of the project is about the actual counting of the plants in the map generated by the EKF-SLAM algorithm. We decided to leave this operation at the end, since, sometimes, the landmark data association might fail producing near, but different, marking points for the same plant.

In particular, we considered as true positives for the plant i for which at least a landmark is located within half of the distance between the ground truth position of the plant in the map and the successive adjacent plant. The false positives class, instead, includes all the double-markings for each plant and those landmarks which are not near to any of the plants within the ground truth map. To effectively extract the number of plants correctly counted, we performed a "cleaning" post-processing procedure to remove all the false positives and maintain only the nearest landmark to the center of the corresponding plant. An example of this process is shown in Figure 3.11. As it can be seen two landmarks have been removed, one from the second plant in the row on the right, starting from the bottom, and one from the fourth plant on the same row. Also the missed detections have been removed from the final map.



(a) (b)

Figure 3.11: Result of the post-processing cleaning operation performed over the mapped plants. The blue points are the ground truth position of the plants, the circles the limit distance within a point is considered as referring to that plant, the green crosses the ekf landmarks belonging to a plant and the red ones the missed detections

4 | Experimental Results

In this section, the methodology followed to design the experiments for the system are presented. To run tests over some real case scenario, we looked for datasets providing at least a 3D LiDAR point cloud, an odometry source and possibly a ground truth trajectory to assess the error of the estimated one. We took inspiration from the Bacchus Long-Term Dataset (BLT) and from a real orchard nursery environment to build a simulated world in Gazebo. Our goal was to adjust all the possible parameters in the simulation, especially regarding the EKF-SLAM, to then test the same values over the dataset.

4.1. BACCHUS Long-Term Dataset

The BLT Dataset [12] is a collection of ROS bags (see subsection 2.4.1) recorded for SLAM algorithms evaluation. It has been recorded from a vineyard located nearby the town of Epanomi, in Greece, and it temporally spans from March to September 2022, for a total of 10 record sessions. All the runs consisted in the traversal of the field multiple times, observing each plant row from the both its long sides for the process of visual odometry estimation. The data acquisition was a conducted by using a Thorvald, a four-wheel drive and steer (4WD4S) robotic platform, equipped with a variety of sensors as shown in Figure 4.1. For our purposes we did use:

- The Ouster OS1-16: a 3D LiDAR mapping 16 planes, publishing on the topic `/os_cloud_node/points`;
- A StereoLabs Zed2: an RGB-D cameras providing both images of the field and the computation of the visual odometry. Out of the two nodes mounted on the robot, one on the front and one on the side, we used the former, which publishes the computed odometry on the topic `/front/zed_node/odom`;
- The Trimble BX992: an RTK-GPS which we used as ground truth for the trajectory for the comparison the one estimated by our system. The odometry completed with the orientation is given by the topic `/odometry/gps`.

In each of the bags, the robot follows the same path which can be explained by looking

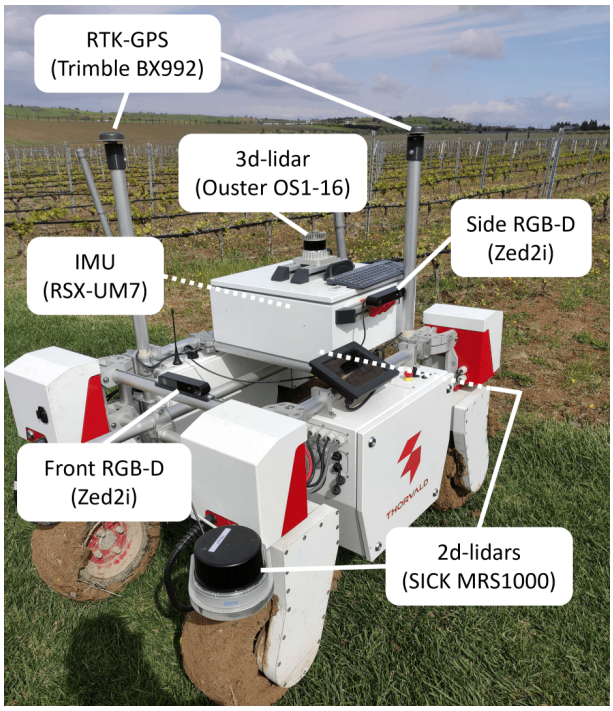


Figure 4.1: The robotic platform (taken from [12])

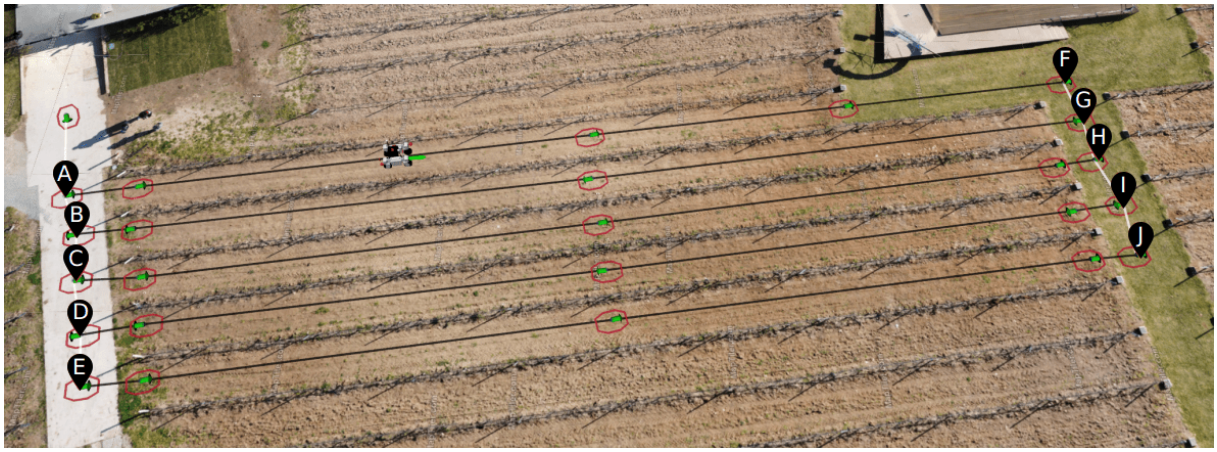


Figure 4.2: Field traversal (taken from [12])

at Figure 4.2. In particular, it starts from the point A, it explores the second, B-G, and then the first, F-A, rows by tracing the path A-B-G-F-A-B-G. Then, traverses the third, H-C, and fourth, D-I, rows and it proceeds with the last one, J-E.

For our test we used the bag produced on April the 6th, 2022, as the vineyard plants did not have leaves and the trunk was well visible. The GPS and visual odometry trajectories are reported in Figure 4.3, where both of them where rotated and shifted near each other for allowing an easier visualization and comparison.

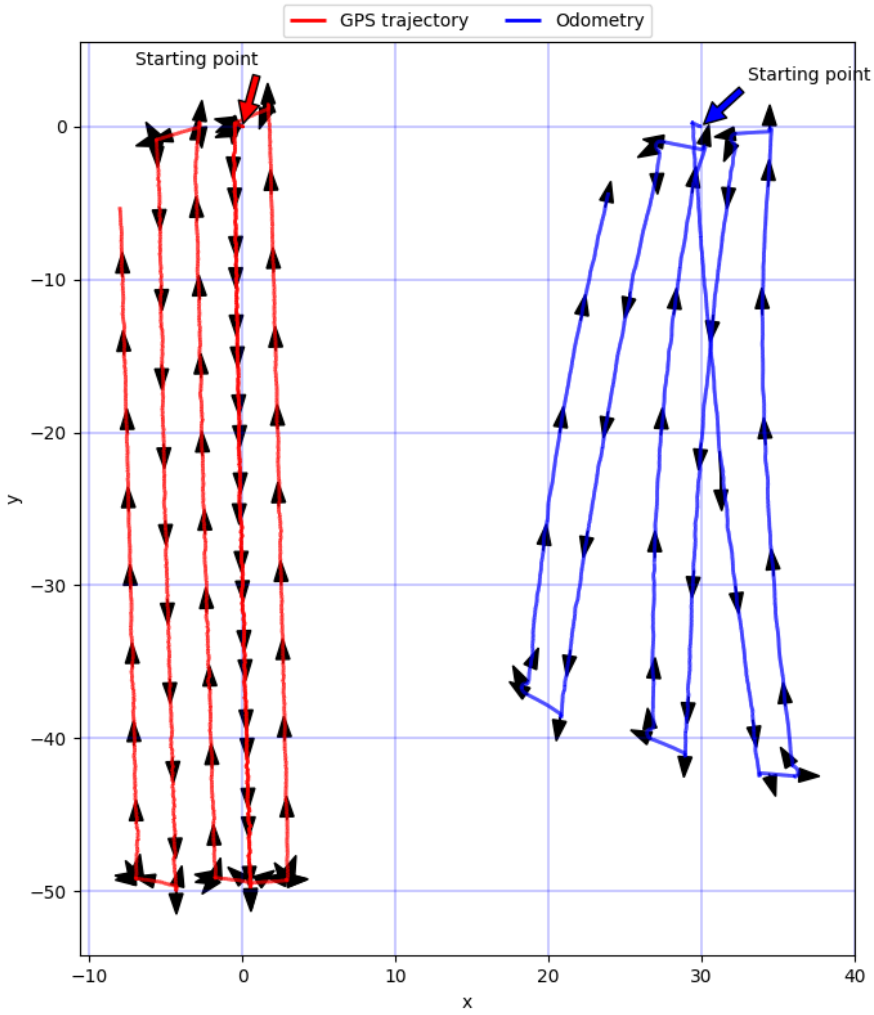
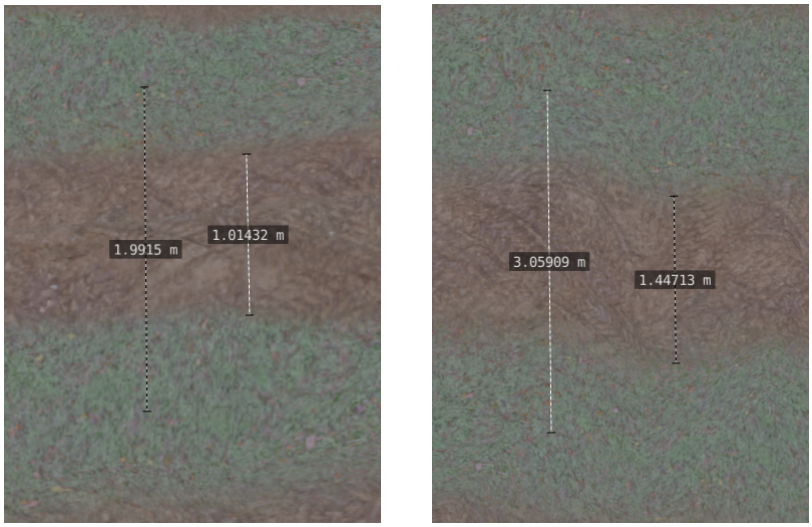


Figure 4.3: GPS and visual odometry trajectory comparison. The label starting point indicates a location near to A in Figure 4.2

4.2. Simulating the real world

By observing the BLT dataset and a real orchard nursery environment, we built a simulation in Gazebo (see Section 2.4.2) to test the functionalities of our algorithms. We tried to replicate as close as possible the visual odometry behaviour so to later export and check the effects of the EKF-SLAM parameters found in the simultation in the real world scenario.



(a) (b)

Figure 4.4: Difference between the original orchard terrain (on the left) and the one used in the test (on the right)

4.2.1. World construction

For the creation the world, we started by choosing the ground plane. We opted for the Clearpath robotics orchard terrain[1] which already provides friction parameters, which were left unchanged, bumps and a texture separating the plant rows. We removed the tree models provided with the world and we reduced the in-row width, which measured about 1 meter in the dataset, as demonstrated in Figure 4.4.

Regarding the plant models, we searched for small ones similar to those in nurseries. The majority of the free available models, however, represented adult trees of various species which did not really fit our needs. At the end, we selected a Mangrove tree model[2] of which we used the top part. The resulting orchard is displayed in 4.5a and it is composed by a total of 78 plants distributed over 6 rows. The trees were positioned at different heights and rotated differently so to introduce variation in the sensing. Their distance was selected by observing the ones in the dataset and ranges from 40cm to 96cm.

4.2.2. Robot model

The robotic platform employed for the exploration of the world is a scout[3], a four-wheel differential steering robot by Agilex Robotics. We mounted an Ouster-1 64 planes 3D

[1] https://github.com/clearpathrobotics/cpr_gazebo
[2] https://sketchfab.com/3d-models/tall-mangrove-tree-b722ee2895e84fd3b02a467cd1ba1cbe
[3] https://github.com/agilexrobotics/ugv_gazebo_sim

(a) (b)

Figure 4.5: The simulated world (on the left) and the robot model (on the right)

LiDAR[4] above a 10cm high support positioned on the top of the robot. The elevation of the sensor served to increase its field of view and reduce the number of points detected from the robot. The final model is illustrated in Figure 4.5b.

4.2.3. Odometry generation

The odometry messages broadcasted by Gazebo contain the actual position of the robot in the world, without any additional noise. This was not a feature intended for our system, since the visual odometry, and, in general, any odometry estimate, is far from being perfect. Hence, we decided to use a ROS node to manually generate a noisy odometry similar to the one computed by the Zed nodes.
We started by aligning the odometry and GPS traced trajectories and by observing how much the former was behaving with respect to the latter in the first ten meters, which are shown in Figure 4.6:

Then, we employed the odometry motion model (see section 3.2), but differently from the EKF-SLAM motion model, we used its original Gaussian noises. In particular, the α parameters defined by model affect different components, whose relations are synthesised in Table 4.1.

We manually adjusted the noise parameters to try to obtain the desired level of deviation of the odometry from the ground truth trajectory. They were tuned by taking into consideration the distance which was between one visual odometry update and the successive, of about 4cm. Hence, we slowed down the odometry given by Gazebo using an additional

[4] `https://github.com/ntnu-arl/lidar_simulator/tree/master`

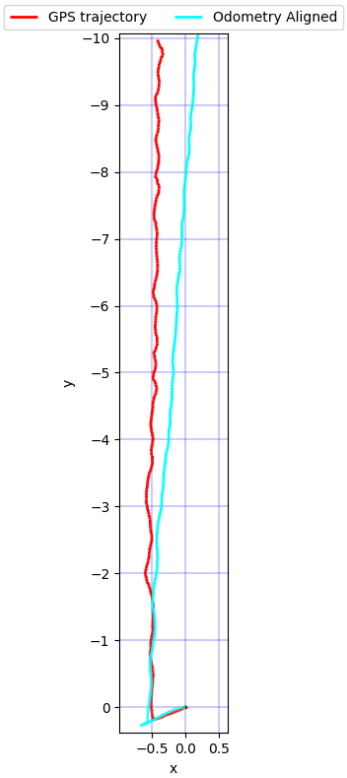


Figure 4.6: Visual comparison of the first ten meters of the visual odometry and the GPS trajectory

	δ_{rot1}	δ_{trans}	δ_{rot2}
δ_{rot1}	α_1	α_2	-
δ_{trans}	α_4	α_3	α_4
δ_{rot2}	-	α_2	α_1

Table 4.1: Alphas parameters included in the computation of the odometry motion model standard deviations. On the rows the affected component of the model and on the columns the affecting one

ROS node to reach a similar value.
Regarding the differences between the visual odometry and the GPS, as we can see from Figure 4.6 even if the robot moves, more or less, along a straight trajectory, the estimates gets farer and farer away from it. Therefore, we imposed a large α_2 parameter to make the translation the prevalent component in the noise modifying the rotations. The selected parameters are reported in Table 4.2 and a set of odometries generated can be seen in **??**.

Moreover, as in explained in 3.2.4 because of the frequency at which the odometry is published, we did not integrated each update directly in the odometry construction, but we agglomerated those resulting in a robot translation inferior to a given threshold to

alpha1	0.00001
alpha2	0.03
alpha3	0.0001
alpha4	0.0000002

Table 4.2: Odometry motion model noise parameters

solve the problem of oversampling of turns which resulted in an high noise generated by the motion model.

4.3. Results and discussion

In this section, the metrics used to evaluate the system performances are presented along with the results obtained for the simulation and the real scenario.

4.3.1. Simulation Metrics

The metrics employed in the simulation have been chosen to evaluate both the goodness of the generated map, the estimated trajectory. Starting from the first, we counted the number of true positives (TP), false positives (FP) and false negatives (FN) as described in Section 3.3. We did not include the true negatives (TN) as they can not be defined in a context where there are no other classes aside from plants which can be detected.
In particular, we defined as true positives those plants which had at least one landmark within their proximity. The false positives category embraced all those detections which were considered as a double count of the same plant and also those falling outside the proximity range of all the plants. Finally, we marked as false negatives the plants which did not have any landmark in their vicinity. After the division of plants in these categories, we computed both the precision and the recall:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

The precision is a measure of the number of the real plants detected out of all those elements, actual plants and not, counted as plant by the system. Instead, the recall indicates the number of effective plants marked out of all the plants present in the environment. Both of them can range between 0 and 1, where a value becomes better as it approaches 1.
Apart from the counting itself, we also measured, by keeping only those landmarks near-

est to their respective ground truth plant, the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE), defined as following:

$$MAE = \frac{1}{N}\sum_{i=1}^{N-1}|y_i - \hat{y}_i| \tag{4.3}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N-1}(y_i - \hat{y}_i)^2} \tag{4.4}$$

The former refers to the average of the distances between the landmarks and the ground truth, whereas the latter accounts for the standard deviation of the residuals, meaning how spreaded the landmarks are with respect to the real position of plants. Both are positive and measured in meters and a smaller value implies a better accuracy in the detections.

For estimating the trajectory error, we used the mean distance metric, measured in meters. Since the perfect position of the robot in the simulated environment and the RTK-GPS in the BLT dataset where both published at an higher frequency in comparison to the noisy odometry updates, we performed a linear interpolation on the EKF-SLAM estimated trajectory by using the timestamps as a way to account for the missing positions. Given two points (x_0, y_0) and (x_1, y_1), by using the linear interpolation we can compute the value for y of a point x in between x_0 and x_1 by using the formula:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) \tag{4.5}$$

4.3.2. Simulation Results

By observing the behavior of the system over the registered point cloud, we manually adjusted the plane, cluster filter and EKF-SLAM parameters which can be viewed on Table 4.3.
On the plant detection side, the minimum plant height was set to 0 since there were no clusters produced by the ground filtering part. Also, the ground plane filter was set with a threshold of 20cm to account for discontinuities and bumps which the terrain model presented, which in some cases, reduced the visible portion of the plant. The other two parameters for cluster filtering, the width and the maximum number, have been chosen based on the characteristic of the plant model represented in the point cloud.
As for the EKF-SLAM parameters, different combinations have been tried, but the ones resulting in Table 4.3, were deemed as those performing the best.

Maximum deviation angle (deg):	8.0
Inliers threshold (m):	0.2
Cluster tolerance (m):	0.15
Minimum cluster number:	4
Maximum cluster number:	100
Maximum plant width (cm):	0.35
Minimum plant height (cm):	0.0

(a) Plane and plant detection parameters

Sensor noise: (cm):	22.36
Process noise on x (cm):	3.16
Process noise on y (cm):	3.16
Process noise on θ (deg):	1.81
Initial process noise on position (x, y) (cm):	0.01
Initial process noise on θ (deg):	1.81
Initial landmarks covariance:	0.8
Initial θ (deg):	90

(b) EKF-SLAM parameters

Table 4.3: Parameters adjusted for both the EKF-SLAM, the plane and the plant detection algorithms. The EKF-SLAM paramters are reported in standard deviation.

To test the simulated environment, we run the system on 10 different generated noises trajectories. For each of them, we used a pseudo-random number generator for drawing the Gaussian noises, which we fixed by using a given seed for the repeatability of results. The details of each run are presented in Table 4.4.
Overall, the EKF-SLAM algorithm demonstrated to not be very robust to the changes in the noise introduced. Indeed, while in few cases it responded very well by producing an almost perfect counting, in others the performances degraded considerably, with a precision of 37.4% or less. By observing the odometries generated, we noticed that the algorithm performed better on cases where the noisy trajectory was very similar to the ground truth, especially in the few meters.

When observing these results, we have to consider that while the motion model used to generate the noisy odometry employed the distances travelled by the robot in the computation, the motion model of the filter is bounded to have fixed process noises, which might have been too small for some movements.
Moreover, also the number of detections obtained for each plant might have had a role in these performances. Indeed, even if the ground was completely removed from the scene, the imposed threshold reduced the portion of visible plants and therefore the possibility for the system to formulate a cluster.

Run No.	TP	FP	FN	Precision	Recall	MAE(m)	RMSE(m)	Mean distance (m)
1	77	2	1	0.975	0.987	0.0975	0.11	0.16
2	41	41	37	0.5	0.526	0.17	0.187	0.313
3	76	7	2	0.916	0.974	0.11	0.126	0.19
4	30	49	48	0.38	0.385	0.164	0.188	0.493
5	52	33	26	0.612	0.66	0.164	0.185	0.37
6	45	38	33	0.54	0.577	0.16	0.177	0.29
7	78	5	0	0.94	1	0.097	0.108	0.153
8	21	70	57	0.231	0.27	0.12	0.184	0.44
9	51	31	27	0.62	0.654	0.16	0.18	0.407
10	76	10	2	0.88	0.97	0.10	0.116	0.164

(a) Original results

Run No.	TP	FP	FN	Precision	Recall
1	77	0	1	1	0.987
2	41	37	37	0.526	0.526
3	76	3	2	0.96	0.97
4	30	48	48	0.385	0.385
5	52	29	26	0.642	0.66
6	45	33	33	0.577	0.577
7	78	0	0	1	1
8	21	63	57	0.25	0.27
9	51	30	27	0.63	0.654
10	76	4	2	0.95	0.97

(b) Post processed results

Table 4.4: Metrics computed over each run of the system in simulation. These results have been obtained by fixing a seed from 10 to 19 included.

Precision	0.65
Recall	0.70
Precision (Post-process)	0.69
Recall (Post-process)	0.7
MAE (m)	0.174
RMSE (m)	0.15
Mean distance (m)	0.298

Table 4.5: Average metrics obtained from the simulation

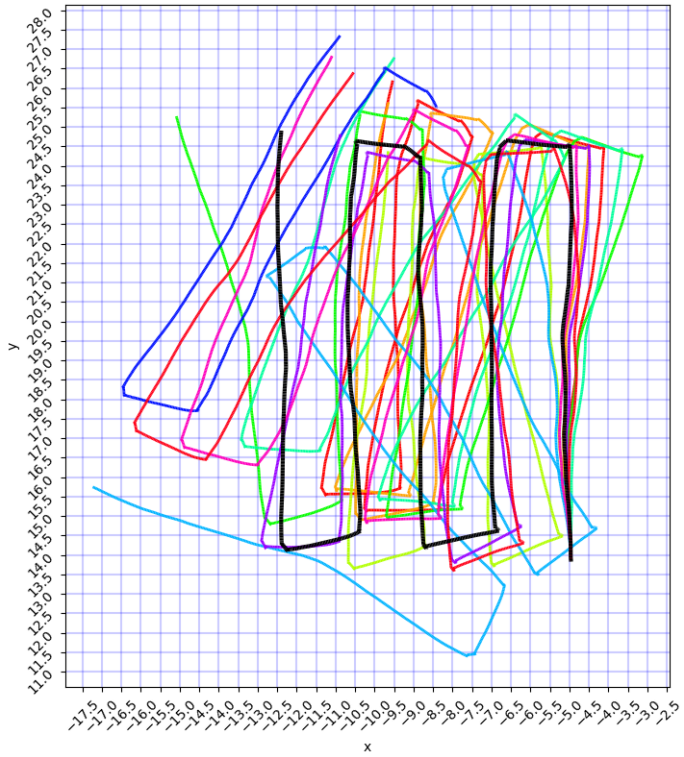
Although the presented system scored less than the other presented approaches which, however, did not perform SLAM, it demonstrated to be able to filter the noise produced by a particular run, Figure 4.8, which is similar in the first 10 meters to the one illustrated in Figure 4.6. The average for each presented metric is reported in **??**.

4.3.3. Real world Results

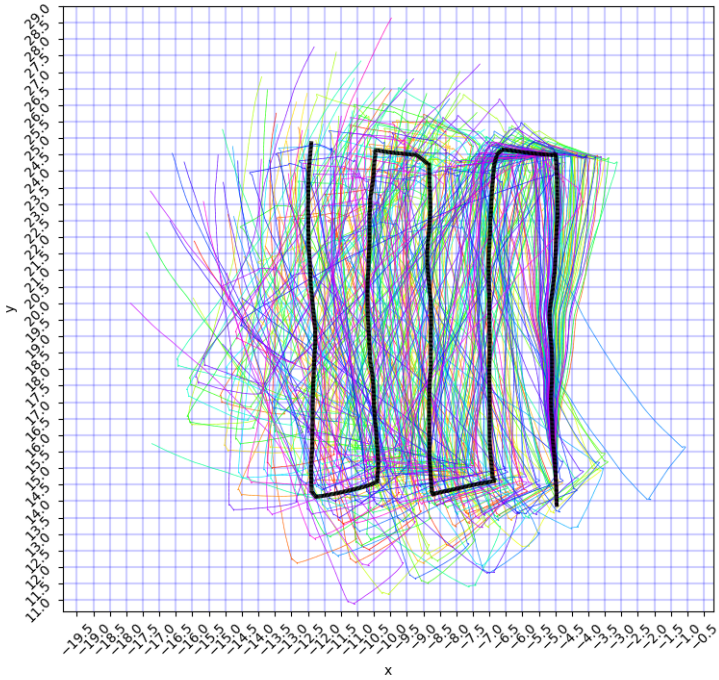
Regarding the real world case, we did not have a map of the vineyard available. We tried to estimate one manually by observing the robot poses and the point cloud, but the transformation between the GPS coordinate frame and the odometry coordinate frame was not provided, so we could not build it. Also, the messages recorded in the dataset did not present sensors synchronized in a proper way, so we could not directly assess an average distance between the odometry and the GPS traced trajectory. Therefore, we plotted the estimated trajectory against the GPS and the starting visual odometry to assess its goodness.
The plant detection phase failed on some plants, as the number of points representing them was less compared to the one used in simulation (the LiDAR used in these recording is a 16-plane one). Also, the algorithm was not able to distinguish between plant, poles and other environmental elements, as the plant detection phase does not consider such eventuality. As for the EKF-SLAM, the parameters found during the simulation have been used as an initial guess for the ones needed in the real case. We had to slightly modify them to account for the presence of noises not modelled by our simulation, such as the one causing the underestimation of more than 5 meters in the first straight line, beginning from the "starting point" and going down until the first turn, of the distance travelled computed by the visual odometry, whose comparison against the GPS trajectory is showed in 4.9a. The changed parameters regard the process noises which have been increased to 10cm for x,y and to 3.6 degrees for θ. The EKF-SLAM managed to produce a better trajectory with respect to the visual odometry, by correcting the length of the path in the first straight line. However, the robot the number of landmarks mismatched by the robot was very high. Especially when it turned back, it started to mistakenly associate landmarks to different plants in the point cloud with respect to those which originated them. This is natural consequence of the fact that the algorithm can not distinguish during the process of plant matching one plant from the other if the landmarks are too near each other.

process noise on x (cm):	10
process noise on y (cm):	10
process noise on θ (deg):	3.62
initial process noise on position (x, y) (cm):	0.01
initial process noise on θ (deg):	1.81
initial landmarks covariance:	0.8

Table 4.6: EKF-SLAM parameters set for the Greek vineyard environment

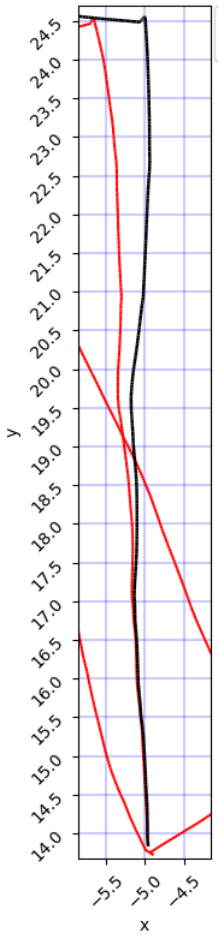


(a) Ten odometries generated



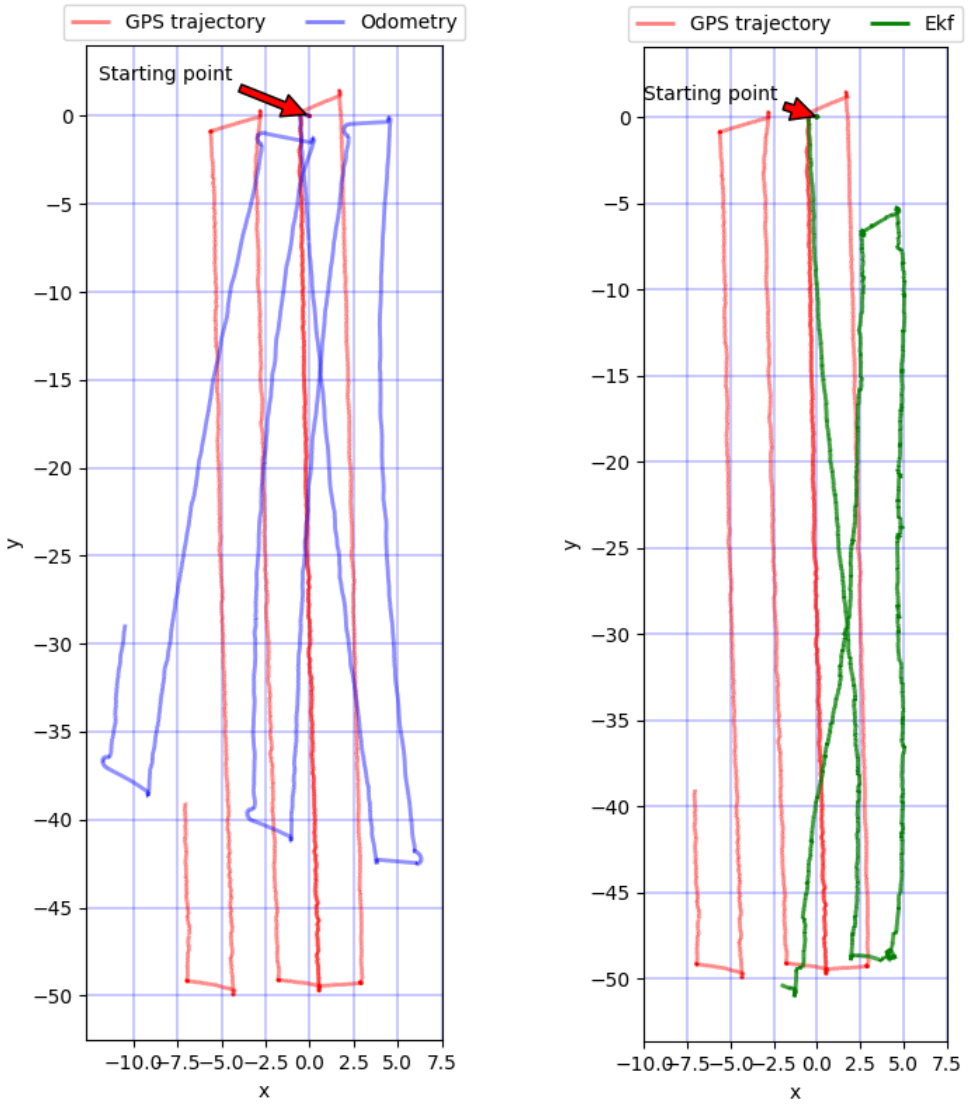
(b) First 100 odometries generated

Figure 4.7: Odometries generated by applying the odometry motion model to the ground truth, depicted in black



(a)

Figure 4.8: One of the generated odometries (run 1 in result table), which resulted to be similar to Figure 4.6 where the ground truth is depicted in black



(a) Visual odometry versus GPS

(b) EKF odometry versus GPS

Figure 4.9: Illustration of the comparisons deriving from the test case over the BLT dataset.

5 | Conclusions and future developments

In this work, an approach to plant counting using the EKF-SLAM algorithm was proposed. The automation of such task would benefit the agricultural field and in particular orchard nurseries, where a continual reassessment of the stock availability is needed in order to take management decisions. Indeed, nowadays this task is performed manually, with high costs in terms of time and money for the farmers.

After researching the existing literature on the topic, we have decided that the most suitable sensor for this activity would have been a 3D LiDAR. This choice was based on natural characteristics of an outdoor environment, such as changes in the light conditions throughout the year and irregularities in the terrain, which pose challenges in the application of camera-based and infrared-based platforms.

We developed a system which can potentially be mounted on existing agricultural vehicles, thus relieving the cost that a stand-alone robot would require. Our pipeline for the plant counting started with the filtering of the point cloud returned by the sensor and the detection and elimination of the ground plane from the scene. We then filtered and clustered the remaining points and considered the groups of points formed by DBSCAN as possible plants. We computed their baricenters and used them as landmarks to perform SLAM for the simultaneous estimate of a map of the field and the correction of the noisy odometry source available to the robot. The motion model applied to the EKF-SLAM system was the odometry motion model used to predict the next state of the filter starting from the current one. We stored the landmarks as couples of (x, y) coordinates and matched them to the plant detections depending on the results of a χ^2 test. Finally, we counted the number of plants marked on the map as last step to address the problem of double-counting which an EKF-SLAM algorithm might produce.

For evaluating the system, we built a simulated orchard environment based on the observation of a real one. Our objective was to fine-tune the noises parameters for the mapping algorithm on the simulation to later export them over the real case. Regarding the simulations, we generated different noises and assessed the system results based on different

metrics, namely the number of true positives, false positives, false negatives, precision and recall, mean absolute error and root mean squared error and finally the mean distance. The results over the simulation has shown that the algorithm is not robust against changes in the noise applied in the odometry estimate. On average, it scored 70% on both precision and recall after the removal of redundant landmarks marked for the same plant. This value is smaller with respect to the other analysed approaches, which, however did not perform SLAM, but had direct access to a GPS signal, which is not always possible to exploit in an orchard environment. On the real world case, due to the lack of an existing map and a way to properly compute it, we relied on a visual assessment of the proposed method. After increasing the process noises found during the simulation to account for not modelled noises, we observed that the EKF-SLAM managed to correct the travelled distance estimated through the visual odometry. However, the mismatching of landmarks at the end of the second turn caused the system to derail and generate new landmarks for previously seen plants. This fact was probably due to the inability of the algorithm to correctly match the landmarks to the plants which originated them as the association process is based on the assumption that the detection relative to a landmark is also the nearest to it in terms of their similarity with respect to the χ^2 test.

5.1. Future work

Different aspects of this project can be improved, starting from the plant detection algorithm, which is not able to distinguish between a plants from other elements which might be present in the environment, such as plant poles, ground and walls. This problem might be addressed with the help of other sensors to discern elements of different colors or shape. Moreover, the next step of the algorithm could also result in the assessment of the plant status, so that the plants which have perished can be counted and located as a different category from the healthy ones. This would represent an additional help for the farmers, which could efficiently find those plants and take action if necessary. Also, it would enlarge the capabilities of the system to perform various plant monitoring activities.
Regarding the EKF-SLAM, there is room for the improvement in the results both in the matching of landmarks to detections and in the correction of the trajectory. A possible continuation could be the estimate of row lines from the LiDAR, by using computer vision algorithms such as the Hough transform, and their usage to correct the odometry, in place of single plants. This would bring benefits to the system, as the lines would be straight, maintaining the same distance from the robot once the direction is fixed and thus solving problems such as the curve estimated during the following of a straight trajectory.

Bibliography

[1] K. A. S. Birrel, S.J. Corn population sensor for precision farming. *ASAE*, (951334).

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[4] S. Fountas, N. Mylonas, I. Malounas, E. Rodias, C. Hellmann Santos, and E. Pekkeriet. Agricultural robotics for field operations. *Sensors*, 20(9), 2020. ISSN 1424-8220. doi: 10.3390/s20092672. URL https://www.mdpi.com/1424-8220/20/9/2672.

[5] M. Garrido, M. Perez-Ruiz, C. Valero, C. J. Gliever, B. D. Hanson, and D. C. Slaughter. Active optical sensors for tree stem detection and classification in nurseries. *Sensors*, 14(6):10783–10803, 2014. ISSN 1424-8220. doi: 10.3390/s140610783. URL https://www.mdpi.com/1424-8220/14/6/10783.

[6] M. C. Hunter, R. G. Smith, M. E. Schipanski, L. W. Atwood, and D. A. Mortensen. Agriculture in 2050: Recalibrating Targets for Sustainable Intensification. *BioScience*, 67(4):386–391, 02 2017. ISSN 0006-3568. doi: 10.1093/biosci/bix010. URL https://doi.org/10.1093/biosci/bix010.

[7] E. Kayacan, Z. Zhang, and G. Chowdhary. Embedded high precision control and corn stand counting algorithms for an ultra-compact 3d printed field robot. 06 2018. doi: 10.15607/RSS.2018.XIV.036.

[8] J. Lowenberg-DeBoer and B. Erickson. Setting the record straight on precision agriculture adoption. *Agronomy Journal*, 111(4):1552–1569, 2019. doi: https://doi.org/10.2134/agronj2018.12.0779. URL https://acsess.onlinelibrary.wiley.com/doi/abs/10.2134/agronj2018.12.0779.

[9] J. Luck, S. Pitla, and S. Shearer. Sensor ranging technique for determining corn plant population. 06 2008.

[10] U. N. D. of Economic and S. Affairs. World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100, 2023. URL `https://www.un.org/en/desa/world-population-projected-reach-98-billion-2050-and-112-billion-2100`.

[11] U. N. D. of Economic and S. Affairs. World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100, 2023. URL `https://www.un.org/en/desa/world-population-projected-reach-98-billion-2050-and-112-billion-2100`.

[12] R. Polvara, S. Molina, I. Hroob, A. Papadimitriou, T. Konstantinos, D. Giakoumis, S. Likothanassis, D. Tzovaras, G. Cielniak, and M. Hanheide. Blt dataset: acquisition of the agricultural bacchus long-term dataset with automated robot deployment. *Journal of Field Robotics, Agricultural Robots for Ag 4.0*. Under Review.

[13] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[14] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3), jul 2017. ISSN 0362-5915. doi: 10.1145/3068335. URL `https://doi.org/10.1145/3068335`.

[15] N. Shalal, T. Low, C. McCarthy, and N. Hancock. Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion – part a: Tree detection. *Computers and Electronics in Agriculture*, 119:254–266, 2015. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2015.09.025. URL `https://www.sciencedirect.com/science/article/pii/S0168169915003002`.

[16] N. Shalal, T. Low, C. McCarthy, and N. Hancock. Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion – part b: Mapping and localisation. *Computers and Electronics in Agriculture*, 119:267–278, 2015. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2015.09.026. URL `https://www.sciencedirect.com/science/article/pii/S0168169915003014`.

[17] T. R. K. R. W. R. H. J. A. Shi Yeyin, Wang Ning. Automatic corn plant location and spacing measurement using laser line-scan technique. *Precision Agriculture*, 14: 478–494, 10 2013. ISSN 1573-1618. doi: https://doi.org/10.1007/s11119-013-9311-z. An optional note.

[18] J. H. J. Spiertz. Nitrogen, sustainable agriculture and food security. a review. *Agronomy for Sustainable Development*, 30(1):43–55, Mar 2010. ISSN 1773-0155. doi: 10.1051/agro:2008064. URL https://doi.org/10.1051/agro:2008064.

[19] J. E. Taylor, D. Charlton, and A. Yúnez-Naude. The end of farm labor abundance. *Applied Economic Perspectives and Policy*, 34(4):587–598, 2012. ISSN 20405790, 20405804. URL http://www.jstor.org/stable/23356432.

[20] S. G. Vougioukas. Agricultural robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):365–392, may 2019. doi: 10.1146/annurev-control-053018-023617. URL https://doi.org/10.1146%2Fannurev-control-053018-023617.

[21] U. Weiss and P. Biber. Plant detection and mapping for agricultural robots using a 3D LIDAR sensor. *Robotics and Autonomous Systems*, 59(5):265–273, May 2011. ISSN 09218890. doi: 10.1016/j.robot.2011.02.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0921889011000315.

[22] C. Wellington, J. Campoy, L. Khot, and R. Ehsani. Orchard Tree Modeling for Advanced Sprayer Control and Automatic Tree Inventory. page 8.

A | Appendix A

A.1. Code organization

The package is structured in the following way:

- **config**: contains the configuration files in `.yaml` format. Files included:
  - `gazebo_param.yaml`: Parameter file for Gazebo simulation
  - `greek_vines_param.yaml`: Parameter file for the BACCHUS dataset
- **launch**: contains the launch files in `.launch` format. Files included:
  - `orchard_world.launch`: Launch file for Gazebo simulation
  - `bags.launch`: Launch file for bags registered from the simulation
  - `bags_ext_greek.launch`: Launch file for BACCHUS dataset bag
- **models**: contains the model folders for the tree used in simulation. Each model has its own folder.
- rviz: contains rviz configuration files in `.rviz` format.
- **src**: contains the nodes source files in `.cpp` format.
  - `cluster_filter.cpp`: node performing the recognition of plants;
  - `ekf.cpp`: node running the EKF-SLAM algorithm;
  - `noisy_odom_motion_model.cpp`: node generating the perturbed odometries;
  - `odometry_throttler.cpp`: node publishing the odometry topic at a reduced rate;
  - `plane_filter.cpp`: node performing the filtering of the plane;
  - `pose_subscriber.cpp`: node subscribing to the given topics for odometry and map storing in .csv files;

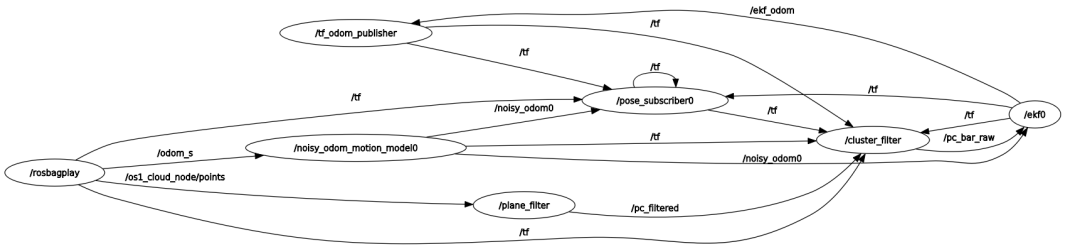


Figure A.1: ROS interaction graph

  - `tf_odom_publisher.cpp`: node publishing the transformation from ekf odometry frame to base_link;

- **urdf**: contains the urdf models used to build the robot and sensor in simulation. Two format are present: `.urdf.xacro` which describes the shapes, links and joints of the sensors and the robot and `.gazebo.xacro` which defines the plugins used by the sensor during the simulation;
- **worlds**: contains the gazebo worlds used for the simulations in `.world` format.

A.2. ROS Nodes

A brief presentation of the nodes in which the whole process is divided is presented in the following, along with the input and output topics, and the configurable parameters. A schematic of the ROS interactions can be seen in Figure A.1.

A.2.1. plane_filter

It estimates the plane model from the input point cloud and outputs the filtered cloud.

Input topics:

- `/os1_cloud_node/points`: the input point cloud.
  Message type: `sensor_msgs::PointCloud2`

Output topics:

- `/pc_filtered`: the filtered point cloud without the ground plane
  Message type: `sensor_msgs::PointCloud2`

Filter parameters:

- `min_pt`: Minimum coordinates for the filtering operation. It is a dictionary contain-

ing `x`, `y`, `z` coordinates and `i`, the intensity;

- `max_pt`: Maximum coordinates for the filtering operation. It is a dictionary containing `x`, `y`, `z` coordinates and `i`, the intensity;

RANSAC parameters:

- `eps_angle`: Maximum deviation angle (in radiants) the plane can be inclined to with respect to the specified axis;
- `threshold`: Maximum distance of points to the plane to be considered as inliers;
- `max_iterations`: Maximum number of iterations for repeating the searching operation

Plane properties parameters (optional):

- `use_plane_z`: when set to true, the node uses the additional constraint for the intersection of the found plane with the z axis;
- `plane_z`: Intersection point with the z axis;
- `distance_t`: maximum distance from the intersection point;

A.2.2. cluster_filter

It takes the filtered cloud and outputs the cluster location in the sensor coordinate frame.

Input topics:

- `/pc_filtered`: the filtered point cloud coming from plane_filter node;
  Message type: `sensor_msgs::PointCloud2`

Output topics:

- `/pc_bar_raw`: the clusters baricenters in sensor frame.
  Message type: `sensor_msgs::PointCloud`
- `/pc_bar_right_db`: the clusters baricenters in base frame on the right of the robot. Added for visualization purposes.
  Message type: `sensor_msgs::PointCloud`
- `/pc_bar_left_db`: the clusters baricenters in base frame on the left of the robot.
  Message type: `sensor_msgs::PointCloud`

Filter parameters:

- `min_pt`: Minimum coordinates for the filtering operation. It is a dictionary containing `x`, `y`, `z` coordinates and `i`, the intensity;
- `max_pt`: Maximum coordinates for the filtering operation. It is a dictionary containing `x`, `y`, `z` coordinates and `i`, the intensity;
- `robot_min_left`: Maximum y coordinate at which the robot is still visible;
- `robot_min_right`: Minimum y coordinate at which the robot is still visible;

Cluster properties parameters:

- `min_plant_height`: Minimum cluster height for it to be considered as a candidate plant;
- `max_plant_width`: Maximum cluster width for it to be considered as a candidate plant;
- `max_cluster_n`: Maximum number of points in a cluster for it to be considered as a candidate plant;

Frame names:

- `base_frame`: Base robot frame. Default: `base_link`;
- `sensor_frame`: Sensor frame. Default: `os1_sensor`;

DBSCAN prameters:

- `cluster_tolerance`: Maximum distance for searching for neighbours;

A.2.3. ekf

Input topics:

- `odom_topic`: the input odometry topic.
  Message type: `nav_msgs::Odometry`
- `/pc_bar_raw`: the clusters baricenters in sensor frame
  Message type: `sensor_msgs::PointCloud`

Output topics:

- `/ekf_pose`: the estimated pose
  Message type: `geometry_msgs::PoseWithCovarianceStamped`

- `/ekf_odom`: the ekf odometry topic
  Message type: `nav_msgs::Odometry`
- `/ekf_landmarks`: the landmarks pose
  Message type: `visualization_msgs::Marker`
- `/ekf_landmarks_cov`: the landmarks covariance
  Message type: `visualization_msgs::MarkerArray`

Odometry parameters

- `eps_angle`: Maximum deviation angle (in radiants) the plane can be inclined to with respect to the specified axis;
- `threshold`: The maximum distance of points to the plane to be considered as inliers;

EKF parameters:

- `start_x`: Start x coordinate for the filter initialization. If not provided the x is set to the first odometry message received;
- `start_y`: Start y coordinate for the filter initialization. If not provided the y is set to the first odometry message received
- `start_theta`: Start orientation (in radiants) for the filter initialization. If not provided the orientation is set to the first odometry message received
- `landmark_cov`: Start landmark covariance when a landmark is first discovered;
- `start_cov_xy`: Start position covariance (in meters);
- `start_cov_theta`: Start orientation covariance;
- `x_cov`: Process noise covariance (in meters) over x;
- `y_cov`: Process noise covariance (in meters) over y;
- `theta_cov`: Process noise covariance (in radiants) over the orientation;
- `obs_noise`: Sensor noise covariance (in radiants);

A.2.4. pose_subscriber

Input topics:

- `noisy_topic`: Noisy odometry topic
  Message type: `nav_msgs::Odometry`

- `odom_topic`: Odometry topic
  Message type: `nav_msgs::Odometry`
- `ekf_topic`: EKF-SLAM odometry topic
  Message type: `nav_msgs::Odometry`
- `gt_topic`: Name of the ground truth odometry topic
  Message type: `nav_msgs::Odometry`
- `map_topic`: Name of the topic in which the landmark map is published
  Message type: `visualization_msgs::Marker`

File names parameters

- `noisy_filename`: Name of the file to which it has to write the noisy odometry;
- `odometry_filename`: Name of the file to which it has to write the odometry;
- `ekf_filename`: Name of the file to which it has to write the ekf odometry;
- `gt_filename`: Name of the file to which it has to write the ground-truth;
- `map_filename`: Name of the file to which it has to write the map;

Topic names parameters:

- `noisy_topic`: Name of the noisy odometry topic to which it has to write the noisy odometry;
- `ekf_topic`: Name of the EKF-SLAM odometry topic to which it has to write the noisy odometry;
- `odometry_topic`: Name of the odometry topic to which it has to write the odometry;
- `gt_topic`: Name of the ground truth topic to which it has to write the ground-truth;

- `map_topic`: Name of the map topic to which it has to write the map;

A.2.5. tf_odom_publisher

Input topics:

- `ekf_odom`: ekf odometry topic
  Message type: `nav_msgs::Odometry`

Topic names parameters:

- `ekf_topic`: Name of the EKF-SLAM odometry topicy;

List of Figures

[1] https://it.mathworks.com/help/vision/ref/pcread.html

List of Tables

List of Symbols

Variable	Description
μ_t	KF filter state mean at time t
Σ_t	KF filter state covariance at time t
$\mathbf{x}_t$	Robot state vector at time t
θ_t	Orientation at time t
$\mathbf{z}_t$	Measurement vector obtained at time t
G_t	Motion model Jacobian at time t
H_t	Sensor model Jacobian at time t
K_t	Kalman filter gain at time t
R_t	Process noise covariance matrix at time t
Q_t	Sensor noise covariance matrix at time t
x_R	Robot x coordinate in map frame
y_R	Robot y coordinate in map frame
θ_R	Robot θ coordinate in map frame
x_s	Landmark x coordinate in sensor frame
y_s	Landmark y coordinate in sensor frame
x_m	Landmark x coordinate in map frame
y_m	Landmark y coordinate in map frame
l_m	Landmark m

Acknowledgements

I would like to thank the professor Matteucci, which gave me the opportunity to participate in this work, my two co-advisors, Paolo Cudrano and Simone Mentasti, for helping me in development of this thesis and doctor Matteo Frosi for guiding me toward the right direction for the plane recognition algorithm. Also, I would like to say thank you to my colleagues: Roberto Basla, Davide Biarese and Marco Balossini, for accompanying me during this journey. I thank my friends Giada and Eleonora and Giorgia, for being present when I needed it the most. I thank my family and in particular my mom and dad, which supported me even in difficult periods, trying to help me as much as they could possibly do. Last but not least, I would like to thank professors Fuligni and Ubertini from my high school for inspiring me to study and continuously research in the IT field.