POLITECNICO
MILANO 1863

School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering

Master of Science Degree in Computer Science and Engineering

3D Reconstruction and Segmentation for Vineyard Plant Phenotyping

Supervisor: Prof. Matteo Matteucci
Co–Supervisor: Eng. Andrea Romanoni
Co–Supervisor: Eng. Giulio Fontana

Master Thesis by:
Riccardo Bertoglio, matricola 875635

Academic Year 2018-2019

To the mankind.

Abstract

Plant phenotyping is the measurement of the observable plant characteristics that result from the interaction of genotype and environment, such as stalk width, leaf area, leaf angle, and color. Plant phenotyping information is relevant for farmers to support their decision-making process. Thus, they can distribute a precise fine-tuned quantity of inputs (such as water, fertilizers, pesticides) and enhance crop performance. Nowadays, plant phenotyping is performed manually by skilled scientists or breeders; this procedure is laborious, expensive, and time-consuming. Automatic plant phenotyping systems aim at overcoming the limitations of current manual methods allowing rapid, non-destructive, accurate, and high-throughput measurements. One of the major flaws in the current methodologies is the incapability to collect, reliably, and in situ, large-scale phenotyping measurements. In addition to the management of a huge quantity of phenotyping data, new modeling techniques, advanced analysis tools, and prediction models are necessary to turn automatic phenotyping into a real-world reality.

In this thesis, we developed a system able to scan entire crop rows and to automatically recognize single plants. We designed a structure equipped with a set of sensors that can be mounted on different platforms like an autonomous robot or a tractor. The system is able to localize itself thanks to a sensors fusion approach of the GPS and cameras information. The localization could work even in the total absence of the GPS signal. Moreover, we designed a reconstruction software component to retrieve the tridimensional model of a crop row. The reconstruction process exploits the localization information from the sensor fusion approach. Laser scans and camera images are assembled to form tridimensional models. Finally, we devised a segmentation algorithm to isolate single plants from the produced reconstruction.

Sommario

La fenotipizzazione delle piante è la misurazione delle caratteristiche osservabili che risultano dall'interazione del genotipo con l'ambiente, quali la larghezza degli steli, l'area e l'angolo delle foglie, e il colore. L'informazione del fenotipo è utile come supporto al processo decisionale degli agricoltori. Infatti, grazie a queste informazioni, possono spargere una precisa e calibrata quantità di risorse (come acqua, fertilizzanti, pesticidi) e aumentare così le prestazioni del raccolto. Oggigiorno, la fenotipizzazione delle piante è eseguita a mano da coltivatori o scienziati esperti; la procedura è molto laboriosa, costosa, e richiede molto tempo. I sistemi automatici di fenotipizzazione mirano a superare le limitazioni degli attuali metodi manuali permettendo di effettuare misure in maniera rapida, non distruttiva, accurata, e con un alto volume di produzione. Uno dei maggiori difetti dei metodi attuali è l'incapacità di effettuare misurazioni affidabili di coltivazioni su larga scala. Oltre alla gestione di una enorme quantità di dati, è necessario sviluppare nuove tecniche di modellazione, nuovi avanzati strumenti di analisi, e nuovi modelli di predizione per trasformare la fenotipizzazione automatica in realtà.

In questa tesi abbiamo sviluppato un sistema capace di scansionare intere file di colture e di riconoscere automaticamente le singole piante che ne fanno parte. Abbiamo progettato una struttura con un insieme di sensori che può essere montato su diverse piattaforme, come un robot autonomo o un trattore. Il sistema che abbiamo sviluppato è in grado di localizzarsi grazie alla fusione delle informazioni provenienti da sensori quali il GPS e le fotocamere. La localizzazione può anche funzionare nella totale assenza del segnale GPS. Inoltre, abbiamo progettato un componente software in grado di restituire un modello tridimensionale di un filare di una coltura. Il processo di ricostruzione sfrutta le informazioni di localizzazione dall'approccio di fusione precedentemente descritto. Le scansioni provenienti da un sensore laser e le immagini delle fotocamere sono assemblate per formare un modello tridimensionale. Infine, abbiamo sviluppato un algoritmo per isolare le singole piante dalla ricostruzione precedentemente ottenuta.

Ringraziamenti

Inizio ringraziando la mia famiglia, i genitori e i nonni, per avermi sempre sostenuto e assecondato nelle mie scelte. Li ringrazio per avermi dato il sostentamento economico affinché potessi portare a compimento gli studi. Inoltre, mi hanno sempre messo nella condizione di vivere una vita agiata, con una libertà economica che mi ha permesso di dar sfogo alla mia inesauribile curiosità. Questa libertà si è materializzata in molteplici passioni e interessi che spaziano tra gli ambiti scientifico, culinario, artistico e sportivo. Un grazie di cuore.

Ringrazio gli amici più stretti. In particolare Brian, Gianbattista e Giulia, per essermi stati sempre vicini, nei momenti felici e in quelli più difficili, per avermi accettato per quello che sono, con sincerità. Un valore che ritengo essenziale in qualsiasi rapporto interpersonale. Ringrazio gli amici del paese con cui ho condiviso tanti bei momenti a partire dalla mia infanzia fino a oggi. Ringrazio gli amici dell'università e, in particolare, Emiliano che si è dimostrato una persona di grande umanità e sensibilità. Infine, un vivo ringraziamento va a i miei ex coinquilini, Andrea e Filippo (da me soprannominato Cilippolo), con cui ho avuto il piacere di stringere un solido rapporto di amicizia durante la convivenza. Grazie per aver assecondato le mie stranezze e bizzarrie.

Ringrazio il mio relatore Matteo per avermi condotto nel percorso di tesi. È un viaggio tortuoso in cui è essenziale essere affiancati da una guida esperta che tracci la strada da seguire. Esprimo inoltre la mia gratitudine per aver creduto in me e per avermi avvicinato alla realizzazione del mio sogno di entrare al dottorato. Comunque andrà, grazie per il supporto. Ringrazio Giulio per essersi sempre mostrato disponibile quando avevo bisogno di aiuto. Grazie anche alla sua notevole esperienza, ha sempre saputo darmi consigli utili e indicazioni precise. Ringrazio per ultimo, ma non per importanza, Andrea, per essere stato altresì una figura importante per lo sviluppo del mio lavoro di tesi. Si è rivelato una preziosa fonte di luce nei momenti bui.

Ringrazio il Politecnico di Milano per avermi dato solide competenze ed un'eccellente istruzione che mi permettono di essere competitivo nel mondo del lavoro. Parallelamente a questo, è stato di immenso aiuto nella crescita personale, dandomi gli strumenti per sviluppare il pensiero critico, tanto importante quanto le conoscenze ingegneristiche. Il pensiero critico ritengo sia una competenza di vitale importanza nella quotidianità attuale, in cui siamo subissati di informazioni che, potenzialmente, possono essere false e avere conseguenze dannose per la singola persona e per la società intera. Inoltre, grazie alle associazioni, agli eventi, alle conferenze e ai corsi extracurricolari, mi ha permesso di lanciare lo sguardo oltre i confini dello studio. Mi ha dato spunti, nuove idee e la possibilità di svagarmi tra un esame e l'altro. Infine, sono felice che questa università veda la diversità come una ricchezza, sostenendo cause che vanno dai diritti civili, per esempio concedendo

il patrocinio al Gay Pride di Milano, fino al processo di internazionalizzazione, attraverso l'insegnamento in lingua inglese. Si tratta di un valore aggiunto che mi fa affermare con convinzione di essere orgoglioso di aver studiato al Politecnico di Milano.

Se è vero che una persona è sagomata dalle persone di cui si circonda e dall'ambiente in cui vive, allora devo ringraziare anche la città di Milano. C'è chi dice che Milano sia una città grigia e i suoi abitanti siano persone fredde. Non è così. Milano è una città che mostra le sue bellezze solo a chi le sa cercare. È la città delle persone curiose, delle persone di mentalità aperta, delle persone che abbracciano le novità e i cambiamenti, delle persone intraprendenti. È una città su misura per i giovani, dinamica e ricca di sfide. È una città che dà grandi opportunità a chi le sa cogliere. Con i suoi musei, ristoranti di ogni nazionalità, persone da ogni parte del mondo, le community di appassionati, gli eventi di divulgazione scientifica, le mostre di design, le installazioni di arte moderna, i suoi parchi, i teatri, è una città che ti permette di sognare in grande. Milano non è grigia, è color arcobaleno.

In questo sommario di vita non posso non ricordare Gianluca che, nel bene e nel male, ha lasciato un segno indelebile nei miei ricordi. È addirittura riuscito a condizionarmi, in maniera decisiva, nella scelta di importanti progetti riguardanti la mia futura carriera lavorativa. A questo proposito, voglio ricordare ed esprimere la mia gratitudine nei confronti dei suoi amici, Stefano e Matteo.

La famiglia, gli amici, i professori, l'università, la città, in questi anni mi hanno plasmato e hanno dato forma alla persona che sono ora.

Grazie a tutti.

Riccardo

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The phenotyping task consists of measuring the observable plant characteristics that result from the interaction of genotype and environment. Researchers want to assess characteristics such as stalk width, leaf area, leaf angle, and color. Further, plant phenotyping information is relevant for farmers to support the decision-making process. Thus, they can distribute a precise fine-tuned quantity of inputs (such as water, fertilizers, pesticides) and enhance crop performance. However, nowadays, plant phenotyping is performed manually by skilled scientists or breeders; this procedure is laborious, expensive, and time-consuming.

This work aims at giving a contribution to the field of automatic plant phenotyping, a branch of the broader Precision Agriculture research field. Automatic plant phenotyping systems aim at overcoming the limitations of current manual methods, by measuring the morphometric and physiological parameters of plants in a rapid, non-destructive, accurate, and high-throughput manner. Even if considerable work has been done, we are still at the beginning of the journey toward a robust and standard phenotyping methodology [1]. One of the major flaws in the current methodologies is the incapability to collect, reliably, and in situ, large-scale phenotyping measurements. In addition to the management of huge quantity of phenotyping data, new modeling techniques, advanced analysis tools, and prediction models are necessary to turn automatic phenotyping into a real-world reality [2].

With these concepts in mind, we propose the new paradigm of *Plant Digital Twin*, that is, the computerized counterpart of a real plant. The idea is to build a tridimensional model of an entire crop and distinguish single plants. This framework enables growth analysis of individual plants and localized treatments. This thesis lays the foundations for this new kind of paradigm by proposing a system able to scan entire crop rows and to recognize single plants.

We designed a platform equipped with a set of sensors that can be mounted on different vehicles like an autonomous robot or a tractor. We chose the following minimal set of sensors able to provide a good trade-off among amount of data, cost, power-consumption and richness of information: a 2D LiDAR, two RGB-D cameras, a GPS, an Inertial Measurement Unit (IMU). The system is able to localize itself by fusing the GPS, the IMU and the cameras data. The localization approach could work even in the total absence of

GPS information, thus relying only on camera images with a Visual Odometry algorithm.

Moreover, we designed a reconstruction software component that has the purpose of retrieving the tridimensional model of a scanned subject. The reconstruction process exploits the localization information from the sensor fusion approach. LiDAR laser scans are merged into a single point cloud one after the other knowing the relative transformation of the sensor origin to the base position of the robot. The cameras color and depth images are registered one to the other into a final point cloud by a graph-based Simultaneous Localization And Mapping (SLAM) approach.

Finally, we devised a segmentation algorithm to isolate single plants from the tridimensional reconstruction. The algorithm segments plants by purely geometric considerations exploiting clustering algorithms. We tested our method on data collected at a botanical garden in Milan. We focused our attention on the reconstruction and segmentation of a double-sided vineyard row. By visual inspection, we can state that the reconstructions accurately represent the scanned subjects in terms of shape, colors, and scale. Moreover, the segmentation approach correctly segmented most of the plants.

The thesis is structured as in the following:

- Chapter 2 provides a review of the literature proposing a new classification taxonomy. Moreover, it explains the research motivations in the field of automatic plant phenotyping and highlights the lacks of the state-of-the-art methods
- Chapter 3 gives the necessary background notions to non-expert readers about all the technologies and the theoretical concepts exploited in this work
- Chapter 4 explains the hardware architecture of the developed phenotyping platform. Moreover, it provides an evaluation and comparison of different sensors and technologies for 3D imaging
- Chapter 5 describes the software architecture explaining the three main components, that is, Data Collection, Reconstruction, and Segmentation
- Chapter 6 shows experimental results obtained from outdoor tests performed at a botanical garden. They are discussed and compared various odometry approaches and the segmentation algorithm
- Chapter 7 makes the conclusions and state some short-term work as a consequence of this work. Moreover, it provides the reader with a long-term vision about future innovations in the field of automatic plant phenotyping
- Appendix A collects tables with the technical details of the tested sensors

Chapter 2

State Of The Art

In this Chapter, we present the state of the art of automatic plant phenotyping. In Section 2.1, we explain the motivations that push us to research on this topic. In Section 2.2, we propose a new taxonomy categorizing all the possible approaches seen in the literature. In Section 2.3, we review a selection of studies explaining papers inclusion and exclusion criteria. Finally, in Section 2.4, we illustrate and discuss the main lacks of the approaches proposed in the literature.

2.1 Motivations

The Food and Agriculture Organization (FAO) of the United Nations stated that, by 2050, world food demand would be 70% higher. To satisfy the increase of food demand, "ninety percent of the growth in crop production globally [...] is expected to come from higher yields and increased cropping intensity" [3]. To reach this goal, breeders and biologists, demand phenotyping data to improve crop performance. Biologists need to develop new high-yielding genotypes of crops adapted to our future climate [1]. In doing this, they necessitate an experimental confirmation by measuring the observable plant characteristics that result from the interaction of genotype and environment. This measurement process is called plant phenotyping. Moreover, farmers can exploit phenotyping information as the input of a Decision Support System (DSS) [4] adopting technologies like variable-rate spraying of water, nutrients, and pesticides.

Nowadays, plant phenotyping has to be performed manually by skilled scientists or breeders; this procedure is laborious, expensive, and time-consuming. Thus, research in the field of automatic plant phenotyping systems has begun to go beyond the limitations of current manual methods. Automatic plant phenotyping should allow the measurement of the morphometric and physiological parameters [5] of plants in a rapid, non-destructive, accurate, and high-throughput manner. Morphometric parameters are like plant height, stem diameter, leaf area, leaf angle, stalk length, and in-plant space. Physiological parameters are like chlorophyll, photosynthetic rate, water stress, biomass, salt resistance, and leaf water content [6].

Even if considerable work has been done in the field of plant phenotyping, we are still at the beginning of the journey toward a phenotyping methodology that can keep

pace with the demand. This deficiency has been vastly highlighted in the literature under the name of Phenotyping Bottleneck [1]. Since [1], the entirety of studies have cited and discussed the Phenotyping Bottleneck. This problem has been recognized as caused by the lack of systems that can collect data in situ on a large scale. Moreover, we need to find a way to manage the huge quantity of phenotyping data that will be produced. New modeling techniques and data management approaches should be investigated. Finally, new advanced analysis tools and prediction models, beyond even the usual statistical tools, are necessary [2], [6].

2.2 Taxonomy

The literature in the field of automatic plant phenotyping is vast and heterogeneous. Current systems exploit a variety of sensors and platforms, giving birth to a broad taxonomy. However, it does not exists a complete and standard classification of the various approaches. Therefore, we propose a new one. We explain the rationale behind our taxonomy with the hope that it will be widely adopted by the scientific community.

Plant phenotyping studies can be classified along different orthogonal dimensions. Before the taxonomy, we describe the general pipeline common to most phenotyping systems, as each step of the process has its independent taxonomy. Some works addressed all the stages of the pipeline, while others investigated just some of them.

2.2.1 Phenotyping Pipeline

The initial step of the pipeline (see Figure 2.1) is **Data Collection**, where the system collects data by its sensors. Some of the most relevant tasks of this phase are the choices of sensors and their positioning [5]; supporting platform; localization and navigation algorithms; data storage (hardware and software) methodologies.

The successive step of the pipeline is **Model Construction**. In this phase, the raw collected sensors data are processed to produce a model that can be successively adopted in the next steps. This stage is necessary whenever the data coming directly from the sensors are meaningless as they are. For example, data coming from 2D lidars mounted in push-broom fashion (laser plane is perpendicular to the ground) do not give any relevant information if they are not assembled exploiting sensor poses information. Indeed, without knowing the positions of the sensors over time, laser scans can just be assembled piling them up one onto the other. Thus, the obtained data will not provide any useful information. Typical technologies used in this stage are algorithms from the fields of Computer Vision, 3D Reconstruction, Robotics, and Machine Learning. A model could be, for example, a point cloud representing the scanned plants.

The next step of the pipeline is **Data Processing**. In this phase, the information coming from the previous stage is processed to calculate phenotyping information like architectural traits, physiological traits, and diseases or pests presence. This stage exploits Data Mining and Machine learning algorithms, Statistics, and Agricultural models [7]. For example, a Data Processing task is to compute the height of plants or the leaves angles from the plants point cloud model.

The last step of the pipeline is called **Data Analysis**. In this phase, the user of the

phenotyping system exploits the information provided by the previous stage. Biologists can use this information as feedback to their experimental activities while breeders can make a crop breeding selection. If the phenotyping information is exploited for successive plantations, the phenotyping pipeline becomes a phenotyping cycle.

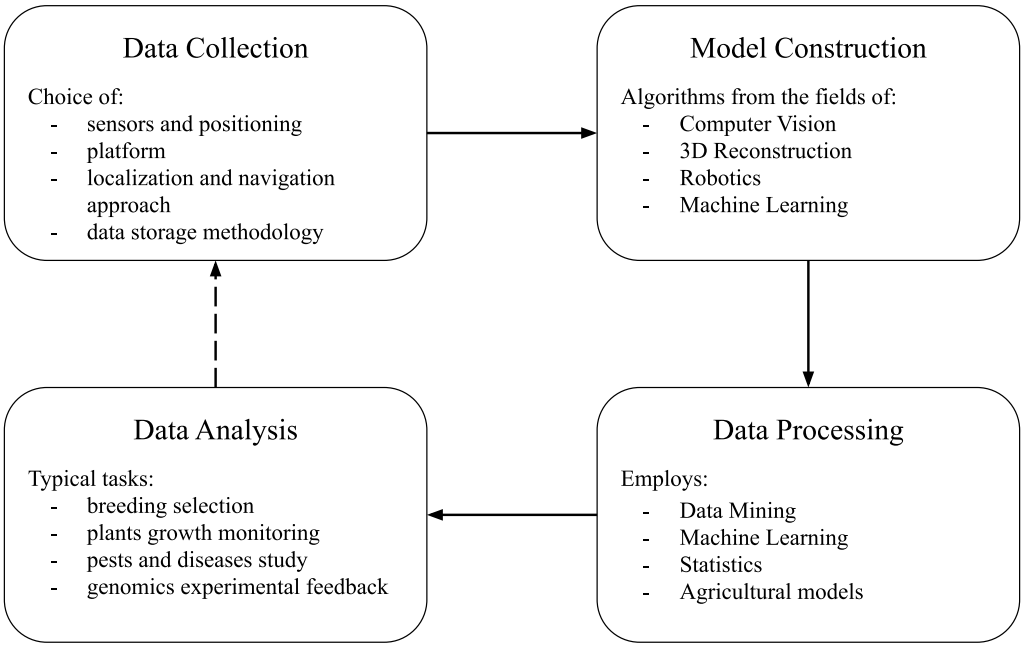


Figure 2.1: Graphical representation of the phenotyping pipeline/cycle.

In the following, we propose a new classification of the literature. We first explain the richest taxonomy related to the Data Collection phase, then the one related to the Data Processing phase. The other stages do not produce any relevant classification to be presented here.

2.2.2 Data Collection classification

Schematic representation of the classification.

- **Operating Environment**
  - *outdoor*
  - *indoor*
- **Motion State**
  - *static*
  - *moving*
    - **Autonomy Level**
      - *autonomous*
      - *semi-autonomous*
      - *human-operated*

- **Ground Contact**
  - *ground*
  - *aerial*
- **Platform**
  - *Environmental Sensors Networks*
  - *Unmanned Ground Vehicles (UGVs)*
  - *Hand-pulled Systems*
  - *Tractor-based Systems*
  - *Phenotyping Towers*
  - *Gantry-based Systems*
  - *Unmanned Aerial Vehicles (UAVs)*
  - *Blimps & Balloons*
  - *Satellite Imaging*
- **Sensor Technology**
  - *RGB Cameras (monocular or stereo)*
  - *RGB-D Sensors*
  - *TOF Sensors*
  - *Structured-light Sensors*
  - *2D/3D LiDARs*
  - *3D Laser Scanners*
  - *Light Curtains (LC)*
  - *Thermal Cameras*
  - *Fluorescence Cameras*
  - *Spectral Cameras*
  - *Normalized Difference Vegetation Index (NDVI) Sensors*
  - *Other Sensors*

The first aspect of the Data Collection classification is the **Operating Environment**. Approaches can be divided as operating *outdoor* or *indoor*. Outdoor environments are crops fields, while indoor environments are laboratories and greenhouses. Here we only consider outdoor phenotyping systems as indoor experiments cannot generalize well for outdoor natural environments [8]. Outdoor phenotyping allows scientists to perform large scale studies and to measure variables difficult to test in indoor settings, like pest and disease tolerance and the effects of varying soil conditions.

Another classification dimension is the system **Motion State**, that is, whether the system is *static* or is *moving*. A system is static or moving, whether it is still or not during the acquisition process.

A *moving* system can be further classified based on its **Autonomy Level**. Indeed, the system could be *autonomous*, *semi-autonomous*, or *human-operated*. The idea of this classification is to evaluate the effort needed to realize the navigation approach. The autonomous systems require the most effort to be realized, and the human-operated ones, almost no effort. The effort is intuitively described by the time the navigation algorithm needs to be implemented, by the complexity of the task, and by the complexity of the adopted technologies and algorithms. An autonomous system can perform a complete acquisition of the designated area with no human intervention along the process. A semi-autonomous system requires repeated human intervention during the data collection process. The system can autonomously move until it reaches a point where it waits for human intervention. After human action, the system autonomously reaches the next waiting point and so on till the end of the operation. A human-operated system requires continuous human intervention during the acquisition process.

The next dimension is the one that classifies the approaches based on the **Ground Contact**, that is, *ground* or *aerial* systems. Ground systems are whatever platform that is sustained in some way by the ground while aerial systems are platforms sustained only by the air, continuously or just during operations.

Another possible classification dimension is the **Platform**, that is, the kind of structure or machine on which the sensors are attached. The platform gives to the sensors the necessary connectivity, the power, support onto which fix them, all the hardware and software needed for data collection, and an adequate motion if it is required. The platforms are: *Environmental Sensors Networks*, *Unmanned Ground Vehicles (UGVs)*, *Hand-pulled Systems*, *Tractor-based Systems*, *Phenotyping Towers*, *Gantry-based Systems*, *Unmanned Aerial Vehicles (UAVs)*, *Blimps & Balloons*, *Satellite Imaging*.

The last classification dimension is the **Sensor Technology** used for phenotyping purposes. Sensors could be of different kinds: *RGB Cameras (monocular or stereo)*, *RGB-D Sensors*, *TOF Sensors*, *Structured-light Sensors*, *2D/3D LiDARs*, *3D Laser Scanners*, *Light Curtains (LC)*, *Thermal Cameras*, *Fluorescence Cameras*, *Spectral Cameras*, *Normalized Difference Vegetation Index (NDVI) Sensors*. Other sensors: *Laser Distance Sensor (LDS)*, *Ultrasonic Sensors*, *Electromagnetic Induction (EMI)*, *Ground Penetrating Radar (GPR)*, *Electrical Resistance Tomography (ERT)*, *Thermometers*, *Humidity Sensors*.

A strong connection is present within the presented classification. Indeed, each kind of platform can be classified along the Operating Environment, Motion State, and Ground Contact dimensions. In Table 2.1, we classify each platform along the cited dimensions. For moving systems, if all autonomy levels are possible, we indicate them just as "moving". Otherwise, the level of autonomy is specified. Satellite Imaging has been classified as autonomous moving systems. Although the effort required to make a satellite autonomous is very high, users buy this service from providers. In this case, the effort is more economical than human work.

2.2.3 Data Processing classification

Data Processing classification has only one relevant dimension, the **Model Scope**, that is, the ability to build a model at the level of each *single plant* or of *groups of plants*. Having

Platform	Operating Environment	Motion State	Ground Contact
Environmental Sensors Networks	indoor & outdoor	static	ground
Unmanned Ground Vehicles	indoor & outdoor	moving	ground
Hand-pulled Systems	indoor & outdoor	human-operated	ground
Tractor-based Systems	indoor & outdoor	human-operated	ground
Phenotyping Towers	indoor & outdoor	static	ground
Gantry-based Systems	indoor & outdoor	moving	ground
Unmanned Aerial Vehicles	outdoor	moving	aerial
Blimps & Balloons	outdoor	human-operated	aerial
Satellite Imaging	outdoor	autonomous	aerial

Table 2.1: Phenotyping platforms classification. For moving systems, if all autonomy levels are possible, we indicate them just as "moving". Otherwise, the level of autonomy is specified.

data at the level of each single plants means that the model can isolate single plants and can associate to them a unique identifier so to track them over time. Data processing strategies that retrieve data related to groups of plants can not distinguish every single plant.

2.3 Literature Review

In this Section, we present a review of the literature of automatic plant phenotyping. In this introduction, we explain papers inclusion and exclusion criteria.

We only include paper related to *outdoor* systems as indoor experiments can not generalize well for outdoor settings, and there is a lack of outdoor systems due to the more challenging environment in which they are operating [8].

Moreover, we only include studies of *moving* systems as vehicles, by approaching closer

to the scanned subjects, can retrieve data with a finer resolution. Indeed, static systems like Phenotyping Towers collect data with larger distance from plants.

For the same reason, we also exclude *aerial* systems as they have a reduced spatial resolution compared to *ground* systems. Nonetheless, a promising class of aerial platforms is that of Unmanned Aerial Vehicles (UAVs). A comparison [9] between UAVs and Unmanned Ground Vehicles (UGVs) demonstrated a similar accuracy on plants height measurements obtained both from aerial reconstructions and ground LiDAR measurements. However, the low load capacity and power autonomy of UAVs, and the strict airspace regulations are an obstacle to UAV phenotyping. Still, future improvements of current technologies could bring UAVs performance comparable to the one of UGVs. See [10] for a review of UAV systems for phenotyping applications.

We do not use the *autonomy level*, the *sensor technology* or the *model scope* as criteria to select studies because all categories of systems of these classifications are relevant for our purposes.

In summary, we only selected studies of outdoor ground moving systems with no other constraints. For ease of comparison, the review is presented through subsections that collect studies with the same type of platform. The kinds of platforms that respect the inclusion criteria explained above are Unmanned Ground Vehicles, tractor-based systems, gantry-based systems, and hand-pulled systems.

In Table 2.2, we list all the works that have been investigated by us to study the automatic plant phenotyping field. Then, in the following, we present the most relevant studies for each platform category.

Authors, Year, Reference	Platform	Phenotyping Sensors	Phenotyping Parameters
Qiu et al., 2019, [11]	Unmanned Ground Vehicle	3D LiDAR (64 planes)	Row spacing, Plant height
Shafiekhani et al., 2017, [12]	Unmanned Ground Vehicle	RGB stereo camera, Temperature sensor, Humidity sensor, Light intensity sensor	Plant height, LAI, Environmental data
Mueller-Sim et al., 2017, [8]	Unmanned Ground Vehicle	2D LiDAR, Custom stereo camera, RGB camera	None
Kicherer et al., 2015, [13]	Unmanned Ground Vehicle	Monochrome camera, RGB camera, NIR camera	Vine berry size and color
Herzog et al., 2014, [14]	Unmanned Ground Vehicle	RGB camera, Monochrome camera	Vine bud burst detection, Vine berry size
Barker et al., 2016, [15]	Tractor-based System	Spectral reflectance sensors (GreenSeeker and Crop Circle), Ultrasonic sensor, IR sensor, Laser distance sensor	NDVI, Environmental data
Deery et al., 2014, [16]	Tractor-based System	2D LiDAR, RGB camera, Hyperspectral sensor, Thermal camera	Plant height, Biomass (through Leaf area and volume), NDVI, PRI
Andrade-Sanchez et al., 2014, [17]	Tractor-based System	Sonar proximity sensor, IR sensor, Spectral reflectance sensor (Crop Circle)	Plant height, Plant temperature, NDVI
Sanz et al., 2013, [18]	Tractor-based System	2D LiDAR	Plant volume, Leaf Area, LAD

Busemeyer et al., 2013, [19]	Tractor-based System	Time-of-Flight (ToF) camera, RGB camera, Laser distance sensors (LDS), Hyperspectral camera, Light curtain	Plant height
Sui et al., 2013, [20]	Tractor-based System	Ultrasonic sensor	Plant height
Comar et al., 2012, [21]	Tractor-based System	Hyperspectral camera, RGB camera, Solar radiation sensor	Green fraction, Canopy adjusted ratio index 2 (MCARI2), MERIS terrestrial chlorophyll index (MTCI)
Llorens et al., 2011, [22]	Tractor-based System	2D LiDAR	Canopy density, LAI
Virlet et al., 2017, [7]	Gantry-based systems	RGB camera, 3D Laser scanner, Thermal camera, Hyperspectral camera, NDVI sensor, Fluorescence camera	Canopy cover, Plant height, NDVI, Chlorophyll
Vadez et al., 2015, [23]	Gantry-based systems	3D imaging scanner (PlantEye F300 from Phenospex), Analytical scales, Relative humidity sensor, Temperature sensor, Light sensor, Wind sensor, Rain gauge	Leaf area, Plant transpiration
White et al., 2013, [24]	Hand-pulled System	Monochrome camera, Ultrasonic sensor, IR thermometer, Radiometers	None

Table 2.2: Automatic Plant Phenotyping studies. The "Phenotyping Sensors" column lists the sensors used only for phenotyping purposes. Sensors for localization or other purposes are not listed. The "Phenotyping Parameters" column lists the phenotyping parameters that have effectively been estimated by experimental activities. Acronyms: NDVI = Normalized Difference Vegetation Index; PRI = Photochemical Reflectance Index; LAI = Leaf Area Index; LAD = Leaf Area Density; NIR = Near Infrared Radiation.

2.3.1 Unmanned Ground Vehicles

Unmanned Ground Vehicles (UGV) are platforms that wander the crops to collect data. They are called unmanned because they operate without an onboard human presence. They can carry a heavy payload of sensors, but they can cause soil compaction. Indeed, disturbance of soil structure through compaction can weaken plants health [25]. UGVs can be both autonomous, semi-autonomous, or even human-(tele)operated. Autonomous UGVs are a very promising category of platforms as they can collect data regularly and frequently.

In [11], the authors developed a field-based high-throughput phenotyping solution for maize, using a 3D LiDAR placed on a mobile robot platform (Figure 2.2). They proposed a solution to measure row spacing and single plants height from the LiDAR reconstruction. Indeed, they designed a single plant detection algorithm based on points density (Figure 2.3). However, they did not provide the confusion matrix of the segmentation algorithm to demonstrate its accuracy. Moreover, their single plant detection approach is not general enough to be adapted to other crop species, like the vineyard. Furthermore, they used a very expensive 3D 64 planes Velodyne sensor, thus hindering the spreading of such a system. Finally, they conducted experimental tests moving the robot on a cement floor. However, these controllable floor conditions cannot be assumed for real farming crops.

Figure 2.2: Qiu et al. developed UGV with the Velodyne HDL64E-S3 LiDAR sensor. (Image source [11])

In [12], the authors developed a semi-autonomous ground vehicle called Vinobot (Figure 2.4) capable of scanning crop rows at the level of every single plant. They also made a phenotyping tower providing data of groups of plants. Throughout the vehicle, they performed 3D reconstructions of single plants with a stereo camera. Moreover, they collected environment data like temperature, humidity, and light intensity. The robot is semi-autonomous as it requires human intervention to align it with a row, then is able to scan the entire row autonomously. However, the single plant recognition has been done by RFID tags placed on every single plant, thus requiring extensive human labor. Moreover, the scanning process needs the robot to stop at every single plant and move the camera

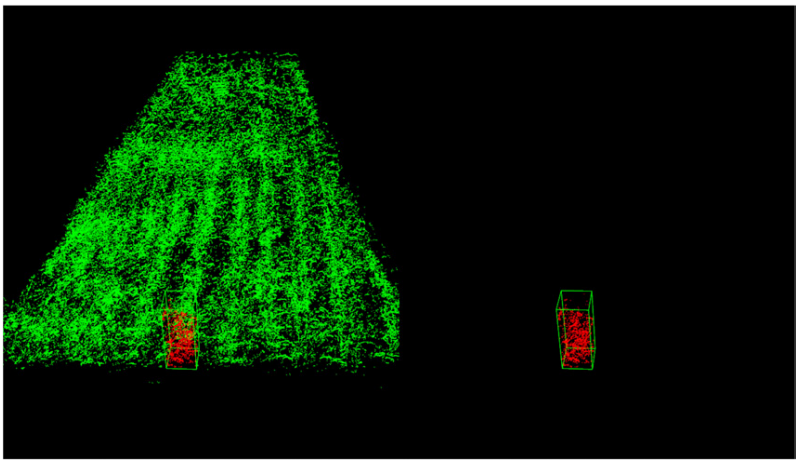


Figure 2.3: Qiu et al. single plant detection. The detected single plant cloud is in red and marked out with a green bounding box. (Image source [11])

to pre-defined positions. This scanning approach is a significant limitation that hinders high-throughput phenotyping.

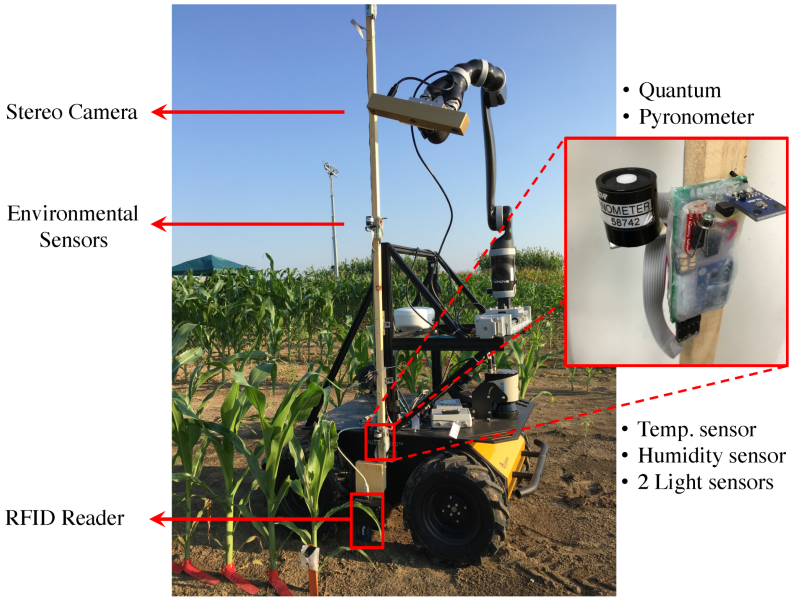


Figure 2.4: Hardware components of Vinobot. (Image source [12])

In [8], the authors developed an autonomous ground vehicle called The Robotanist (Figure 2.6) capable of collecting phenotyping data with passive and contact-based sensors. Employing a 2D LiDAR in push-broom configuration, they performed 3D reconstructions of an entire field. However, they did not provide any quantitative evaluation of phenotyping parameters, and they did not perform single plant segmentation. Moreover, the navigation approach only relies on an RTK-GPS (plus an AHRS sensor) that is not a reliable source of information. The authors stated: "[...] as the season progresses and the sorghum grows taller than the GPS antenna, that capability will be lost.". Moreover, during field validation, the robot has been primarily teleoperated.

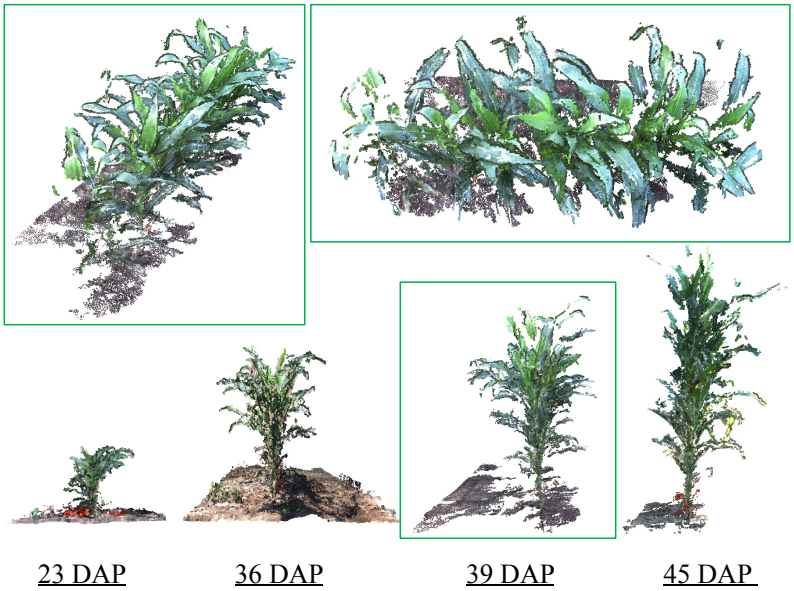


Figure 2.5: Typical examples of 3D reconstructed plants at four different DAP (days after planting) using stereo images collected by the Vinobot. (Image source [12])

Figure 2.6: The Robotanist in sorghum breeding plots. (Image source [8])

2.3.2 Tractor-based Systems

Tractor-based systems are ground human-operated systems. Agricultural tractors can be equipped with a variety of sensors as they can carry a heavy payload. As tractors weight more than UGVs, they make the soil compaction problem more severe. Moreover, they have the disadvantage of requiring a human presence on board. Since farmers drive their tractors through the field for crop works, it is still interesting to mount sensors on them to collect data while they are performing other operations in the field. The disadvantage is to have sporadic data over time.

In [15], the authors equipped a tractor (Figure 2.8) with phenotyping sensors, and

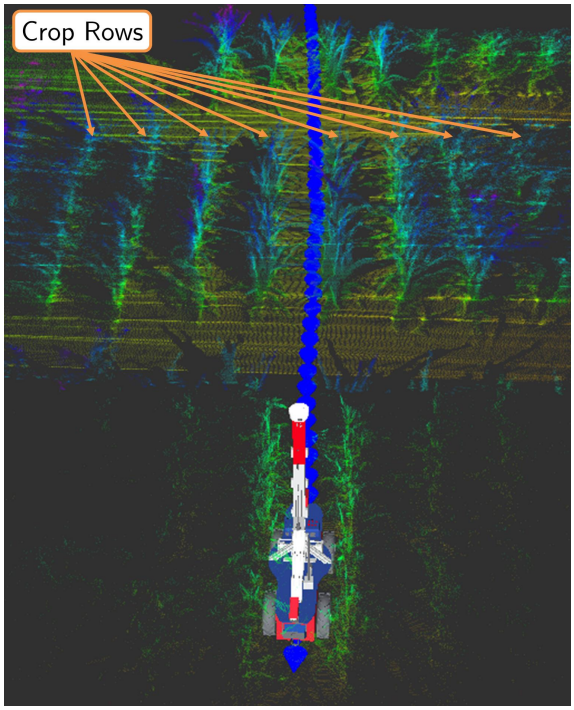


Figure 2.7: Data from planar laser scanners in push-broom configurations collected in a Sorghum bicolor breeding plot. (Image source [8])

they measured parameters like canopy temperature, crop height, and canopy spectral reflectance. They reached a maximum speed of 0.89 m/s with a sensors sampling rate of 10 Hz. They mainly investigated how ambient light and temperature conditions affect sensor measures. They found a statistically significant effect of ambient light intensity and temperature on readings from the GreenSeekers, the Crop Circles, the ultrasonic sensor, the laser distance sensor, and the IRT. Especially for this latter sensor they recommended a correction method using ground truth measurement.

Figure 2.8: The tractor-based system developed by Barker et al. (Image source [15])

In [16], the authors proposed a review to evaluate the role of proximal remote sensing

buggies for field-based phenotyping. These systems are called "proximal" because the distance from the instruments to the crop surface is much shorter than in aerial or satellite remote sensing [24]. The word "buggy" stands for "moving vehicle". In particular, they analyzed the advantages and disadvantages of each type of platform and sensor technology.

Moreover, they investigated a tractor-based system (Figure 2.9) that can traverse ~ 1.8 m width plots at a typical operating speed of 1 m/s. They did not state how they did localization, but it is reasonable to think that they fused the RTK-GPS (~ 2 cm resolution) information with the one from wheel encoders (~ 1 mm resolution). However, especially in natural environments, wheel odometry could give poor accuracy due to wheels slipping on wet terrain. Moreover, ground hollows introduce errors in wheel odometry as it assumes planar movements.

Furthermore, they performed 3D reconstructions both by stereo RGB images and LiDAR sensors (see Figure 2.10). They also showed how the red light of the LiDAR could be used to discriminate between plants and soil based on the laser reflectance. Finally, they proposed to count plants from the higher spikes of the laser reconstruction. However, this method for counting plants can be used just for some specific plants species. Indeed, not all plant species have spikes, like the vine.

Figure 2.9: The Phenomobile tractor-based system investigated by Deery et al. (Image source [16])

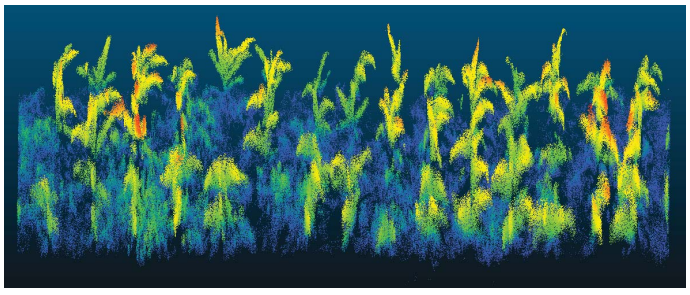


Figure 2.10: Point cloud calculated from the LiDAR. (Image source [16])

In [19], the authors developed a cart called BreedVision equipped with sensors pulled by a tractor (Figure 2.11). The trailer has a track width of 1.25m, and it has been designed for phenotyping of small grain cereals up to a plant height of 1.6m. Moreover, the trailer

has been shaded with a black canvas to avoid exposure to direct solar irradiation. During measurements, the platform was pulled by a tractor with a constant speed of 0.5 m/s.

The authors also proposed hardware and software architectures for data collection and processing. In particular, the data collection system is an industrial PC which incorporates a MySQL database server for data storage. Except for the integration of USB interface sensors, a gigabit ethernet has been chosen as the primary communication bus. Since sensors have different communication interfaces, each one is connected to its microcontroller. As for data processing, they developed a software pipeline that exploits modules reuse for two different tasks, that is, phenotyping traits calibration and traits determination procedures.

Finally, they statistically evaluated the repeatability of the measurements and the accuracy of the plant height parameter. However, due to the trailer structure physical constraints, this system can only be used with specific farming settings and plant species. Indeed, this kind of platform is hardly adaptable for vineyard phenotyping.

Figure 2.11: BreedVision sensor platform during outdoor measurements in the field. (Image source [19])

2.3.3 Gantry-based Systems

Gantry-based systems are ground systems that are usually autonomous or semi-autonomous. These are the systems that can carry the most massive payload of sensors. Moreover, they do not cause soil compaction as these systems need the installation of rails on which the gantry can move. However, these structures are cumbersome and very expensive. Even if they can cover large plots of plants, one need multiple of these systems to cover an entire field, thus resulting in several construction jobs in the field. Moreover, these structures reduce arable land.

In [7], the authors developed a gantry-based system (Figure 2.15) that can support a hefty payload of sensors till 500kg. Systems like this, in principle, can localize themselves with high accuracy. The authors proposed a method to evaluate the positioning precision of the gantry with cameras and barcodes. As for plant phenotyping, they performed different statistical analysis on parameters like canopy coverage, plants height, NDVI index, chlorophyll fluorescence, and ground and maximum fluorescence (Figure 2.13). An additional disadvantage is the required human intervention before each run as the hyperspectral imagers require an initial scan on a standard reflectance panel for exposure calibration.

In [23], the authors developed a gantry-based system that can perform 3D imaging

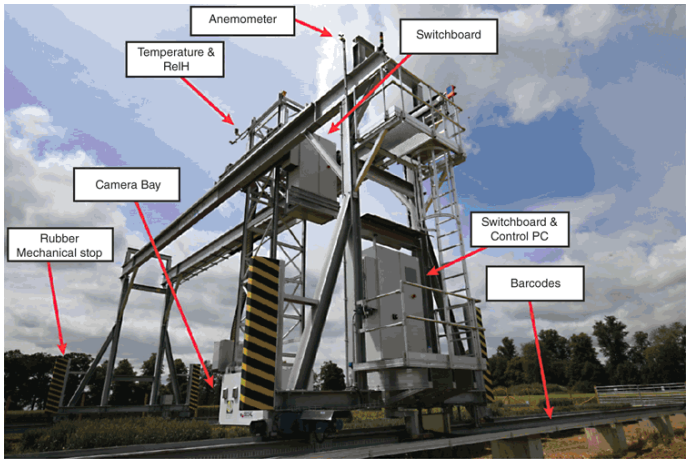


Figure 2.12: The Field Scanalyzer gantry-based system developed by Virlet et al. (Image source [7])

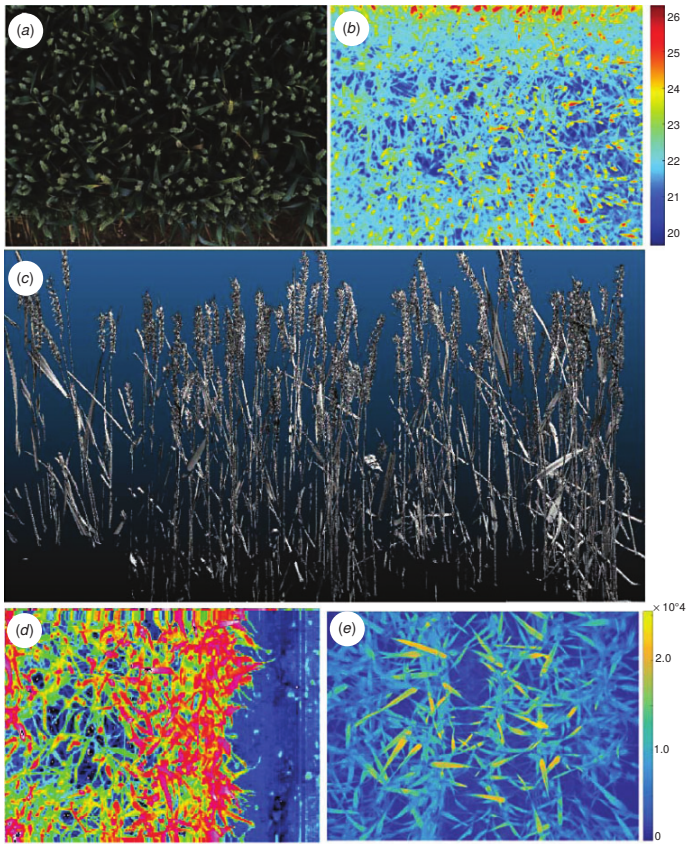


Figure 2.13: (a) RGB image taken from a canopy using the visible camera at 2.5 m above the canopy; (b) thermal infrared image (heat scale in °C) taken at 2 m above canopy; (c) 3D image of wheat canopy take at 3 m above canopy; (d) false colour coded reflectance image at 800 nm taken at 2.5 m above canopy, and (e) false-colour coded fluorescence image (arbitrary units) taken at 0.7 m above canopy. (Image source [7])

combined with lysimetric capacity, to assess canopy traits affecting water use (leaf area, leaf area index, transpiration). They succeeded to develop a high-throughput system that can scan each experimental unit (sector) at least 12 times a day, creating the opportunity of measuring leaf movements and their possible importance for plant water use. Lysimetry

was performed via gravimetric measurement of plant transpiration with analytical scales. They also exploited environmental sensors to monitor relative humidity (RH%) and temperature (T°C), integrating values every 30 min, one light sensor, one wind sensor, and one rain gauge. Moreover, they developed a web-based interface to inspect phenotyping data.

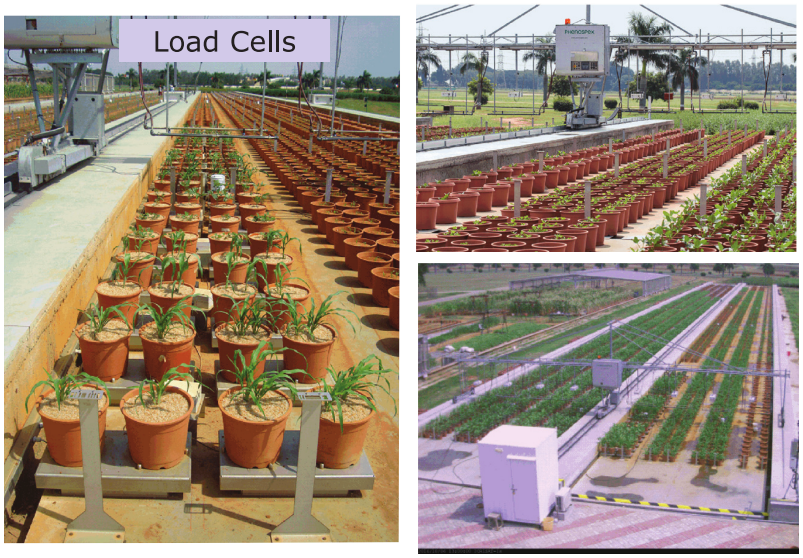


Figure 2.14: The LeasyScan gantry-based system developed by Vadez et al. (Image source [23])

2.3.4 Hand-pulled Systems

Hand-pulled systems are ground human-operated platforms that are lightweight than tractor-based systems, but they can still carry a considerable payload of sensors. Due to the less weight, the soil compaction problem is less severe. Moreover, utilizing less thick tires, they can pass through more dense crops and are less susceptible to damage the crops. However, as they require constant human action, they are not suitable for frequent data collection.

In [24], the authors proposed a cart made with two bicycles frames (Figure 2.16). They mounted two monochrome cameras and three infrared thermometers. With two instrument support arms, their systems weighed 40kg. During tests, they reached an average speed of approximately 0.36 m/s. The study was focused on the building process of the structure, and they provided a qualitative comparison of hand-held, cart, and tractor-based systems. They did not provide any data analysis methodology neither they presented any collected data.

2.4 Gap Analysis

The analysis of the literature showed the advantages of adopting autonomous Unmanned Ground Vehicles. These systems do not require human intervention like tractors or carts, and they can freely move in every kind of field without limited maneuverability. They are more lightweight than tractors causing less soil compaction. They do not require

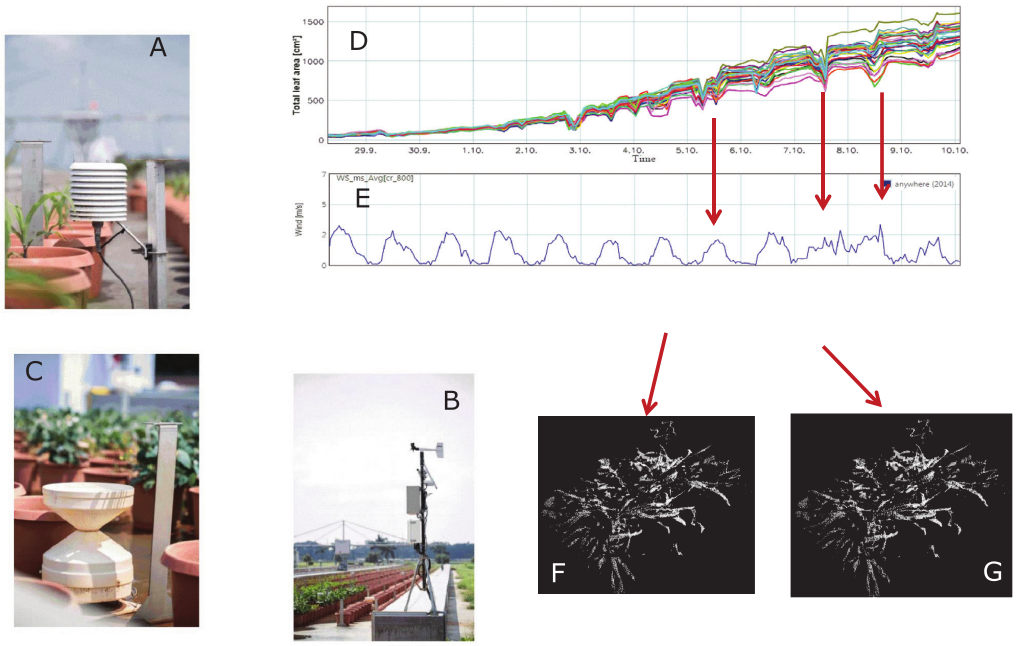


Figure 2.15: An example of LeasyScan gantry-based system collected data. (A–C) Set of environmental sensors: (A) temperature, relative humidity, (B) solar radiation, wind speed, (C) rainfall. (D) Information on plant parameters in time visualized through web-based software interface (Hortcontrol). Environmental data visualized in Hortcontrol, e.g., (E) wind. (F, G) 3D-point clouds accessed from Hortcontrol, at the LeasyScan platform. The Hortcontrol allows the basic data operations and quality control (e.g., data obtained during the windy part of the day (F) are of less quality compared to data obtained during windless part of the day (G) and are filtered for further analysis. (Image source [23])

Figure 2.16: View of the proximal sensing cart in a field of camelina. (Image source [24])

installation of bulky, cumbersome and expensive structures like gantries and they can still carry a fair sensor payload.

However, there is the need to understand what is the minimal and less expensive set

of sensors that can still provide useful phenotyping information and accurate localization. Moreover, effort should also be put to understand how sensors' positioning influences successive data processing and analysis procedures.

Moreover, more sophisticated localization methods should be investigated. The entirety of approaches exploits the GPS as principal localization information. Nonetheless, the GPS is an unreliable source of data as the signal is susceptible to environmental conditions and shadows of satellites from natural obstacles such as plants. The localization approach should neither be based on laborious and time-consuming works of manually tagging single plants. It should instead exploit other sensors information beyond the GPS. For example, visual information from cameras images or laser scans from LiDAR sensors. Thus, alternative localization algorithms should be investigated. Moreover, the position of single plants could be used as a reference for navigation purposes.

Finally, single plants segmentation is an essential characteristic that phenotyping systems should have. Recognizing single plants will help not just the navigation approach but will also allow tracking the plants' growth over time. To the best of our knowledge, no study has still proposed a Data Processing approach to distinguish single plants from a vineyard reconstruction. By such a new approach, it will be possible to assign a unique identifier to every single plant and recognize them in subsequent data collection without human intervention. This framework leads to the concept mentioned above of *Plant Digital Twin.*

Chapter 3

Background

In this Chapter, we give the reader an overview of the major frameworks, tools, and algorithms utilized in this thesis. In Section 3.1, we describe the Robot Operating System (ROS) and its core concepts with a focus on some relevant packages. In Section 3.2, we explain a typical robotics problem called Simultaneous Localization and Mapping (SLAM). Finally, in Section 3.3, we present the Point Cloud Library (PCL), a widespread library to process point clouds, a common data structure to represent 3D reconstructions.

3.1 Robot Operating System (ROS)

The Robot Operating System is an open-source set of frameworks for programming a robot. It is a collection of tools, libraries, and conventions for easy development of robot software. Since for single individuals the complexity of programming an entire robot stack is high, the ROS project aims at the collaboration of teams specialized in specific areas of knowledge. Common research areas are mapping, obstacle avoidance, localization, and others. From ROS Wiki:

> *It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.*[1]

In the following, we explain the main ROS components and terminology.

Packages All ROS code is organized in packages. A package contains all the files that serve for a specific purpose. For example, a package could be a driver for interfacing with a sensor or a piece of software for robot localization. Packages are the most atomic unit of build and the unit of release. This means that a package is the smallest individual thing you can build in ROS and it is the way software is bundled for release (meaning, for example, there is one Debian package for each ROS package), respectively.

[1] `http://wiki.ros.org/ROS/Introduction`

ROS packages tend to follow a common structure. Here are some of the directories and files you may notice.

- `include/package_name`: C++ include headers
- `msg/`: Folder containing Message (msg) types descriptions. There are two parts to a .msg file: fields and constants. Fields are the data that is sent inside of the message. Constants define useful values that can be used to interpret those fields (e.g. enum-like constants for an integer value). Message types are referred to using package resource names. For example, the file `geometry_msgs/msg/Twist.msg` is commonly referred to as `geometry_msgs/Twist`[2]
- `src/package_name/`: Source files, especially Python source that are exported to other packages
- `srv/`: Folder containing Service (srv) types descriptions
- `scripts/`: executable scripts
- `CMakeLists.txt`: CMake build file
- `package.xml`: an XML file that must be included with any catkin-compliant package's root folder. This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages[3]
- `CHANGELOG.rst`: Many packages will define a changelog which can be automatically injected into binary packaging and into the wiki page for the package

The Master Now we shift the focus from the organization of code to its execution. The ROS software architecture is compliant with the publisher-subscriber framework. The idea is to create a direct connection between processes so to exchange messages. The ROS Master process serves as a well-known entry point for naming and registration. To better understand the concept of Master, we propose the typical example of publisher and subscriber processes (see Figure 3.1). A publisher is a process that exhibits a stream of data with a unique name. Moreover, it communicates this name to the Master. Another process, the subscriber, can declare the intention to receive data from a stream with the same name. The ROS Master sends to the latter process the couple "IP:Port" in order to create a direct connection between publisher and subscriber.

Nodes Node is the name for ROS processes. The publisher and subscriber processes cited in the previous chapter are nodes. Nodes work together forming a graph structure. All running nodes have a (Graph Resource) Name that uniquely identifies them within the ROS computation graph. The ROS naming convention has a hierarchical structure of namespaces as it allows encapsulating names of resources (like nodes and topics). For example, the node /wg/node1 has the namespace /wg.

[2] `http://wiki.ros.org/msg`
[3] `http://wiki.ros.org/catkin/package.xml`

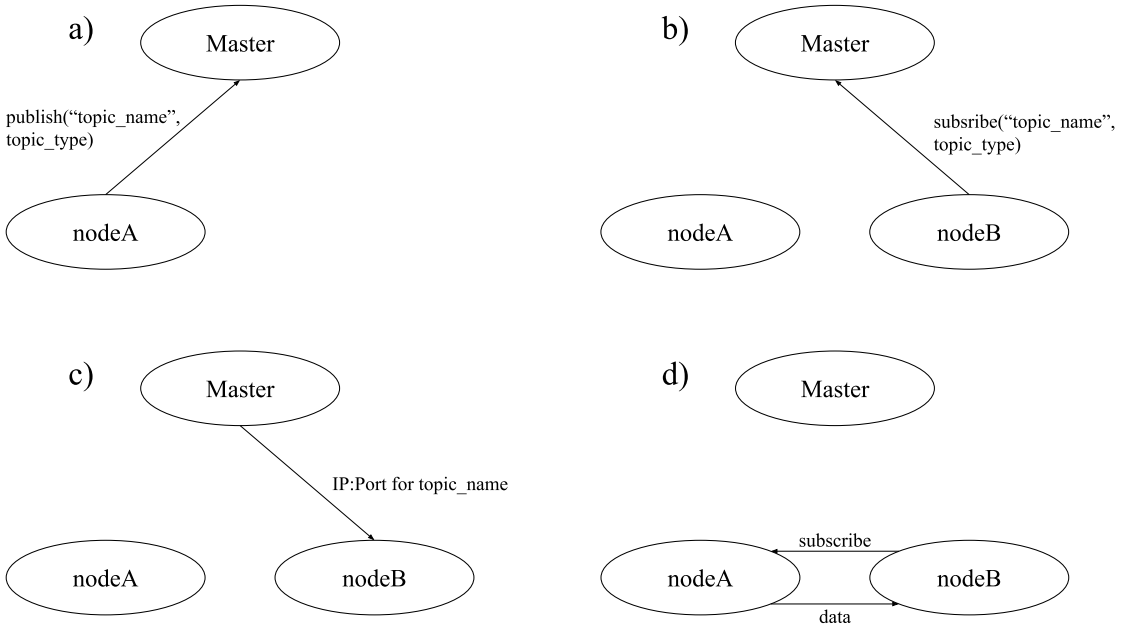


Figure 3.1: Here we see an example of a typical ROS nodes communication via topic. Starting from a), the nodeA inform the Master that it is publishing a topic called "topic_name" and of type topic_type. In b), we see another node called nodeB that asks the Master the network information to subscribe to a topic named "topic_name". Then, in c), the Master sends to nodeB the information required and finally, in d), the two nodes can communicate directly.

Topics Topics[4] are the way through which ROS nodes exchange messages. A topic is a unidirectional, asynchronous, strongly typed, named communication channel. Topics are characterized by the anonymity of the publisher/subscriber nodes. In general, nodes are not aware of who they are communicating with. Thus, a publisher node writes its data to a topic with a specific name related to a message type. Instead, a publisher node read data from a topic that has the name related to the message type the node is interested in. For example, an IMU filter node publishes messages on the `/imu/data` topic. Topics follow the same naming convention for nodes having a unique Graph Resource Name.

Messages Messages are the information exchanged by nodes via topic communication channels. They are a specific data structure based on a set of built-in types.

Launch files Launch files are XML to launch conveniently multiple nodes with a single command. Moreover, one can do the necessary name remappings, set parameters, and include other launch files. When a launch file is run, the Master node is started automatically.

Bags A bag is a file format for storing ROS message data. Bags can be replayed offline acting as a publisher without any difference from the online topics.

[4] `http://wiki.ros.org/Topics`

3.1.1 `tf` Package

The `tf` package serves for managing the coordinate frames of the robot. A robot has typically various 3D coordinate frames attached to its body (Figure 3.2).

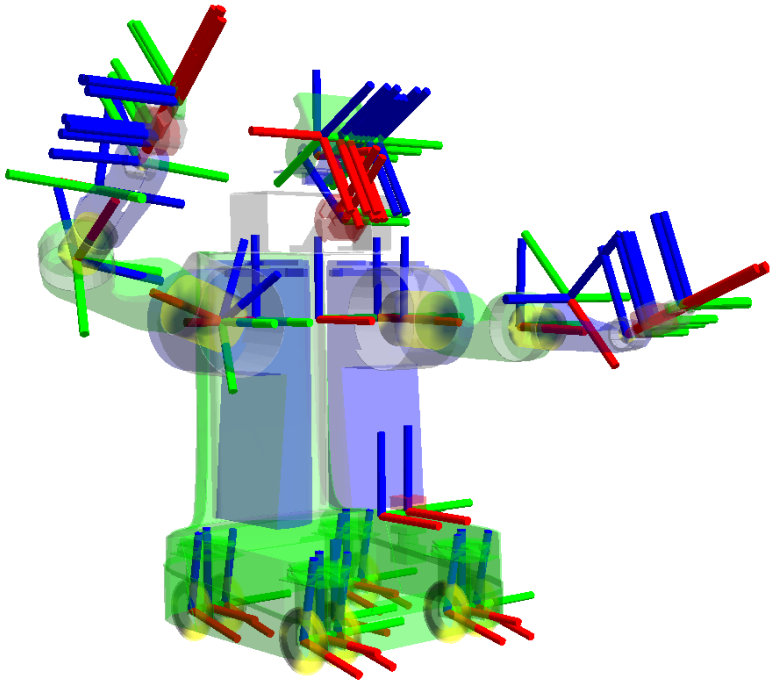


Figure 3.2: A robot with multiple `tf` frames for components like head, hands, etc. (Image source [26])

Each sensor and actuator is characterized by a frame. The `tf` package keep track of the transformations (`tf` stands for "transformation") between the various coordinate frames. Notice that, as the robot moves, some transformations could change values. This task is critical since all the information coming from the robot should be related to a common reference frame. For example, if a robot has two LiDAR sensors, how can we combine the information coming from them if we do not know their positions relative to, let us say, a world coordinate frame?

An important thing to highlight is the structure used to organize transformations, from the ROS Wiki:

> *tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time.*[5]

To better understand the concept of the transformations, let us present a real-world problem. We have a mobile robot base with a laser sensor mounted on the top of it. The laser sensor gives us the distance between the laser origin and the obstacle in front of it. However, we would like to know the distance between the robot front and the obstacle in order to avoid it. This setting is depicted in Figure 3.3.

[5] `http://wiki.ros.org/tf`
[6] `http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF`

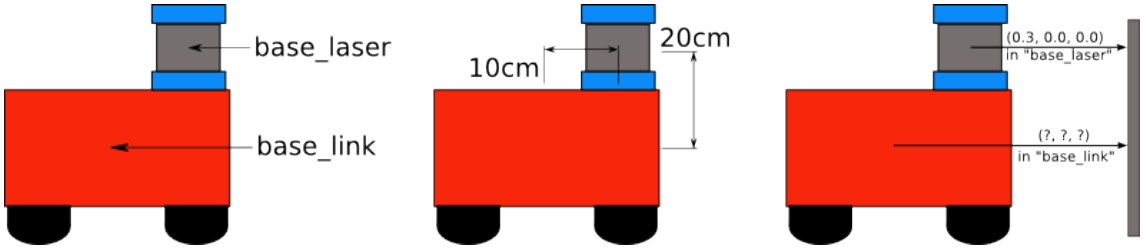


Figure 3.3: At the left you see the position of the `base_laser` and the `base_link`; in the center you see the translation between the two mentioned points; at the right you see the distance computed by the laser, and the problem is to know the distance from the center of the robot and the obstacle.[6]

There you can see the laser sensor center, the `base_laser`, and the robot base center, the `base_link`. With the robot dimensions, knowing the distance from the robot front or center to an obstacle is the same. The center of the robot is a convenient point to which relate all the transformations. To insert a transformation in the `tf` tree, we need to know the rotation and translation between the two elements. We imagine the robot coordinate frame as a right-handed reference system with the x-axis pointing to the front of the robot and the z-axis pointing upward.

In this case, the laser sensors origin is perfectly aligned with that of the robot base, so the rotation is the identity. As for the translation, the sensor origin is 10 cm (0.10 m) forward along x-axis relative to the robot center, and 20 cm (0.20 m) upward along z-axis relative to the same robot center. This is the transformation to add in the `tf` tree, as in Figure 3.4. In this way, we can know the distance from the robot center to the obstacle. The above-explained transformation is called "static" because it remains the same during the robot operations. If the transformation were between the robot center and a robotic arm, it would have been "dynamic".

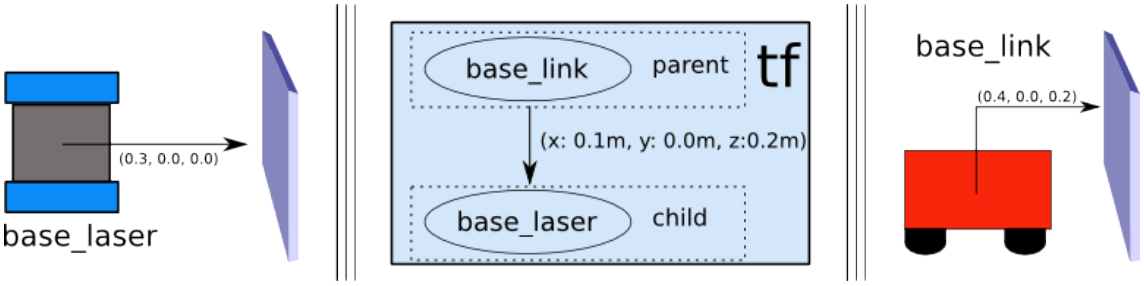


Figure 3.4: The laser sensors measured distance is translated to the distance between the robot center and the obstacle via the `tf` tree[7]

A typical `tf` tree contains at least other two relevant coordinate frames, that is, the `map` and the `odom` frames. In Figure 3.5 you can see an example of a general `tf` tree. The frames names and semantic meanings adhere to the ROS conventions that can be found at this link `https://www.ros.org/reps/rep-0105.html`. The `odom` frame has to be connected by a transformation to the aforementioned `base_link` frame and it embeds the robot odometry source such as wheel odometry, visual odometry or others. The robot position relative to this frame can drift over time without any bound, so the `odom` frame

[7] `http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF`

is useless as a long-term global reference. The robot pose in the `odom` frame is guaranteed to be continuous, that is, it does not have discrete jumps.

Instead, the robot pose in the `map` frame should not significantly drift over time so it can be used as a long-term global reference. However, the `map` frame it is not continuous, so discrete jumps in position estimators make it a poor reference frame for local sensing and acting.

The last reference frame is the `earth` frame, and it is used when multiple robots with different `map` reference frames need to interact. Otherwise the `earth` frame it is not used.

Figure 3.5: An example of a complete ROS `tf` tree. Since the `earth` frame is present, this tree could be of a robot that interacts with robots that have a different `map` frames. The tree structure after `base_link` tree is usually more complex and richer of sensors and actuators frames.

3.2 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is one of the most common problems faced by the robotics community. As the name suggests, the task consists in knowing the robot position in an unknown environment while building a map of the surrounding. The problem is not trivial because the robot, to know its location, needs to know how the surrounding environment is structured, that is, it needs to have a map. On the other hand, to build a map of the environment, the robot needs to know where it is.

Even if it appears to be a chicken-and-egg problem, several approximate solutions have been proposed in literature since the early 1990s. The problem is made difficult by the fact that sensors give imprecise measurements. The noise affecting sensors data introduces errors in the estimation of both the robot position and the map landmarks location. Some of the most common solutions include methods like the particle filter, extended Kalman filter, and graph-based SLAM.

3.2.1 Visual and Laser Odometry

An essential component of all SLAM algorithms is the odometry. Odometry is a typical robotics task. It consists of computing the change in position of the robot over time, relative to a starting position. Traditionally, odometry has been computed in many ways. One of the most common forms is wheel odometry.

Wheel odometry is calculated by integration of velocity measurements over time. The velocity information is given by sensors, called encoders, applied to the shafts of the wheel motors. In recent years other forms of odometry have been researched to overcome the errors introduced by velocity integration. One common approach is Visual Odometry (VO).

Visual Odometry (VO) works similarly to wheel odometry by incrementally estimating the robot pose through the examination of the changes that motion induces on the images of its onboard cameras. For VO to work effectively, there should be sufficient illumination

and a static scene with enough texture to allow the apparent motion to be computed. Furthermore, consecutive frames should be captured by ensuring that they have sufficient scene overlap.

The advantage of VO compared to wheel odometry is that VO is not affected by wheel slip in uneven terrain or other adverse conditions. It has been demonstrated that compared to wheel odometry, VO provides more accurate trajectory estimates, with relative position error ranging from 0.1 to 2%. This capability makes VO an interesting supplement to wheel odometry and, additionally, to other navigation systems such as global positioning system (GPS), inertial measurement units (IMUs), and laser odometry. In GPS-denied environments, such as under-water and aerial, VO has utmost importance. [27]

Another alternative to wheel odometry, similar to VO, is laser (or LiDAR) odometry. Laser odometry estimates the motion of a vehicle by scan-matching of consecutive laser scans. To perform this matching is usually adopted an algorithm called ICP (Iterative Closest Point). ICP takes two successive laser scans and finds the transformation to align one to another; this transformation is precisely the change of position of the robot between that two sensors measurements.

`rtabmap` is a ROS wrapper of the RTAB-MAP SLAM library. Beyond a SLAM node, it provides other two nodes, namely `rgbd_odometry` (or `stereo_odometry` depending on the input camera) and `icp_odometry` to compute visual and laser odometry respectively. A detailed explanation of these two nodes can be found in Subsection 5.2.1.

3.2.2 `robot_localization` Package

`robot_localization` is a ROS package for fusing different sources of odometry or other motion information, such the one from an Inertial Measurement Unit (IMU), in order to have have a new odometry with increased accuracy. From the ROS Wiki:

> *`robot_localization` is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, `ekf_localization_node` and `ukf_localization_node`. In addition, `robot_localization` provides `navsat_transform_node`, which aids in the integration of GPS data.*[8]

The state to which this definition refers is a 15-dimensional tuple that describes the motion state of the vehicle:

$$(X, Y, Z, roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z})$$

The X, Y, Z components are the position of the robot reference frame origin in the 3D space. The $roll, pitch, yaw$ are the components that describe the orientation of the robot reference frame. Then we have the linear and angular velocities and the linear acceleration of the robot, respectively.

The package is the implementation of a Kalman Filter. The Kalman Filter is an algorithm that outputs a state estimation by weighting the different inputs based on their uncertainty. The inputs are the various inaccurate sensors measurements and the

[8] `http://docs.ros.org/melodic/api/robot_localization/html/index.html`

estimate from a prediction model. However, the filter works properly when all errors are Gaussian. Extensions and generalizations to the method have also been developed, such as the Extended Kalman filter and the Unscented Kalman filter, which work on nonlinear systems. These are precisely the algorithms implemented by the `robot_localization` nodes. For more information on technical details, refer to [28].

Additionally to Kalman Filter nodes, the package provides the `navsat_transform_node`. This node allows the integration of GPS data in the localization algorithm of the system. Raw GPS data is provided with a format difficult to be directly integrated into the coordinate frame to which the robot is referring. This node translates the GPS coordinates into a position relative to the robot world frame. In this way, the information is immediately available for usage, for example, as the input of a Kalman Filter.

3.2.3 `rtabmap_ros` Package

The `rtabmap_ros` package is a wrapper for the RTAB-Map library [29]. RTAB-Map is an open-source LiDAR and visual Simultaneous Localization and Mapping (SLAM) library for large-scale and long-term online operation. Let us break down word by word this definition.

LiDAR and Visual As previously said, to know their pose, robots need to perform measures of the environment with their sensors. SLAM is a general theoretical problem that has been studied without imposing a particular kind of sensor. The first applications of SLAM have been made with laser sensors as they are among the most accurate and require less post-processing computation than other data, like images. Successively, along with the advances in Computer Vision and camera sensors, visual-based SLAM has started to become popular.

> *SLAM approaches are generally visual-based or lidar-based only, and are benchmarked often on datasets having only a camera or a lidar, but not both, making difficult to have a meaningful comparison between them. [...] RTAB-Map can be used to implement either a visual SLAM approach, a lidar SLAM approach or a mix of both, which makes it possible to compare different sensor configurations on a real robot.* [29]

(Graph-Based) SLAM We have already described the SLAM problem, but in particular, RTAB-Map is a graph-based SLAM approach. Graph-Based SLAM approaches are based on the optimization of graphs that represent the map of the environment. Each node of the graph represents a robot pose. Each edge between two nodes encodes a sensor measurement that constrains the connected poses.

> *Obviously, such constraints can be contradictory since observations are always affected by noise. Once such a graph is constructed, the crucial problem is to find a configuration of the nodes that is maximally consistent with the measurements. This involves solving a large error minimization problem.* [30]

Large-scale and long-term online operation Due to the high computational complexity of the optimization problem for graph-based SLAM approaches, only relatively recently this approach became popular thanks to the advances of linear algebra techniques. Still, these algorithms tend to explode in memory consumption and computational load as time passes by, and the graph increases in the number of nodes. RTAB-Map tries to overcome this problem with a smart memory management strategy.

When performance degrades under a threshold, some nodes are moved from the Working Memory (WM) to a storage memory (Long Term Memory - LTM). They use some heuristics to decide when a node should be put in the LTM. The nodes stored in the LTM are moved back to the WM when happening some particular events that involve neighborhood nodes of the one stored. In this way, both computational time and memory usage are kept under thresholds. These thresholds are set to allow the system to continue to work in real-time. This means that the graph optimization processing time will not affect the consumption frequency of incoming sensors data. Moreover, one can perform online SLAM even in large environments where other SLAM algorithms will quickly run out of memory. This is one of the most innovative features of RTAB-Map.

3.3 Point Clouds

Environment perception is an essential characteristic of a robot. It serves for different purposes, such as obstacle avoidance, mapping, and localization. Laser sensors are an excellent way to describe the world because they measure distances with high accuracy. Moreover, they can be exploited to build a 3D model of the environment. These such models contain rich information that allows to carry out complex tasks compared to simpler models.

For example, in the case of obstacle avoidance, a table could not be seen by a 2D laser, placed at low height in front of a robot, with a restricted Field Of View. Conversely, it will be certainly perceived with a sensor that gives a 3D perception of the environment. Laser sensors costs from thousands to tens or hundreds of thousands of euros. However, with the advent of cheap 3D imaging sensors like RGB-D cameras, they are rapidly diffusing in the robotics community. 3D reconstructions can be supported by a variety of data structures such as point clouds, voxels grids, meshes, and surfel-based map representation. Point clouds are one of the most common structures used.

A point cloud is a data structure used to represent a collection of multi-dimensional points and is commonly used to represent three-dimensional data. In a 3D point cloud, the points usually represent the X, Y, and Z geometric coordinates of an underlying sampled surface. When color information is present (see figure 3.6), the point cloud becomes 4D.[9]

The points could be sampled from a real scene using sensors like LiDARs, stereo cameras, 3D cameras like the Kinect or Intel RealSense RGB-D cameras. Otherwise, the points can be synthetically created using Computer Graphics tools. In figure 3.7, you can see the point cloud of a rabbit.

[9] http://pointclouds.org/about/

[10] http://pointclouds.org/about/

[11] https://www.codeproject.com/Articles/839389/Fast-Point-Cloud-Viewer-with-Csharp-and-OpenGL

Figure 3.6: RGB Point clouds of common use objects. (Image source[10])

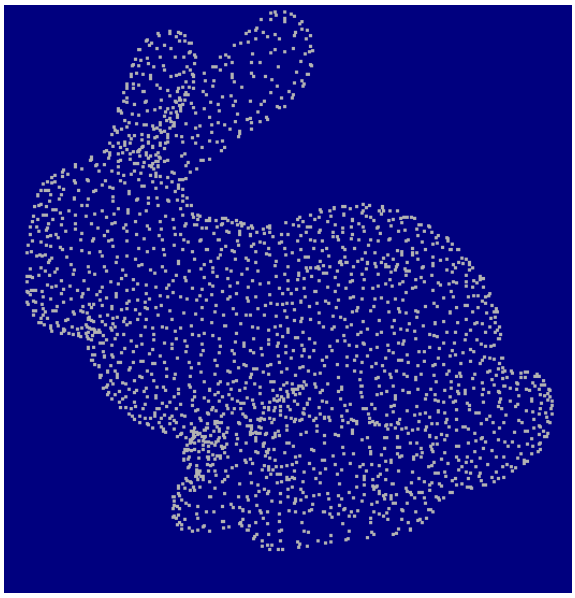


Figure 3.7: Point cloud of a rabbit. (Image source[11])

3.3.1 The Point Cloud Library (PCL)

Point clouds are usually composed of hundreds or thousands or even millions of points, depending on the scale of the scanned environment and the sensors technical characteristics. Such a huge quantity of data needs efficient algorithms to be handled.

The Point Cloud Library (or PCL) is a large scale, open project [31] for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the-art algorithms, including filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them; to name a few.[12]

Each set of algorithms to accomplish a particular function, such as segmentation, are derived from the same base classes. This entails that all processing tasks follow a common pipeline. If you are going to look at some PCL code, you will see this pattern repeated for every single task. The basic interface for such a processing pipeline in PCL is [31]:

[12] `http://pointclouds.org/about/`

- create the processing object (e.g., filter, feature estimator, segmentation)
- use `setInputCloud` to pass the input point cloud dataset to the processing module
- set some parameters
- call `compute` (or filter, segment, etc) to get the output

Moreover, PCL easily integrates with ROS. To efficiently process the extremely large point clouds, a core characteristic of PCL is the Perception Processing Graphs (PPG). As the typical ROS nodes graph structure, the PPG structure is characterized by nodes that perform single tasks such as normal estimation, segmentation, etc. Each node is executed in ROS as nodelet. Nodelets are an alternative computational unit to ROS nodes that use pointers passing instead of serialization/deserialization of messages over a network. In the case of a large quantity of data as for point clouds, nodelets are made necessary. In the following, we explain some relevant PCL library algorithms that are used in this work.

RANSAC RANSAC is the abbreviation of RANdom SAmple Consensus. It is an iterative method that is used to estimate parameters of a mathematical model from a set of data containing outliers. The algorithm takes as input the kind of model you expect to find in the data, for example, a plane. The assumption is that part of the data are inliers, and the rest are outliers. The goal is to find the part of the data that best represents the input model. The algorithm uses some threshold parameters given as input to discriminate between inliers and outliers. If the model is a plane are considered only as hypothetical inliers the 3D points within a certain distance along the perpendicular direction the considered plane.

RANSAC achieves its goal by iteratively selecting a random subset of the original data. These data are hypothetical inliers, and this hypothesis is then tested as follows:

1. A model is fitted to the hypothetical inliers, i.e., all free parameters of the model are reconstructed from the inliers
2. All other data are then tested against the fitted model and, if a point fits well to the estimated model, also considered as a hypothetical inlier
3. The estimated model is reasonably good if sufficiently many points have been classified as hypothetical inliers
4. The model is reestimated from all hypothetical inliers because it has only been estimated from the initial set of hypothetical inliers
5. Finally, the model is evaluated by estimating the error of the inliers relative to the model

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are classified as inliers or a refined model together with a corresponding error measure. In the latter case, we keep the refined model if its error is lower than the last saved model.[13]

[13] `http://pointclouds.org/documentation/tutorials/random_sample_consensus.php`

An advantage of RANSAC is the ability to estimate robust models even in the presence of many outliers. The disadvantage is the computational effort required. Indeed, the model gets more accurate as more iterations are performed. Moreover, the confidence parameters provided as input are problem-specific. A characteristic of RANSAC is that it can only find one model in the data. If multiple models are present, it may fail to find either one.

Euclidean Cluster Extraction Euclidean Cluster Extraction [32] is a clustering algorithm, that is, an algorithm that aims at grouping a set of data in such a way that the elements part of a group are more similar to each other than to the elements belonging to other groups. There are various algorithms to solve this problem; examples are Euclidean Cluster Extraction and Kmeans.

Clustering on point clouds is generally performed to isolate single objects or parts of them. For example, in the case of a vineyard point cloud, we would like to isolate single vine plants. As the name suggests, Euclidean Cluster Extraction uses the Euclidean distance between points. The idea is to group points if they are close to each other within a certain radius threshold. The pseudo-code of the algorithm is the following:

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

K-means K-means [33] is another clustering algorithm. In particular, K-means aims to partition n elements into a fixed number k of groups (or clusters). It assigns a point to the cluster whose centroids is the nearest. The centroid is the point at the geometric center of a cluster and has not necessarily to be an element of the cluster.

The algorithm is usually implemented iteratively. The input of the algorithm are the elements to cluster and a vector of initial centroids (the number k is derived from the size of the centroids vector). The initialization procedures for centroids are various, from random initialization to some heuristics or exploiting previous knowledge.

Algorithm 1, shows the pseudo-code of K-means applied on point clouds.

Algorithm 1: k-means on point clouds

input : A point cloud X of n points such that $x_i \in X$; the vector of initial centroids M of size k such that $\mu_j \in M$

output: A partition of the input point cloud X in k clusters

```
/* clusters initialization */
for i ← 0 to n do
    ClusterIndex(x_i) ⟵ i mod k;
end
/* iterate till convergence */
while not_converged do
    not_converged ⟵ false;
    /* iterate over all points */
    for i ← 0 to n do
        min ⟵ Distance(Centroid(ClusterIndex(x_i)), x_i);
        /* iterate over all clusters */
        for j ← 0 to k do
            d ⟵ Distance(μ_j, x_i);
            if d < min then
                min ⟵ d;
                ClusterIndex(x_i) ⟵ j;
                not_converged ⟵ true;
            end
        end
    end
    /* compute centroid of each cluster */
    for j ← 0 to k do
        for i ∈ ClusterIndices(j) do
            μ_j ← μ_j + x_i;
        end
        μ_j ← μ_j / Size(ClusterIndices(j));
    end
end
```

Chapter 4

Hardware Architecture

In this Chapter, we explain the rationale behind our hardware architecture. In Section 4.1, we make a list of a set of sensors that we investigated for phenotyping applications. We describe their characteristics and functioning principles, and we explain how we selected the best subset of them. In Section 4.2, we explain how the sensors supporting platform has been built and calibrated. Finally, in Section 4.3, we list the multiple setups of sensors that we investigated during outdoor field experiments.

4.1 Sensors

In this Section, we make a list of sensors that can be exploited to do plant phenotyping, as suggested in [5]. Starting from the literature, we investigated a set of devices to understand their advantages and disadvantages. The devices taken into consideration are representatives of the main state-of-the-art 3D imaging techniques. These technologies are laser scanning, stereo vision, Time-Of-Flight, and structured light. We tested each of the following listed sensors, both indoor and outdoor. The indoor scene was our lab while the outdoor scene was a group of plants in a park of our university.

Moreover, we operated the sensors under stress conditions like in bright light and even direct sunlight. To perform the experiments, we recorded raw data in ROS bag files (see Section 3.1). We then replayed back the bags, and we qualitatively evaluated the output point clouds and depth images. Finally, we drew conclusions also taking into consideration the technical specifications. Once we came up with a final set of sensors, we mounted them on a platform to further investigating their localization and reconstruction capabilities.

ZED Camera The ZED camera (see Figure 4.1) is a stereo camera, that is, a sensor that retrieves depth information by the stereo triangulation (see Figure 4.2) of two images coming from two different image sensors. The idea is to simulate the human binocular vision, and therefore to reconstruct three-dimensional images. The ZED camera is equipped with two RGB cameras, and it computes 6-DoF positional tracking via real-time depth-based visual odometry and SLAM. The camera needs to be used with a GPU; therefore, we employed the Nvidia Jetson TX2 GPU together with Ubuntu 16.04. We recorded bags with different settings of quality and frame rate. In Figure 4.3, it is showed the output

point cloud, and the respective RGB left camera image of a scene recorded in our lab. Notice how wavy and noisy is the point cloud as seen from the ROS visualization (`rviz`) software.

Figure 4.1: ZED Camera. (Image source[1])

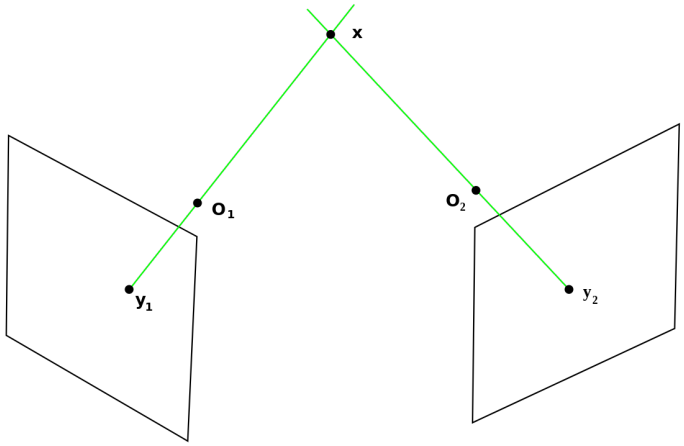


Figure 4.2: This Figure explains the principle of stereo triangulation. The two images with overlapped views are matched to find points correspondences. $\mathbf{y_1}$ *and* $\mathbf{y_2}$ *are a couple of matched points. The 3D point* $\mathbf{x}$ *is detected by the intersection of two lines passing through the images points and their respective camera focal points. (Image source[2])*

Kinect One (v2) Kinect One (or Kinect v2) (see Figure 4.4) is a motion-sensing input device that was produced by Microsoft for Xbox One video game console. It is equipped with an RGB camera, an IR (infrared) camera, and three IR projectors. This new Kinect version has greater accuracy with three times the fidelity over its predecessor. Moreover, it can track without visible light with the help of its projectors. It provides depth information exploiting a Time-Of-Flight (TOF) technology (see Figure 4.5). The TOF principle is the following: knowing the speed of light, the distance to be measured is proportional to the time the projectors' light takes to travel from the emitter to the obstacle and then back to the IR sensor.

To have more accurate depth information, we calibrated the Kinect following the procedure described here

[1] `https://www.stereolabs.com/zed/`
[2] `https://en.wikipedia.org/wiki/Triangulation_(computer_vision)`
[3] `https://www.stemmer-imaging.com/en/knowledge-base/cameras-3d-time-of-flight-cameras/`

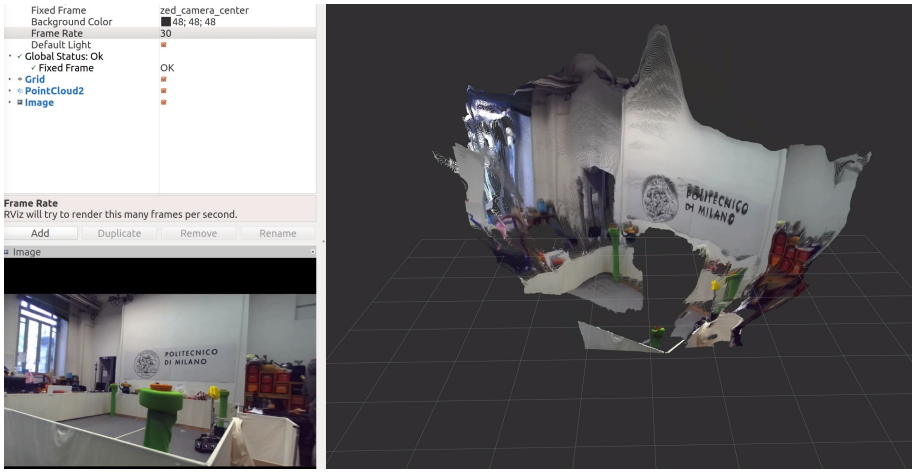


Figure 4.3: Our lab point cloud from the ZED camera seen on the `rviz` software. In the bottom-left, you can see the left RGB camera image.

Figure 4.4: Kinect One (v2).

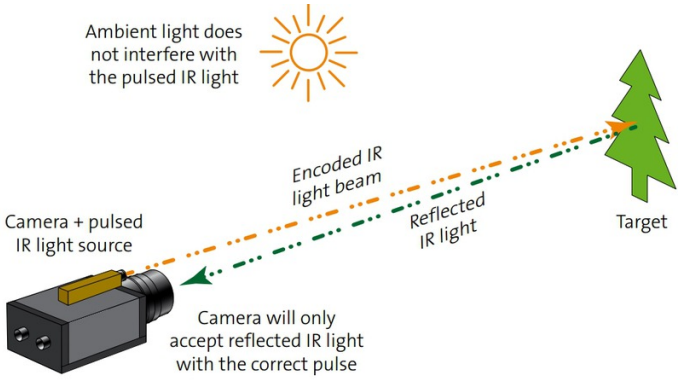


Figure 4.5: This Figure explains the principle of Time-Of-Flight. The camera projects a modulated light toward the object to measure. The sensor transforms the detected light into an electric signal, and the distance is computed by the phase shift between the emitted and reflected light. (Image source[3])

`https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration`. We repeatedly acquired images of a calibration checkerboard from different perspectives and distances. We gave all the images as input to the calibration algorithm that gave us the calibration matrices. In Figure 4.6, it is shown the difference before and after calibration. Figure 4.7, shows again the lab point cloud captured with the Kinect. Notice the

fidelity of shapes to the real scene. Instead, in Figure 4.8, is visualized the point cloud of an outdoor scene with a bunch of trees.

(a) Before: note the double borders at the edges of the objects.

(b) After: note how the color is now correctly applied on the depth data.

Figure 4.6: Difference between before and after Kinect calibration. The images represent the color image superimposed to the depth image.

In [34], the authors analyzed the Kinect One sensor to understand its suitability for close-range 3D modeling. Moreover, they made a comparison evaluation with its predecessor and as an alternative to photogrammetry methods.

Kinect v2 showed to be superior compared to its predecessor due to the newer depth technology. The comparison with photogrammetry methods tends to confirm the higher efficiency of image-based reconstruction methods when accurate 3D models are required, regardless of the object size. However, a satisfactory accuracy of 1 cm was reached by Kinect v2 in experimental reconstruction tests. They also investigated Kinect pre-heating time and calibration methods. Indeed, as reported in many contributions dealing with RGB-D cameras, a pre-heating time has to be considered to obtain constant measurements. It has been estimated that almost 30 minutes is needed with the Kinect v2 sensor.

In [35], they found similar results concerning the reconstruction accuracy.

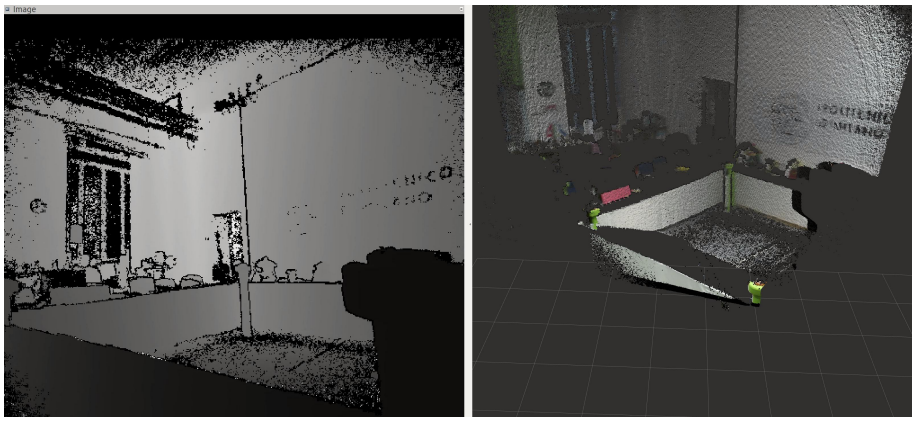


Figure 4.7: On the right, the point cloud captured by Kinect One of the previously showed lab scene. On the left the relative depth image.

XtionPRO Live XtionPRO live (see Figure 4.9) is a motion-sensing device produced by Asus for gaming applications. It is similar to the Kinect v1 device as they both are

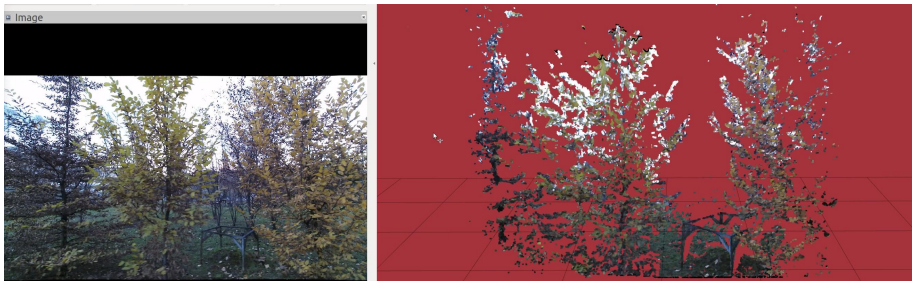


Figure 4.8: On the left a real outdoor scene as captured by the Kinect RGB camera. On the right, the relative computed point cloud.

structured-light cameras. It is equipped with an RGB camera, an IR sensor, and an IR structured-light source. Structured-light works by projecting a known pattern on the subject to measure. The distortion of the projected pattern is correlated to the distance of the subject (see Figure 4.10). In [36], the authors present various structured-light techniques and applications. In Figure 4.11, it is shown the already mentioned lab scene as captured by the XtionPro sensor. In [37], the authors showed the potential of low-cost 3D cameras (XtionPro Live and Kinect v1) for the measurement of plant woody structure.

Figure 4.9: XtionPRO live.

Intel RealSense D435(i) Camera The Intel RealSense D435 (see Figure 4.12) is an RGB-D (the D stands for Depth) camera. It is a device that, similarly to the Kinect and the Xtion, provides color and depth information. It is equipped with an IR stereo camera (two IR sensors), an IR projector, and an RGB camera. The model D435i additionally provides 6-DoF position information via the internal IMU.

The RealSense D4XX cameras series depth principle is stereo triangulation as the one of ZED camera. However, in this case, two IR images are used in place of RGB ones. The infrared projector projects non-visible static IR pattern to improve depth accuracy in scenes with low texture. The left and right sensors capture the scene and send images data to the depth imaging processor, which calculates depth values for each pixel in the image by correlating points on the left image to the right image. The feature enhancement caused by the IR projector leads to a higher depth accuracy than simple stereo vision. In Figure 4.13, it is shown the already mentioned outdoor scene captured by the D435 camera.

[4] `https://imgbin.com/png/wjRvyKrG/kinect-structured-light-3d-scanner-structured-light-camera-png`

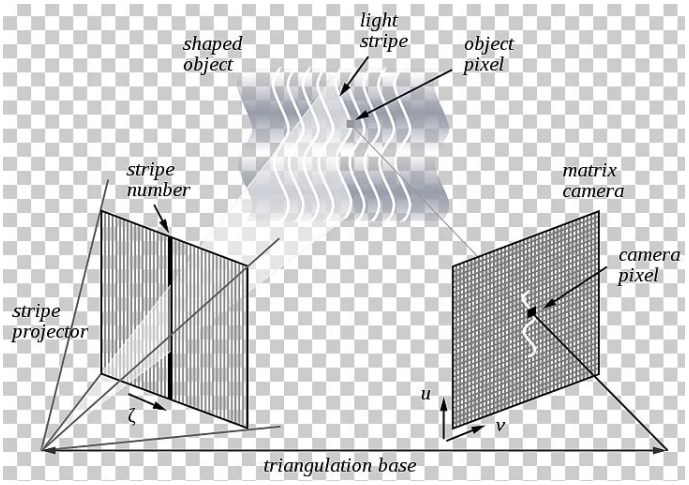


Figure 4.10: The structured-light principle allows the computation of depth information via an IR camera and an IR light source. The principle is similar to stereo vision with the difference that a light emitter replaces a camera. The IR source, on the left, projects a vertical lines pattern on the subject to measure. The undulated object reflects a distorted line that is captured by an IR camera, on the right. Then, if the physical relation between the camera and the projector is known, the depth of a point can be calculated via triangulation. (Image source[4])

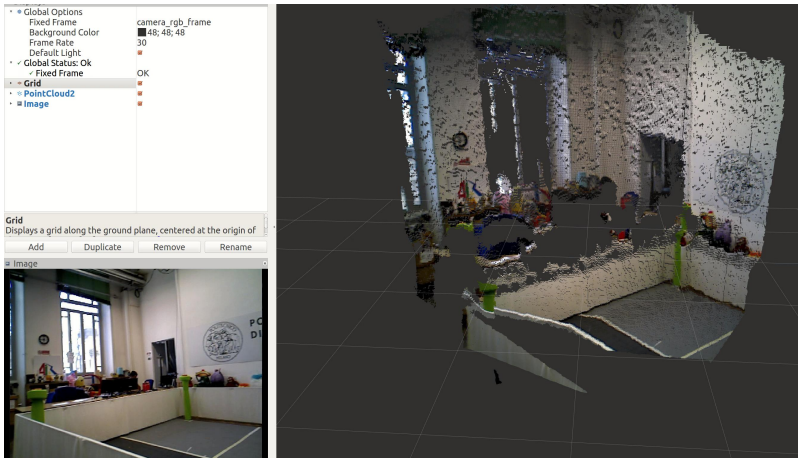


Figure 4.11: On the left the lab scene as captured by the XtionPro camera. On the right, the relative computed point cloud.

Figure 4.12: Intel RealSense D435(i) Camera.

Sick LMS100-1000 2D LiDAR The Sick LMS100 (see Figure 4.14) is a 2D laser sensor, that is, a device with a single scanning plane. The laser sensors retrieve with

Figure 4.13: On the left, the outdoor scene RGB image as captured by D435 camera. On the right, the relative computed point cloud. Note the higher level of detail compared to the Kinect outdoor point cloud.

high accuracy distance information of the subjects intersecting the scanning plane. The principle is very similar to Time-Of-Flight. The distance is calculated as proportional to the time the laser beam needs to return after being emitted and reflected by an obstacle (see Figure 4.15).

Moreover, LiDAR sensors measure the intensity of the reflected beam giving additional information that can be exploited in the post-processing pipeline. The intensity is based, in part, on the reflectivity of the object struck by the laser pulse. In [38], the authors show the potential of laser and ultrasonic sensors for measuring phenotyping data of vineyard trees. The obtained results indicated that an ultrasonic sensor is an appropriate tool to determine the average canopy characteristics, while a LIDAR sensor provides more accuracy and detailed information about the canopy.

Figure 4.14: Sick LMS100-1000 2D LiDAR.

[5] https://www.elprocus.com/lidar-light-detection-and-ranging-working-application/

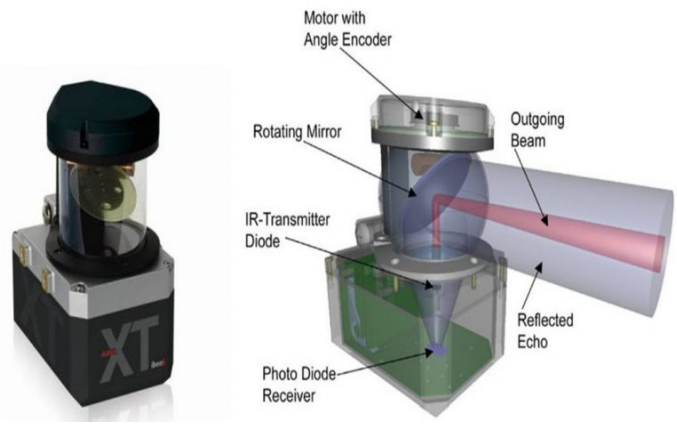
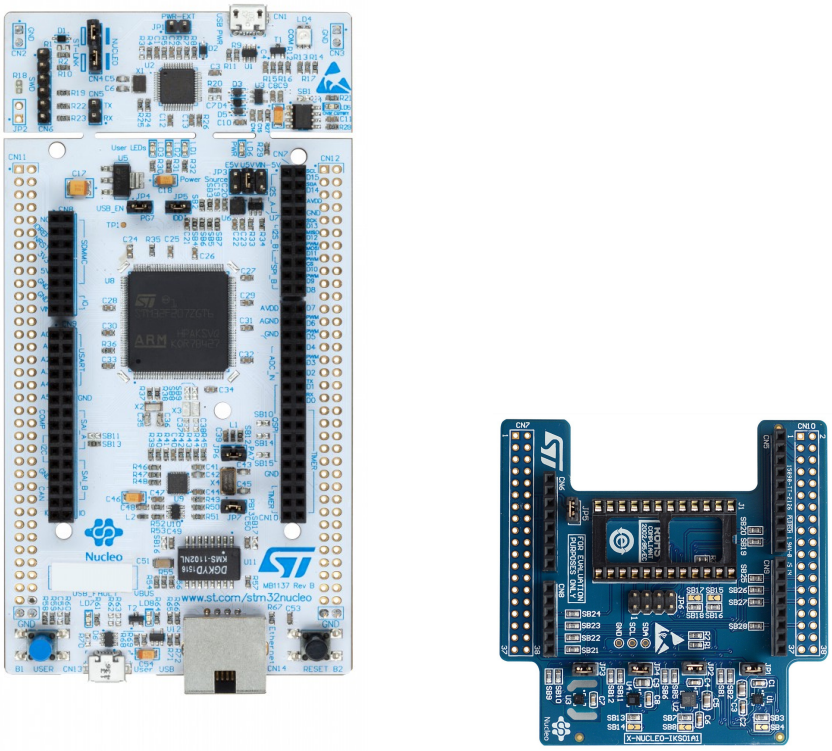


Figure 4.15: This Figure explains the principle behind LiDAR sensors. A laser beam is emitted by a rotating mirror toward the subject to measure. The rotating mirror captures the light reflected by the subject and directs it toward a photodiode sensor that transforms the light pulse to an electric signal. The distance is proportional to the time the light needs to travel from the emitter and return to the target. (Image source[5])

Sick LD-MRS400001S01 3D LiDAR The Sick LD-MRS (see Figure 4.16) is a 3D LiDAR. Unlike the LMS100 model, it scans along four different planes. Moreover, it gives the possibility to use just two layers with the advantage of an increased angular Field Of View.

Figure 4.16: Sick LD-MRS400001S01 3D LiDAR.

Other sensors We also have taken into consideration the STM32F746 Nucleo-144 board with on top an X-NUCLEO-IKS01A1 multi-sensors board that provides IMU and magnetometer data (see Figure 4.17). Finally, for absolute earth-referenced positioning information, we considered a Trimble 5700 GPS receiver (see Figure 4.18) with its external antenna.



(a) STM32F746 Nucleo-144 board.

(b) X-NUCLEO-IKS01A1 multi-sensors board.

Figure 4.17: STMicroelectronics Nucleo-144 and X-NUCLEO-IKS01A1 boards.

Figure 4.18: Trimble 5700 GPS receiver.

4.1.1 Comparison

Here we make a comparison between the sensors for 3D imaging taking into account their final application, that is, outdoor plant phenotyping. The following considerations come from the investigation of the literature and the inspection of sensors technical specifications and of the already mentioned recorded data in indoor and outdoor environments. The complete and schematic comparison is represented in Table 4.1.

The most critical parameter is the depth range. Depth range is the distance interval in which the sensor can provide measurements of the scanned subject. The lower limit of the range is critical for us as we are investigating proximal sensing platforms. Thus, if the low range is too far from the sensor origin, close plant scanning cannot be performed. All the sensors perform well as exception of the Xtion that has a too far low limit range (0.8 m) for most phenotyping applications. The D435 has the smallest and remarkable low limit range of about 0.1 m. The upper limit range is not a problem since it is at least in the order of meters for all sensors.

Bright-light performance is either a crucial parameter since data collection is usually performed during day hours. Indeed, devices that employ IR information are sensitive to sunlight. Structured-light technology has interference issues in outdoor environments. Low-cost versions of structured-light cameras such as the XtionPro have low resolution and are highly sensitive to outdoor lighting [39]. The problem of lack of performance under sunlight is present even in Time-of-Flight (ToF) cameras. Some of the ToF cameras have an on-board background illumination rejection circuitry, but with varying performance under sunlight depending on the operating range and the power of NIR (Near Infrared Radiation) emitters [39]. In [40], the authors showed that the Microsoft Kinect v2 performed worse on depth measurements quality at higher light intensities compared to the Intel D435 camera. As for stereo cameras, even if they are less sensitive to sunlight than ToF cameras, direct sunlight and shadows in a sunny day affect strongly their depth image generation [41], [39]. Instead, LiDAR sensors are very little susceptible to sunlight, and they generally perform well in outdoor environments [42]. Thus, we gave to the XtionPro and the Kinect v2 the worst bright-light performance. Moreover, we placed the D435 and the ZED cameras at a medium quality level, and the LiDAR sensors at the highest quality level.

In low-light condition, the ZED stereo camera and the D435 camera perform the worst since their sensors exploit ambient illumination. Instead, other sensors are advantaged by an active depth sensor technology that works independently of visible light. However, phenotyping measurements are supposed to be performed during day hours. Further research should be done to understand the consequences of evening/night hours measurements.

As for depth accuracy, the LiDARs perform the best compared to all other sensors [42]. Depth accuracy of stereo cameras varies with the type of algorithm used for correspondences matching, and the performance is adversely affected by the lack of surface texture on the object. In [39], they concluded that even if stereo cameras should be advantaged of accurate stereo matching algorithms with an efficient implementation over specialized hardware, they still can not be comparable to ToF cameras. Thus, we assigned to the ZED camera the lowest depth accuracy. Although the D435 is a stereo camera, it has a good accuracy because of the IR images increased in features by the IR projector. These extra features to track ease the stereo matching task. In [40], the authors demonstrated

that the Intel D435 performed better than Microsoft Kinect v2 in different depth accuracy tests. However, after visual inspection of outdoor reconstructions, we experienced that the Kinect v2, the XtionPro and the D435 camera have the same depth accuracy level.

As we are oriented toward the best effective but cheap subset of sensors is interesting to take into consideration the economic aspect. We list the sensors ordering them by increasing price: RealSense D435, Kinect v2, XtionPro live, ZED camera, LMS100, LD-MRS. The LiDARs cost thousands of euros while the other sensors cost hundreds of euros.

Finally, it would be better to have a compact and lightweight suite of sensors. The most compact and lightweight sensor is the D435. Instead, the LiDARs are heavy sensors. Unexpectedly, the heaviest and biggest sensor is the Kinect v2.

Concluding, the XtionPro live is the first sensor to be excluded as it performed the worst in the most critical parameters. In addition to the previous considerations, it also has the narrowest Field Of view. This outcome was predictable as the Xtion is an older device relative to the other cameras. The ZED camera is also excluded as the stereo technology entails a lower depth accuracy, and an imprecise model leads to errors in the phenotyping measurements. Finally, the Kinect v2 is excluded favoring the RealSense D435 since the latter is purposely designed both for indoor and outdoor environments, thus having better bright-light performance. Moreover, the D435 is more lightweight, smaller, and cheaper.

As a result of this comparison, we decided to mount the D435 camera and both the LiDAR sensors as they have an unattainable high depth accuracy. The future aim is to find a way to merge the accurate LiDAR data with the color information from the D435 cameras.

4.2 Platform Construction and Calibration

The platform (see Figure 4.19) is constituted of a four pneumatic wheels garden trolley — the trolley (see Figure 4.20) steers by its front two wheels connected to the handle by an axle. The platform is thought to be manually pulled by the handle. It has a declared load capacity of 250kg. It supports a movable wood box structure, and on top of that, it is placed a movable aluminum chassis supporting the sensors. The chassis is build by modular **item®** aluminum profiles.

The computational system is constituted by an Nvidia Jetson TX2 GPU that allows keeping pace to the broad data stream produced by the camera. The Nvidia GPU is accessed via Ubuntu Remmina Remote Desktop through a client notebook PC. To provide the necessary data connections, we mounted the Netgear GS108 1 Gbit switch and a USB 3.0 Hub. The power is provided by two 12V lead-acid batteries connected in parallel. As the Nvidia GPU and the PC require 19V power supply, we mounted the DCDC-USB-200 Mini-Box DC-DC converter. In Figure 4.21, you can see the diagram representing data and power connections.

Calibration The calibration is the process through which the transformations among sensors are assessed. These transformations are precisely those concerning the `tf` tree as explained in Subsection 3.1.1. Moreover, in that same Subsection, is explained why the transformations are necessary to exploit sensors data. We manually calibrated our sensors with rulers. For our purposes, this method has been sufficiently accurate for the

	ZED Camera	Kinect One	XtionPro live	RealSense D435	LMS100	LD-MRS
Depth Range	Good	Good	Bad lower limit	Good	Good	Good
Bright-light Performance	Medium	Weak	Weak	Medium	Good	Good
Field Of View	Wide	Wide	Narrow	Wide	Wide angle	Narrow angle
Depth Accuracy	Low	Medium	Medium	Medium	High	High
Cost	Medium	Low	Low	Low	High	High
Low-light Performance	Weak	Good	Good	Weak	Good	Good
Weight	Low	High	Medium	Low	High	High
Size	Medium	Large	Medium	Small	Large	Large

Table 4.1: A qualitative comparison of 3D imaging sensors for plant phenotyping. The evaluation parameters are listed by decreasing importance; meaning that parameters at upper levels count the most for considerations about their inclusion in a phenotyping sensor suite. Depth Range is the most important because a sensor with a too far lower limit cannot be used in most farm settings. Bright-light performance is necessary to be high as these sensors are thought to be used for outdoor phenotyping applications. Field Of View is either a critical parameter as a restricted FOV, worsen by short scanning distances, can limit the phenotyping to sections of the plants. Low-light performance is not so important as measurement processes are assumed to be done during day hours.

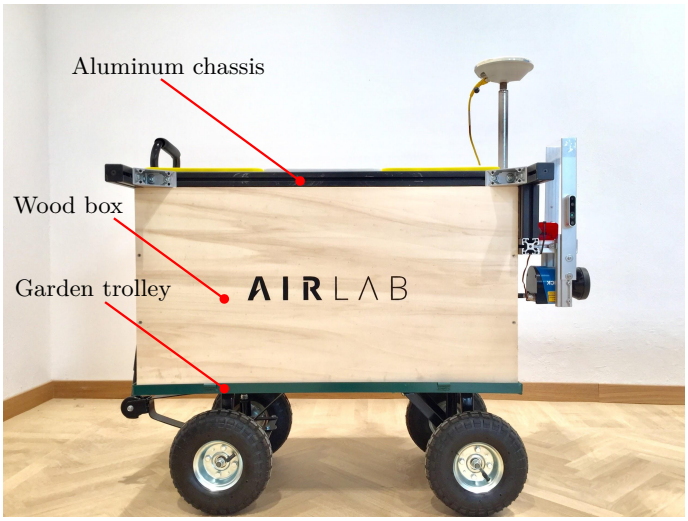


Figure 4.19: Platform components.

Figure 4.20: Cart supporting platform.

following reasons. The IMU does not need to be calibrated with high accuracy since we are just using the magnetometer for the ROS `navsat_node` that needs only to know the orientation relative to an earth reference when the robot starts operations. The LiDAR does not need an accurate calibration as we did not merge its data with those from the cameras. So, the LiDAR reconstruction is misaligned with that of the cameras with a certain degree of error. In the future, we expect to use more sophisticated methods to have these point clouds aligned. In our case, the task is complicated by the fact that the camera gives 3D data while the 2D LiDAR gives planar information. Moreover, they do not have overlapping FOV.

Since the chassis is symmetric, the sensors have been mounted in specific positions to ease the calibration task. We imposed a `base_link` point at a symmetric position along the y-axis of the robot (see Figure 4.22). Then, thanks to the regular shapes and 90 degrees angles of the chassis components, we easily measured the distances with a ruler.

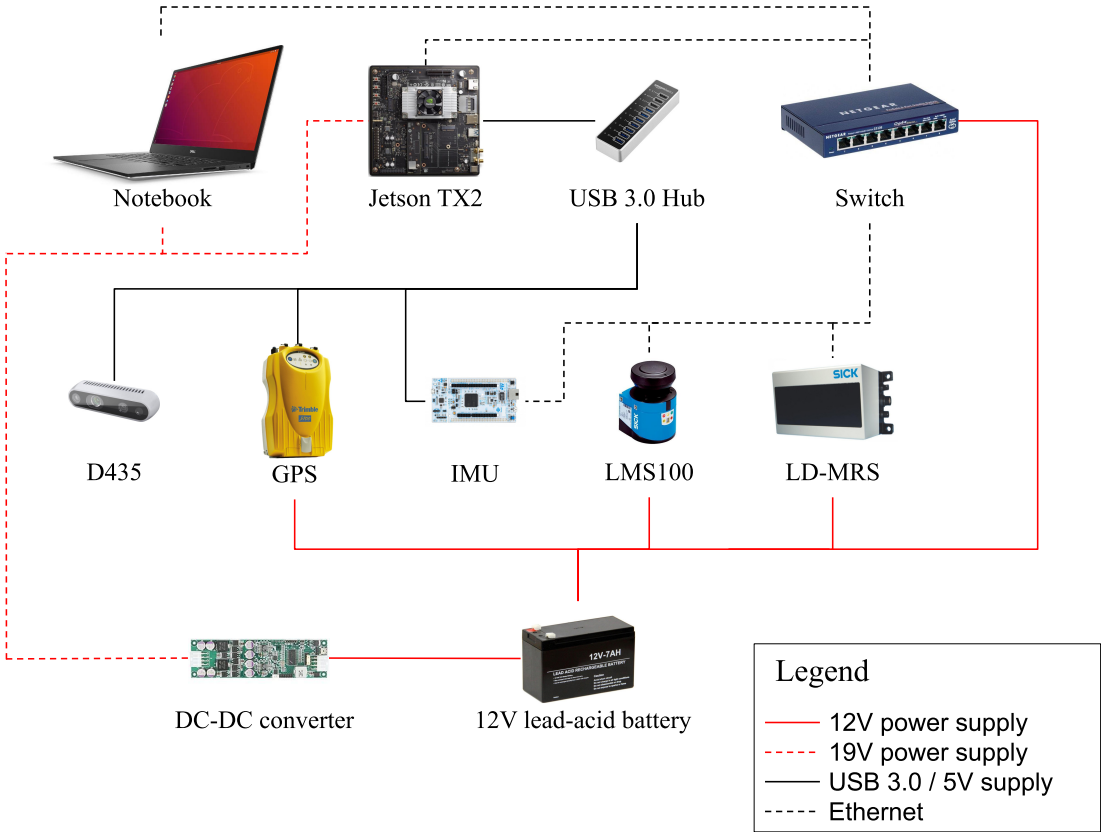


Figure 4.21: Sensors data and power connections diagram.

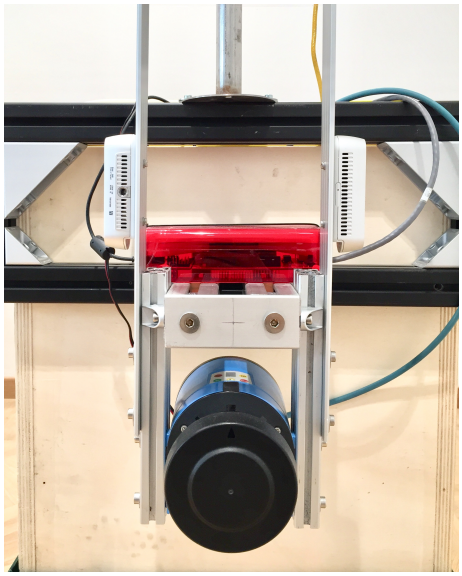
For each sensor, we measured the distances along x,y,z axes between the `base_link` point and a special point on the sensor chassis. The special point could be, for example, a corner point. Then, the distance from this special point and the sensor origin is given by the sensor datasheet. As for the rotations, we made their calculation easy as we placed sensors faces parallel to robot reference system axes. After the measuring process, all translations and rotations are added as transformations in the ROS `tf` tree.

4.3 Setups

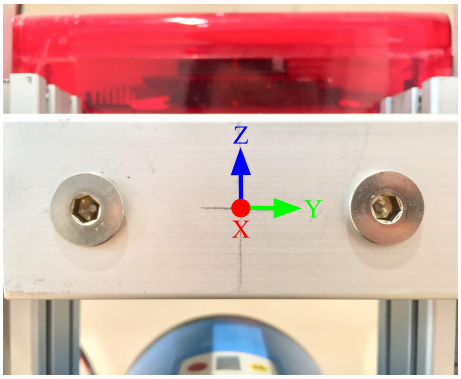
During all the thesis' experimental phase, we modified our sensor setup multiple times as the result of the feedbacks obtained during the experiments. The platform has gone through three main revisions, and we are going to explain them in the following.

First setup In the first setup (see Figure 4.23), we mounted the 2D and the 3D LiDARs, the RealSense D435, the IMU, and the GPS. The D435 camera was mounted laterally with the aim to scan the vine leaves and to be used as a source for the visual odometry algorithm. However, with its limited FOV, the camera is not able to see the bottom part of the plants, that is, the trunks.

The two LiDARs were placed with scanning planes parallel to the ground in order to compute ICP/laser odometry (see Subsection 3.2.1). We also had in mind to understand



(a) Platform front side with the final sensors configuration.



(b) A close view at the `base_link` point. The x-axis points toward the reader as the coordinate system is right-handed.

Figure 4.22: Platform front and `base_link` position.

which sensor was better suited for this application. In indoor environments, ICP odometry demonstrated to be an accurate mean of localization. In outdoor experiments, it has lower accuracy due to the less structured environment. Moreover, it was more subject to odometry losses.

Both indoor and outdoor, laser odometry has been frequently misled (causing heavy drifts) by fast rotations and uneven terrain. The 3D LiDAR was thought to provide more accurate odometry because of the richest information relative to the 2D one. Instead, odometry with 3D four planes LiDAR was more prone to errors. Likely, the four planes laser does not give enough information to balance the ICP computational errors of the additional degree of freedom. For these reasons, we decided to avoid laser odometry and stick to visual odometry.

Second setup In the second setup (see Figure 4.24), we mounted the same sensors of the first setup with the additional D435i camera. The camera D435 remained in the same position and with the same aim. The D435i was placed in a symmetric opposite position to the D435 to scan the other side of the vineyard.

The 2D LiDAR changed both position and purpose. We mounted it at the cart front with the laser plane vertical to the ground. The goal was of scanning the plants from the ground to the top leaves. With this configuration, the spatial resolution of the scans depends on the sensor frequency and the robot moving speed. At 50 Hz and speed of 1 m/s, the resolution (along the moving direction) is of 2 cm.

The 3D LiDAR was still placed horizontal relative to the ground but at a lower height so to detect the trunks. Then, in the post-processing phase, the trunks and poles would be easily found as they are peak points separated by valleys of empty intra-trunks space. However, since the 3D LiDAR has multiple laser planes, as the robot moves, the same area

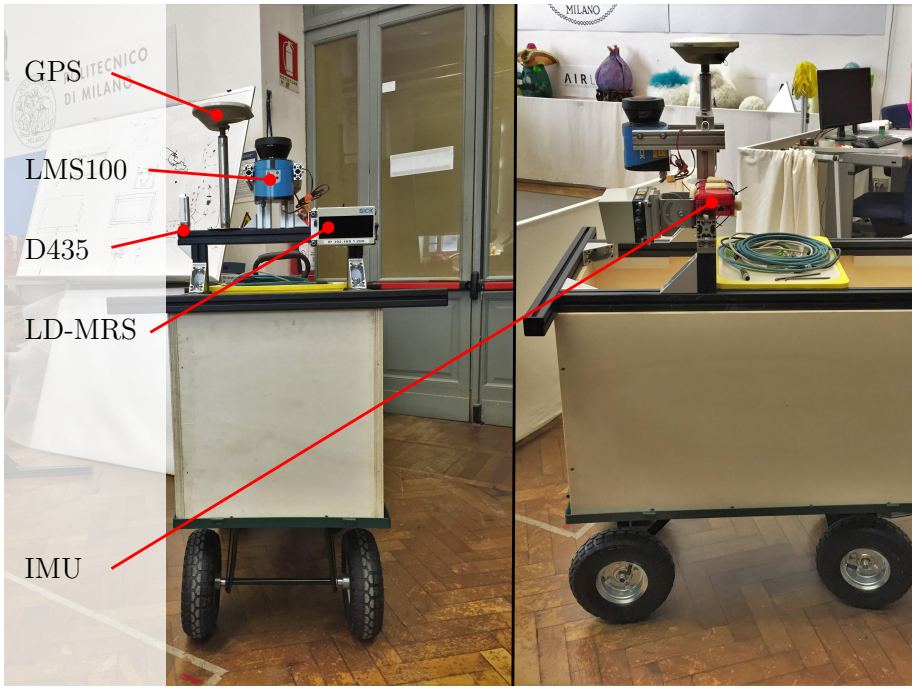


Figure 4.23: Sensors first setup.

is scanned numerous times. Thus, if raw scans would be merged based on odometry information, they will result in many overlapped scans. Indeed, to get meaningful information, the scans should post-processed and matched via, for example, ICP registration.

This extra computational load is not necessary in the case of 2D scans since they are assembled just with odometry information. After visual inspection of the collected data, we noted that the assembled point cloud of 2D scans was of high accuracy and fidelity. For this reason, we decided to give up with the 3D sensor, thus avoiding extra processing load.

Final setup In the third and final setup (see Figure 4.25), we maintained all the previous sensors but the 3D LiDAR. We repositioned the cameras closer to the center of the robot so to gain more view on the plants. This setup is the final configuration: a compact, lightweight, and relatively cheap suite of sensors for phenotyping. The 2D LiDAR is the component that most take part in the total weight and cost of the sensor suite. In the future, cheaper and tinier 2D LiDAR sensors should be inspected.

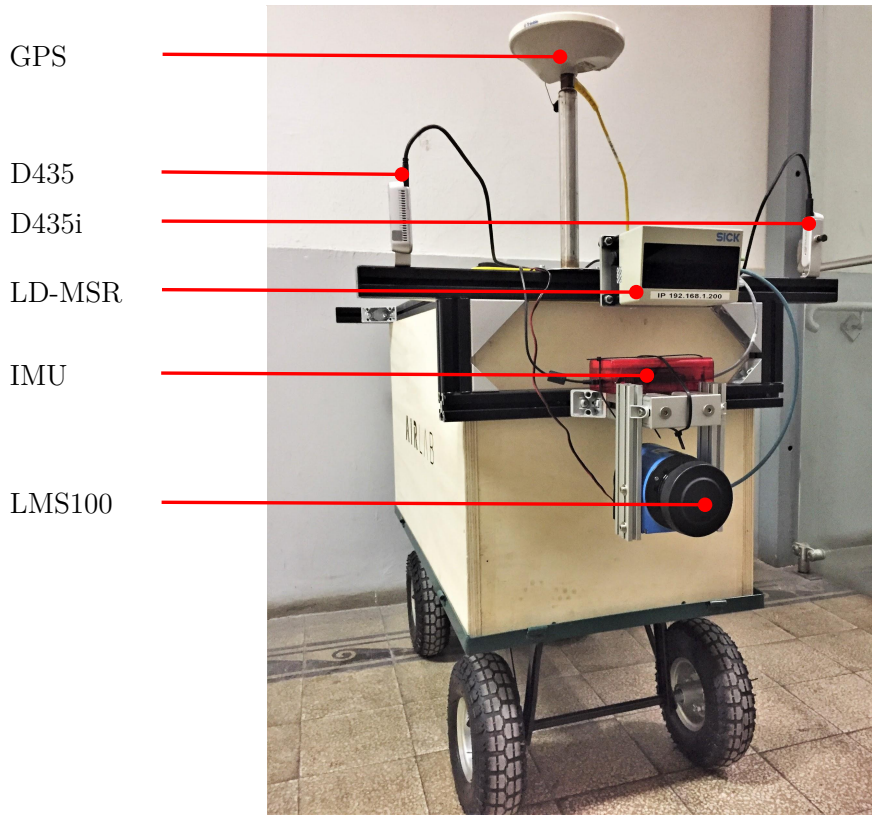


Figure 4.24: Sensors second setup.

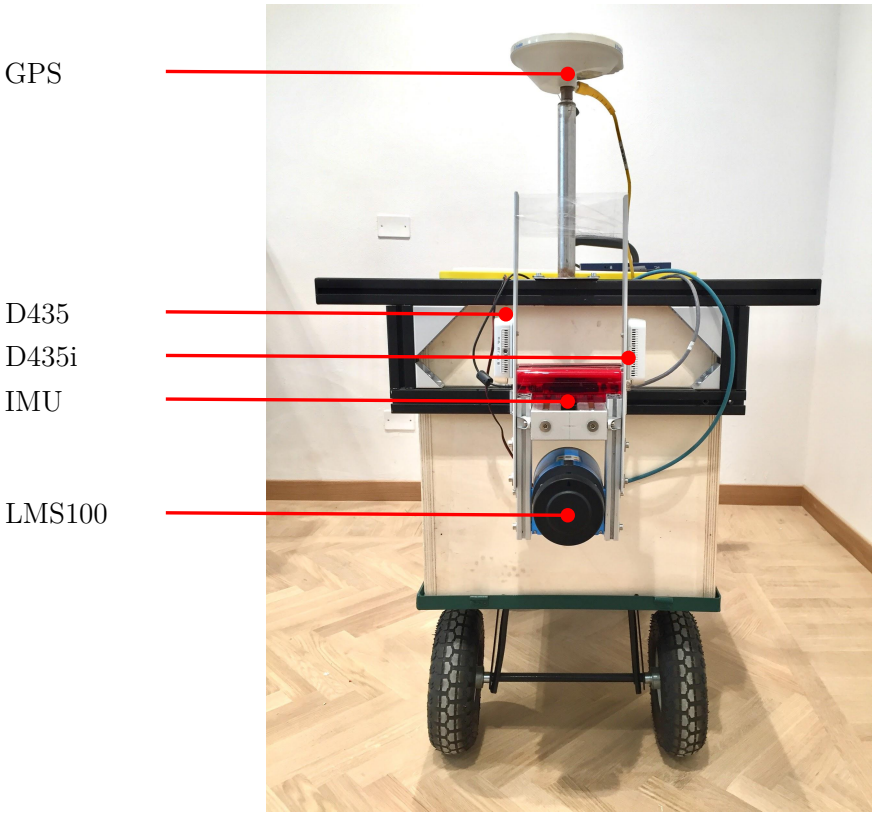


Figure 4.25: Sensors final setup.

Chapter 5

Software Architecture

In this Chapter, we provide in Section 5.1 a general view of the software architecture and its components that will be explained in the following sections. In Section 5.2, we describe the structure of the Reconstruction component responsible for building a 3D model of plants from the sensor data. The Reconstruction component is further divided into three subcomponents: the Odometry, the LiDAR Reconstruction, and the Visual Reconstruction. Finally, in Section 5.3, we explain the segmentation algorithm that isolates single plants from a point cloud.

The repository containing all the code, the user manual, and the installation instructions can be found at `https://github.com/ricber/Plant-Phenotyper.git`.

5.1 General View

From a high-level view, the software architecture can be seen as composed of three elements: the Data Collection, the Reconstruction, and the Segmentation (see Figure 5.1).

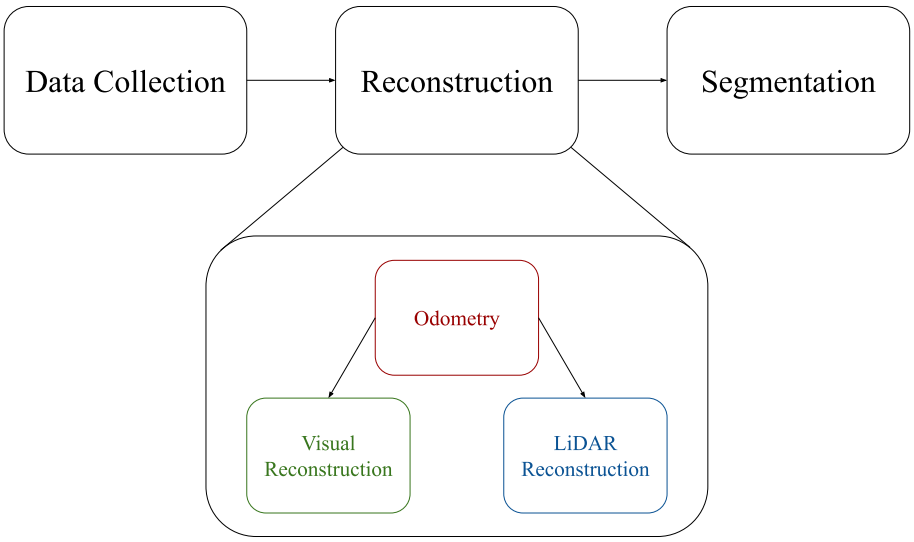


Figure 5.1: Software Architecture components diagram. The same colors will be used in the following diagrams to distinguish between the different subcomponents.

The Data Collection is constituted of all the code for interfacing with the sensors and to record data. This component will not be treated here as it just contains standard ROS code for recording and managing bag files, commonly used ROS drivers, and sensor-specific drivers. They are "plug-and-play" nodes and do not require to go deep in technical details. Documentation of these ROS packages can be easily found on the internet, for further information refer to our GitHub repository at `https://github.com/ricber/Plant-Phenotyper.git`.

The Reconstruction component is, in turn, composed of three subcomponents: Odometry, LiDAR Reconstruction, and Visual Reconstruction. The Odometry is the component characterized by all the code for localization purposes. Laser Reconstruction is the component that builds a tridimensional model with laser data. The Visual Reconstruction does the same but with camera images, thus resulting in a colored tridimensional model.

Finally, the Segmentation component has the purpose of processing the tridimensional laser model to isolate single plants. Reconstruction component has been written in C++ and XML via the ROS framework. The Segmentation component has been written in C++ via the PCL library.

5.2 Reconstruction

The Reconstruction component is composed of three subcomponents: Odometry, LiDAR Reconstruction, and Visual Reconstruction. In this Section, we explain in detail the characteristics of these three subcomponents and their interactions with the Data Collection and Segmentation components.

A key concept to highlight here is the choice of how we make corrections to the odometry. The local odometry is subject to accumulate drift that must be corrected by a global odometry source. The concept of Loop Closure (will be better explained in Subsection 5.2.2) is a fundamental characteristic of every SLAM algorithm that serves precisely for the scope of being a global reference for corrections. However, in agricultural settings, we can not assume to have loop closures. A loop closure happens when a robot returns to an already visited location, thus creating a loop in the graph of a SLAM algorithm.

A vehicle traversing row by row a crop field, will difficulty have a repeated view of the same scene, thus making loop closure detection useless [43]. Moreover, rows can be extremely long, sometimes spanning a thousand meters or more. Thus the accumulated drift would be so heavy making the correction task very hard. For these reasons, we decided to deactivate the loop closure detection of the `rtabmap` algorithm and relying on the GPS as a global reference. Therefore, the `rtabmap` ROS node is used to build a colored 3D reconstruction based on an external source of odometry obtained fusing different sensors. In Subsection 5.2.1, we explain how we did the sensor fusion.

5.2.1 Odometry

The Odometry is responsible for computing the change in position of the robot over time relative to its starting position. This component can exploit different sensor inputs selecting them via a parameter. The possible sensors are visual or LiDAR sensors, IMU, and GPS. Various combinations of these three categories are possible. One between Visual

or LiDAR sensors is mandatory. Then it is possible to add the IMU or the GPS, or both. The possible combinations can be listed with strings of the form: (Visual Odometry|LiDAR Odometry)[_IMU][_GPS].

In the following, we explain the ROS graph architecture in the case of a complete set of sensors, that is, (Visual Odometry|Lidar Odometry)_IMU_GPS. In Figure 5.2 is depicted the Odometry ROS graph where ellipses graphically represent nodes, and rectangles represent topics. The nomenclature is the same used in the code, but convenience names are added to nodes and written in red font. In the following, we describe the functioning of each node. Finally, it will be explained the `tf` tree structure since it is widely affected by Odometry updates.

Visual Odometry The concept of Visual Odometry (VO) has already been explained in Section 3.2. The odometry is computed by the `rtabmap` nodes `rgbd_odometry` or `stereo_odometry`. As the name suggests, these nodes are for RGB-D cameras and stereo cameras, respectively. These nodes take as input the last produced RGB image, the relative depth image, and the `tf` tree. In our case, the node takes the images from the D435 camera and outputs an odometry topic named `odom_rgbd`. The last produced camera image frame is the starting point of the following steps (see Figure 5.3):

1. Feature Detection: features are salient points of an image that can also be recognized if the same scene is captured from a different perspective. Such points are like corners. The algorithm used for feature detection is Good Features To Track (GFTT) [44]. As we want to match two images with an overlapped view, the features need a descriptor that allows to identify them. The descriptor used by this node is BRIEF. Moreover, for RGB-D images, the depth image is used as a mask for GFTT to avoid extracting features with invalid depth. For stereo images, stereo correspondences are computed by optical flow to determine the depth of the detected features. The extracted features of the input image with their descriptors are passed to the next stage.

2. Feature Matching: features from the preceding stage are matched to the last Key Frame (F2F) or the Feature Map (F2M). The Feature Map contains 3D features from last Key Frames. How features become part of a Key Frame will be explained later. All the matched couples of features are passed to the next stage.

3. Motion Estimation: knowing the 3D positions of the last Key Frame or Feature Map features, it is possible to estimate the motion of the camera through the current frame matched features. The current features can be coupled with their respective 3D points by assuming that the environment is static. With these couples of 3D points and their respective 2D projected image points, the camera position transformation is estimated through the Perspective-n-Point (PnP) algorithm. In order to make the estimation more resilient to outliers, a RANSAC version of PnP is utilized.

4. Motion Prediction: this component computes a prediction of the new camera position based on a motion model given as input the current velocity. This is used to predict the position of the Key Frame or Feature Map features in the current frame. This limits the search window for Feature Matching to provide better matches.

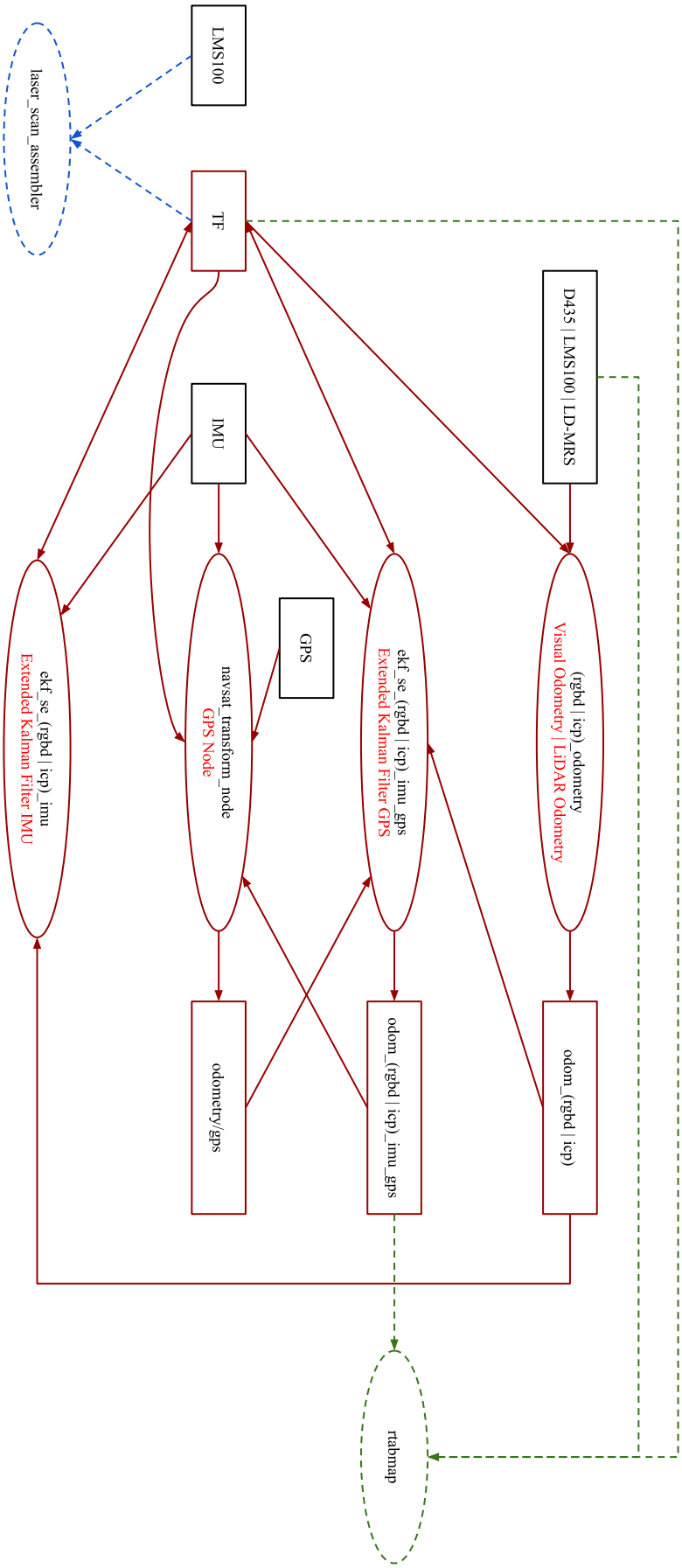


Figure 5.2: Odometry ROS nodes graph. Colors are relative to those used in Figure 5.1

5. Local Bundle Adjustment: the estimated transformation is refined using local bundle adjustment [45] on features of all Key Frames in the Feature Map (F2M) or only those of the last Key Frame (F2F).

6. Pose Update: with the last camera pose and the transformation, the new camera pose is computed. The velocity is computed and given as input of the Motion Prediction code. Moreover, an odometry message is published and the `tf` tree edge `/odom->base_link` is updated.

7. Key Frame and Feature Map update: the features coming from the Update Pose component are used as the update of the Feature Map or Key Frame if the number of inliers computed during Motion Estimation has fallen under a certain threshold. For F2F, the Key Frame is just replaced by the current image frame. For F2M, the Feature Map is updated by adding the unmatched features of the new frame and updating the position of matched features that were refined by the Local Bundle Adjustment module.

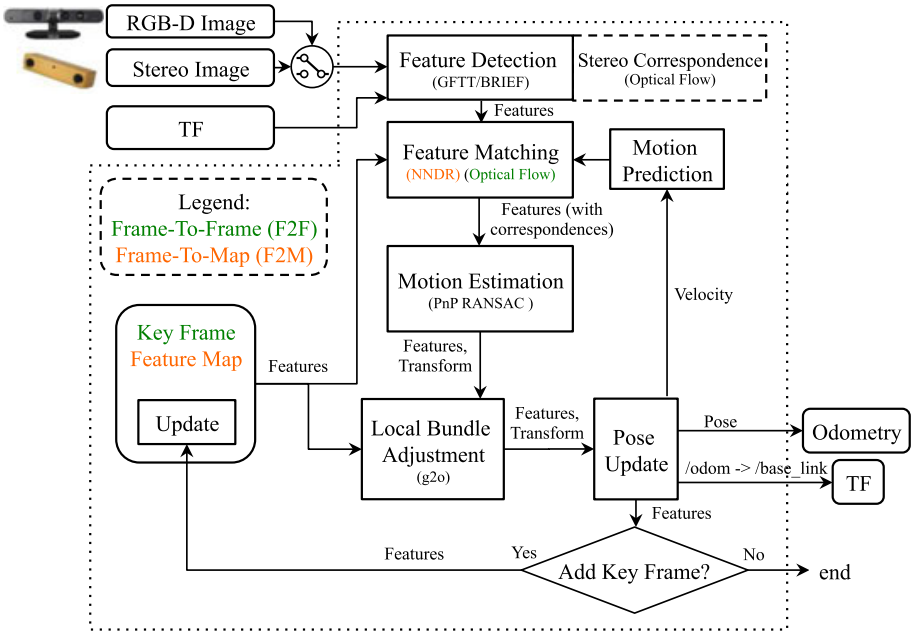


Figure 5.3: RGB-D `rtabmap` odometry node.

LiDAR Odometry The concept of Laser Odometry (LO) has already been explained in Section 3.2. The odometry is computed by the `rtabmap` node `icp_odometry`. As the name suggests, this node uses the ICP algorithm to find the transformation between successive laser scans. The node takes as input the last produced sensor scan, and the `tf` tree. The sensor could be both a 2D LiDAR, thus producing a so-called Laser Scan in ROS terminology, or a 3D LiDAR, thus producing a so-called Point Cloud message. The sensor scan is the starting point of the following steps (see Figure 5.4):

1. Point Cloud Filtering: the input point cloud is downsampled, and normals are computed. Downsampling allows to reduce the computational effort, and normals are used as additional information in the following stage.

2. ICP Registration: the current point cloud is matched with the ICP algorithm to the last Key Frame (S2S way) or to the Point Cloud Map (S2M way). The ICP version (see [46] for an analysis of ICP variants) used by `rtabmap` exploits normals to ease the matching task.

3. Motion Prediction: this component gives a hint to the ICP Registration algorithm about the current position of the robot. The prediction is based on the velocity computed from a constant velocity motion model, or on an external odometry source like wheels odometry. In this last case, the pose update is a correction of the eternal source of odometry.

   > *Using an external initial guess can help estimate the motion in the direction in which the environment is lacking features. For example, a robot with a short-range lidar moving in a long corridor in which there are no doors (i.e., not distinguishable geometry) would only see two parallel lines. If the robot accelerates or decelerates in the direction of the corridor, ICP would be able to correct orientation but it would not be able to detect any changes in velocity in the direction of the corridor. In such case, using external odometry can help estimate velocity in the direction in which ICP cannot.* [29]

4. Pose Update: after calculating the transform via ICP, the new pose is computed based on the last robot pose. The `tf` tree is updated, and an odometry message is given as output.

5. Key Frame and Point Cloud Map Update: if the ICP correspondence ratio goes under a fixed threshold, the Key Frame or Feature Map is updated. In the case of S2S, the current frame becomes the new Key Frame. In the case of S2M, the current map is subtracted from the new point cloud, then the remaining points are added to the Point Cloud Map.

GPS node The GPS signal provides absolute positioning information relative to an earth reference system. Thus, it is used to correct the drift of a local odometry source. As already explained in Subsection 3.1.1, the local odometry must be continuous while the long-term one can present discreet jumps. Thus, the GPS signal is perfect for building the correction odometry transformation `map->odom` to the local odometry `odom->base_link`. However, GPS data is provided in a format (latitude, longitude) that prevents direct integration as odometry source. Indeed, a translation from the earth frame to the robot's world frame is necessary. The node responsible for this translation is the `navsat_transform_node` of the `robot_localization` ROS package (see Subsection 3.2.2).

The node takes as input IMU (with magnetometer), odometry, and GPS messages. The IMU serves to know the orientation relative to the true north. As the magnetic north is different from the true north of a certain degree of angle, the node gives the possibility to specify the magnetic declination by a parameter. The odometry must be relative to a frame placed in the starting position of the robot. In our case, this frame is `map`. The

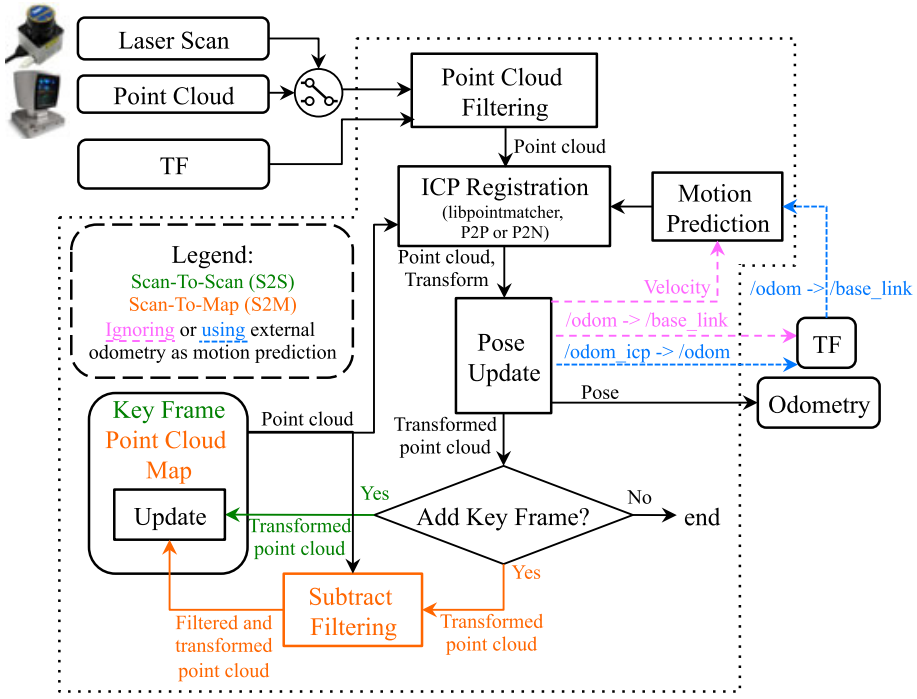


Figure 5.4: LiDAR `rtabmap` odometry node.

odometry source serves to know the robot position in case the first GPS message arrives after the robot has started operations. The odometry is read from the `tf` tree. The node output an odometry topic named `odometry/gps`.

Extended Kalman Filter The Extended Kalman Filter is implemented by nodes from the `robot_localization` ROS package (see Subsection 3.2.2). It fuses all input information into single output odometry with a Kalman Filter with the aim of state estimation. It is possible to choose between Extended or Unscented Kalman Filter. Each input should have its covariance matrix to weight the sensor information properly. Different inputs are possible: odometry sources (pose and twist messages), twist messages (angular velocity), pose messages (position and orientation), or IMU messages.

Our architecture contains two EKF nodes: `ekf_se_(rgbd|icp)_imu_gps` (informally, Extended Kalman Filter GPS), and `ekf_se_(rgbd|icp)_imu` (informally, Extended Kalman Filter IMU). As we are operating in a planar environment, we set the `two_d_mode` parameter to `true`. The EKF node allows to specify, for each type of input, a 15-dimensional tuple with boolean values. Each element of the tuple corresponds to an estimated variable of the robot state (as explained in Subsection 3.2.2). Thus, it is possible to deactivate some variables setting them to `false`.

The `ekf_se_(rgbd|icp)_imu` fuses the Visual or LiDAR Odometry with the IMU data. The IMU is fused with $\dot{yaw}, \ddot{yaw}$, and $\ddot{X}$. As the robot is moving on a fairly flat surface, we do not consider *roll*, *pitch*, Z, and their respective velocities and Z acceleration. Moreover, as our robot is nonholonomic, we exclude also the variables relative to Y. Instead, the Visual or LiDAR odometry is fused using $\ddot{X}$ and $\ddot{yaw}$. The position, that is, variables X, Y, and Z, is not fused since it would be redundant information as it is computed from the same source of odometry. Finally, the produced odometry is written

on the `tf` tree being the transform `odom->base_link`.

Instead, the node `ekf_se_(rgbd|icp)_imu_gps`, informally called Extended Kalman Filter GPS, fuses the odometry source with the IMU and the GPS. The odometry and the IMU are fused in the same way as the just described node. The `odometry/gps` topic from `navsat_transform_node` is fused with X and Y variables since the GPS provides absolute position information. The Z variable is not fused as we assume to move on a planar surface. Finally, the produced odometry is given in output with the `odom_(rgbd|icp)_imu_gps` topic. The `rtabmap` node, the one responsible for Visual Reconstruction, subscribes to this topic.

`tf` tree

The `tf` tree (see Subsection 3.1.1) incorporates odometry e sensors transforms. Our `tf` tree is depicted in Figure 5.5. At the top of the tree, there is the optional `utm->map` transform. This transform is calculated by the `navsat_transform_node` if the `broadcast_utm_transform` parameter is set to `true`. The transform is helpful if we want the data to be georeferenced.

Going down the tree, the successive transform is the `map->odom`. This transform is published by the EKF GPS node that fuses the Visual or LiDAR odometry with the IMU and the GPS. In case we do not want to rely on the GPS to compute long-term odometry, the `rtabmap` node could replace this transform through loop closure corrections (see Subsection 5.2.2). Finally, the IMU could be excluded from EKF sensors fusion by a parameter.

The next transform is the `odom->base_link`. It is published by the EKF IMU node that fuses the Visual or LiDAR odometry optionally with the IMU. In case the IMU is not fused, the broadcaster of the transform is just the `(rgbd|icp)_odometry` node.

After `base_link` frame, the tree forks into the various sensors transforms. Differently from the upper transforms that are dynamic, these last transforms are static. Indeed, sensors should not change the position relative to the `base_link` during operation, since they are rigidly fixed to the chassis.

5.2.2 Visual Reconstruction

The Visual Reconstruction subcomponent builds a 3D colored model of the plants exploiting camera RGB and depth images. The Visual Reconstruction is performed by the `rtabmap` node. Here we describe the general structure of the node. In the following, we refer to the block diagram of Figure 5.6.

The inputs of the node are camera images, the already mentioned `tf` tree, a source of odometry and the laser scans. The camera input is mandatory, with the option to choose from an RGB-D camera or a stereo one. Instead, the laser input is optional. It is possible to feed both a LaserScan message from a 2D LiDAR or a PointCloud from a 3D LiDAR. The odometry could be of whatever kind, like a fusion of different sources of odometry (e.g., from a `robot_localization` node).

All the input data are synchronized via the timestamp. The timestamp is a field, part of the header of every ROS message, that tells the time when the message has been produced. The synchronized information is passed to the Short Term Memory (STM)

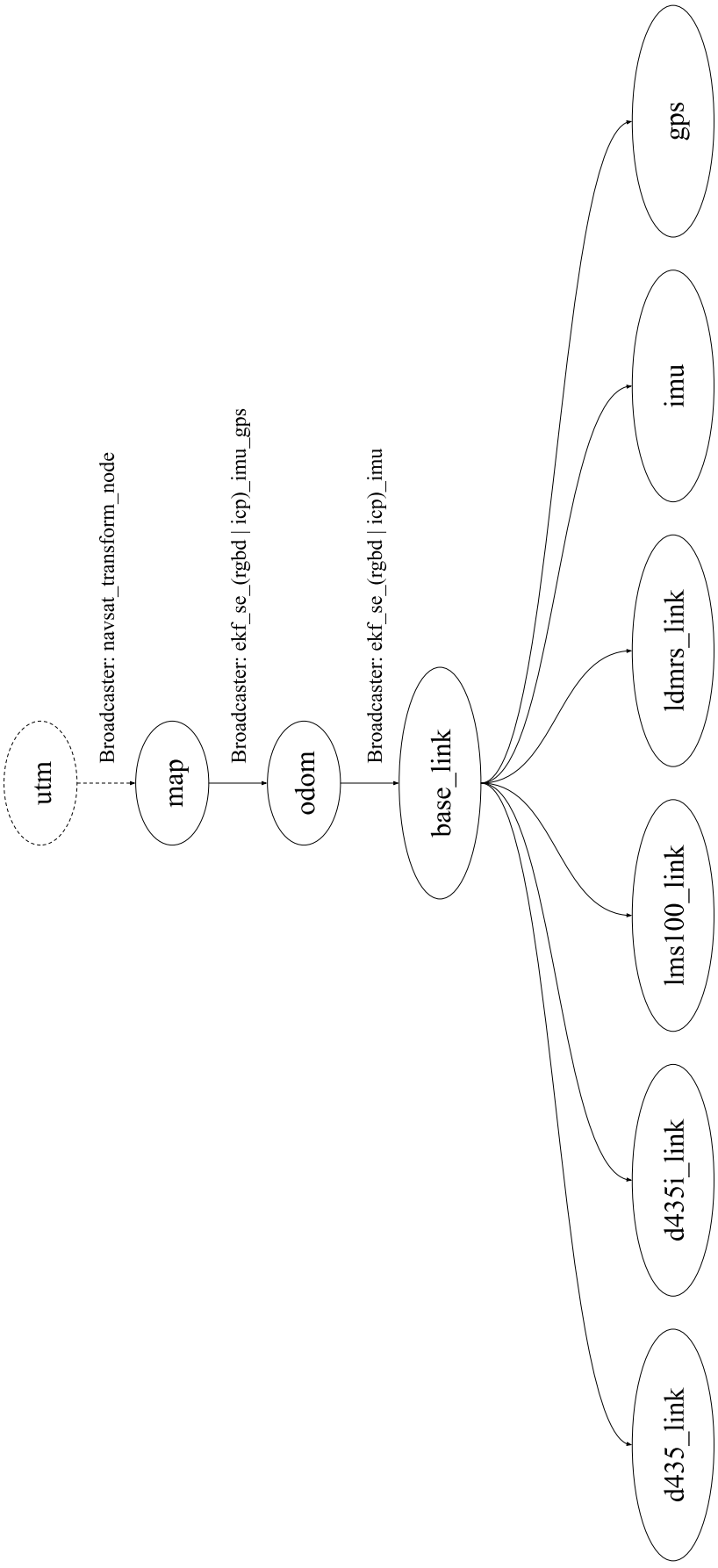


Figure 5.5: tf tree of our system.

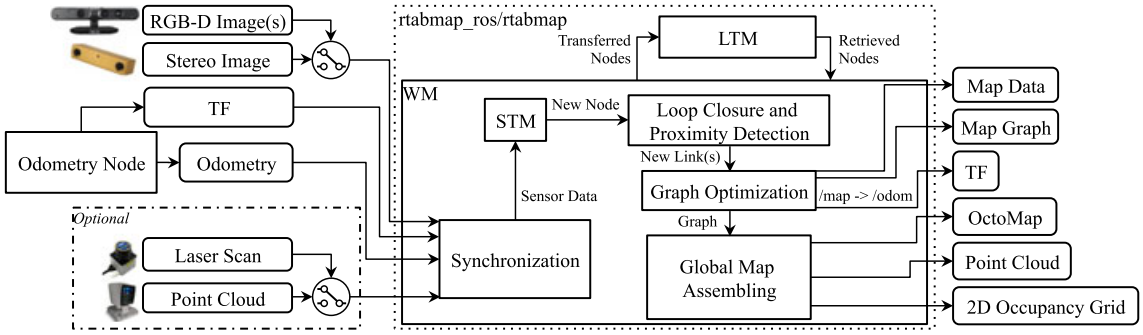


Figure 5.6: Block diagram of `rtabmap` ROS node. [29]

and here, a new graph node is created. Indeed, since `rtabmap` is a graph-based SLAM algorithm, the map is represented by a graph with nodes and links. Each node contains the odometry pose, sensor's raw data, and additional information useful for next modules (e.g., visual words for Loop Closure and Proximity Detection, and local occupancy grid for Global Map Assembling). A link contains a rigid transformation between two nodes. There are three kinds of links: Neighbor, Loop Closure and Proximity links. Neighbor links are added in the STM between consecutive nodes with odometry transformation.

The Loop Closure is an essential characteristic of all SLAM algorithms. The Loop Closure happens when a new node corresponds to an already visited location. Indeed, each newly created node is checked against the nodes in the Working Memory (WM) to search for a Loop Closure. In case a match is found, the algorithm adds a link, thus creating a loop in the graph. If it has passed a long time since the robot last visited the same location, the odometry has likely accumulated errors. Once a loop closure is found, the graph is optimized propagating the corrections to all the links.

The Proximity Detection adds links between nodes that should be close to each other. For example, when the robot traverses back a corridor with a camera that has a lateral view, a Loop Closure can not be detected since there are no overlapping views of the same scene. However, Proximity constraints are added.

After graph optimization, the new map is assembled by the Global Map Assembling module. The global map is assembled with all the local occupancy grids associated with each node. Voxel grid filtering is done to merge overlapping surfaces. Once the map is assembled, it is given as output with different possible types to choose: PointCloud, OctoMap (3D occupancy grid), or 2D Occupancy grid. The correction of graph optimization could be optionally reflected on the ROS `tf` tree. Specifically, on the transform that corrects the robot position relative to the `/map` frame. Finally, note that the nodes are transferring back and forth to the LTM, as explained in Subsection 3.2.3.

In our case, the `rtabmap` node reads the global odometry from the topic `odom_-(rgbd|icp)_imu_gps` as represented in the diagram of Figure 5.2.

5.2.3 LiDAR Reconstruction

The LiDAR reconstruction subcomponent builds a point cloud of the scanned plants via laser data. It exploits the `laser_scan_assembler` node of the `laser_assembler` ROS

package [1], and a ROS node written by us. The node block diagram is showed in Figure 5.7.

The node takes as input laser scans and the `tf` tree for the odometry information. The input is processed by the Projector module, which projects the scan into Cartesian space. The projection is passed to the Transformer module that transforms the scan from its reference frame (e.g., `lms100_link` frame), to a fixed reference frame, in our case `map`. Then the scan is converted to a PointCloud2 ROS message and put in a rolling buffer of dimension n. When the service `assemble_scans2` is called, all the clouds in the rolling buffer in a specified interval of time are assembled, and the resulting point cloud is given as output.

Our ROS node, named `assemble_scans_node`, calls the `assemble_scans2` service at a certain frequency (0.25 Hz). The received point cloud is converted to a more convenient type for further processing, that is, from `PointCloud2` ROS type to PCL `pcl::PointCloud<pcl::PointXYZI>` type. After the conversion, the point cloud is saved in a .pcd file.

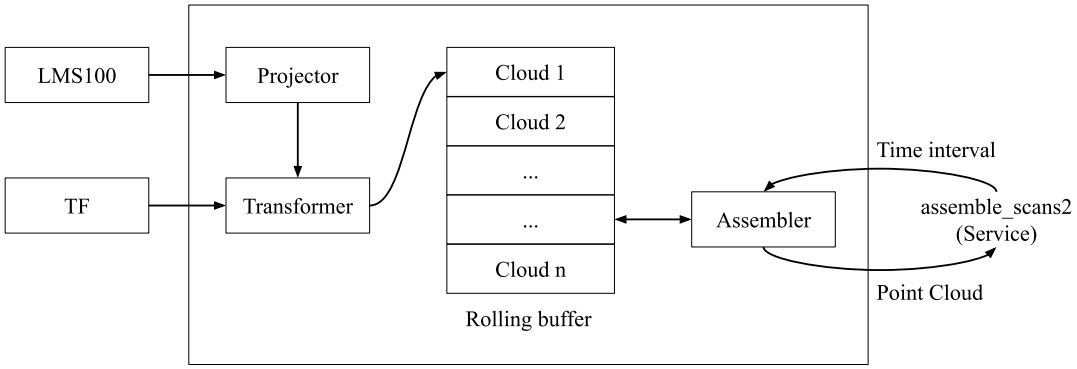


Figure 5.7: Block diagram of `laser_scan_assembler` ROS node.

5.3 Segmentation

The segmentation component has the purpose of isolating single plants from the entire point cloud produced by the LiDAR Reconstruction component. The code has been written in C++ and based on the PCL library. It is constituted by a pipeline that is graphically represented in Figure 5.8 and explained step by step in the following. The input point cloud is showed in Figure 5.9.

1. Affine Transformation (Figure 5.10): the input point cloud named `cloud` is rotated and translated via an affine transformation. An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially, still lie on a line after transformation) and ratios of distances. This first step is made necessary as in the second sensor setup (see Section 4.3), the 3D LiDAR was covering part of the scanning plane of the 2D LiDAR, placed under it. Therefore, a trace remained in the central portion of the vineyard row point cloud.

[1] `http://wiki.ros.org/laser_assembler`

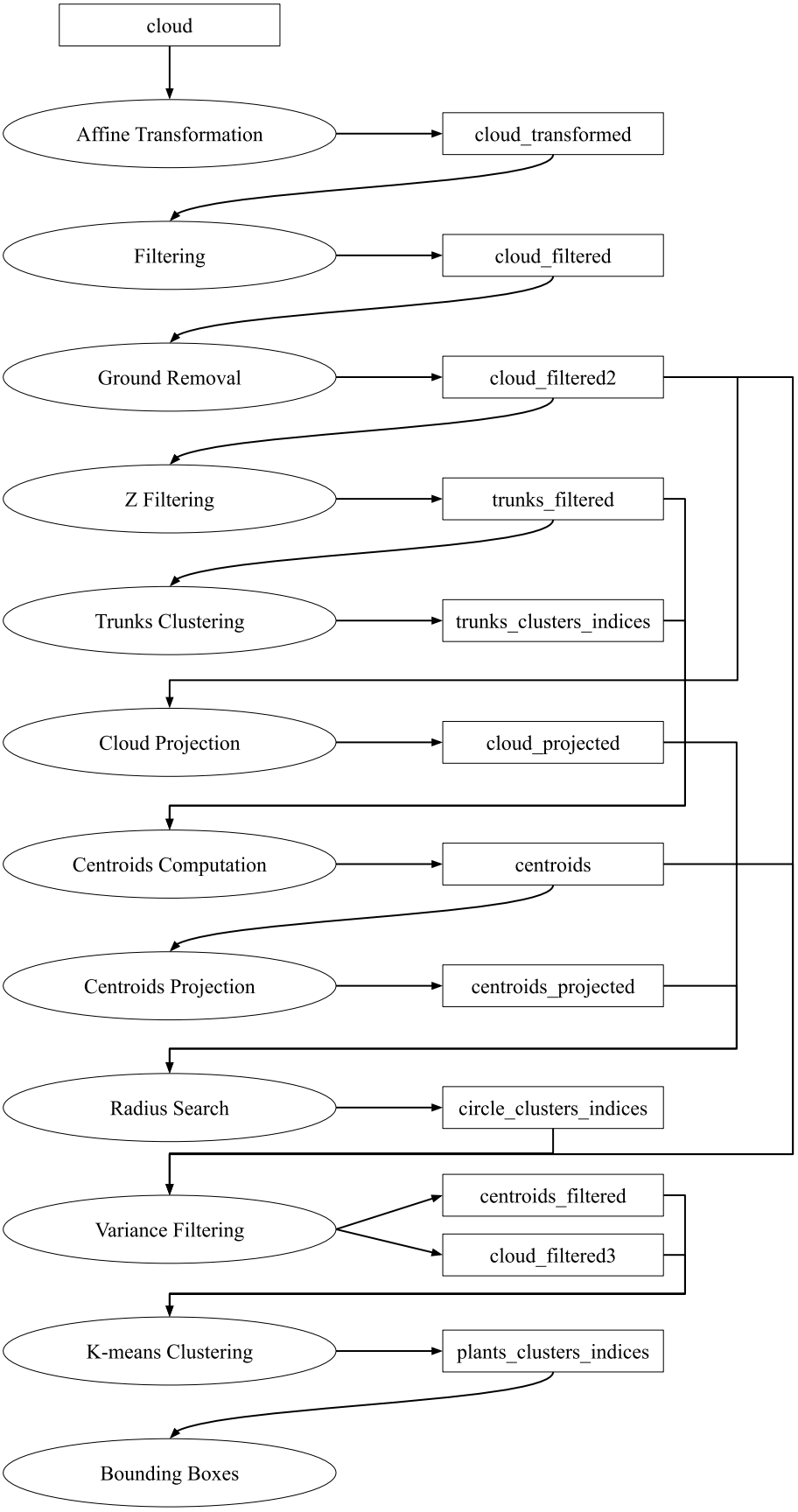


Figure 5.8: Segmentation algorithm block diagram. Ellipses represent code modules. Rectangles represent data structures.

In order to remove it, we filter out all points that have their y coordinate in a specific range (see Figure 5.11). This filtering is possible because the vineyard row sides can be seen as parallel planes, and they can be made perpendicular to the y-z plane of a certain reference system. As the point cloud is referenced to the frame of the robot starting point, it is transformed in order to be referenced to a convenient reference system. The x-axis of the new reference system is parallel to the longest dimension of the vineyard. The z-axis points upward, and the y-axis is placed in the direction to make the reference system right-handed. The transformation is manually tuned, but it can be set automatically by fitting planes to the vineyard sides and placing the conventional reference system with two axes on the middle plane of those two.

2. Filtering (Figure 5.12): the point cloud `cloud_transformed` is filtered along the x and y axes to remove the central LiDAR trace, the points before and after the vineyard row, and the points with low intensity. Finally, statistical outliers removal is applied.

   > *Our sparse outlier removal is based on the computation of the distribution of point to neighbors distances in the input dataset. For each point, we compute the mean distance from it to all its neighbors. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset.*[2]

3. Ground Removal (Figure 5.13): the ground is removed from the `cloud_filtered` point cloud by fitting a plane with RANSAC (see Subsection 3.3.1). We have tuned the parameter `DistanceThreshold`, which determines how close a point must be to the model in order to be considered an inlier. This fixed value is reasonable to stay the same for different plantations and successive scans if the ground grass is of the same height.

4. Z Filtering (Figure 5.14): the point cloud `cloud_filtered2` is sectioned at certain heights along the z-axis to isolate trunks and poles. The lower bound of the section is at the level of the ground. Indeed, lower parts of the plants are usually not covered by leaves. Lower and upper bounds of the section are manually tuned. These fixed values should be reasonable even for different plantations.

5. Trunks Clustering (Figure 5.15): the `trunks_filtered` point cloud is clustered to isolate single trunks and poles. It is used the Euclidean Cluster Extraction algorithm explained in Subsection 3.3.1. We set the `ClusterTolerance` parameter to 0.07m, which is the radius threshold d_{th} of the algorithm. The `MinClusterSize` and `MaxClusterSize` parameters specifies the minimum and maximum number of points that a cluster can have. In our case, we set them to 100 and 10000, respectively. These numbers depend on the size of trunks and poles, and they could change for different plantations.

[2] `http://pointclouds.org/documentation/tutorials/statistical_outlier.php`

6. Cloud Projection (Figure 5.16): the filtered plants point cloud `cloud_filtered2` is projected on the plane found by RANSAC in the Ground Removal stage.

7. Centroids Computation: the `trunks_filtered` point cloud together with the `trunks_clusters_indices` is used to compute the centroid of each trunk/pole. In our specific case, the centroid is a point denoting the geometric center of a set of 3D points represented by their (x, y, z) coordinates.

8. Centroids Projection (Figure 5.17): the `centroids` point cloud is projected in the same way as in stage Cloud Projection.

9. Radius Search (Figure 5.18): the goal of this module is to search the `cloud_projected` point cloud to find, for each projected centroid, all the points with Euclidean distance under a certain radius threshold. The points inside a circle which center is a centroid, are part of a circular cluster. The search is sped up with the help of a Kd-Tree. The radius has been tuned and fixed at 0.22 m. It depends on the size of trunks and poles; thus, it could be different for other plantations.

10. Variance Filtering (Figure 5.19): the goal of this module is to compute the variance of each circular cluster found in the preceding stage, so to filter out poles. Indeed, poles should have their respective circle with a lower variance compared to those of trunks. This can be understood thinking about the projections of the plants point cloud in correspondence of trunks and poles. When projecting a pole, the resulting planar shape is a small circle. When projecting in correspondence of a trunk, the upper canopy of the plant leads to a big semi-circular shape. Thus, computing the variance of points around each centroid, poles should have lower variance and can be filtered fixing a threshold. We manually tuned the `circvar_thr` threshold to 0.0115. This threshold is dependent on the size of the poles.

11. K-means Clustering (Figure 5.20): the filtered point cloud `cloud_filtered3` is clustered via K-means algorithm (see Subsection 3.3.1) to isolate single plants. The K-means algorithm takes as input, beyond the point cloud to cluster, the set of initial centroids `centroids_filtered`. These centroids are the ones remaining after eliminating those related to the poles.

12. Bounding Boxes (Figure 5.21): the goal of this module is to surround the clustered plants with bounding boxes. Bounding boxes are wire framed cubes, that is, cube structures with empty space inside. The preceding stage gives, as a result, a vector of `plants_clusters_indices` to index the points of each clustered plant. Bounding boxes are placed around plants taking into account the extreme points, for each cluster, along each direction of the reference frame system.

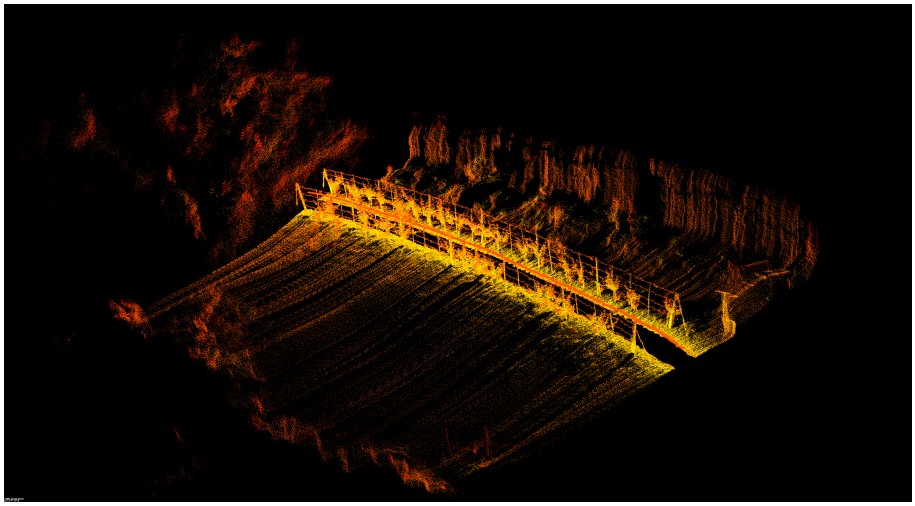


Figure 5.9: Segmentation Algorithm - Input Point Cloud: `cloud` point cloud colored based on intensity channel. Brighter/yellow points have high-intensity; red/darker points have low-intensity.

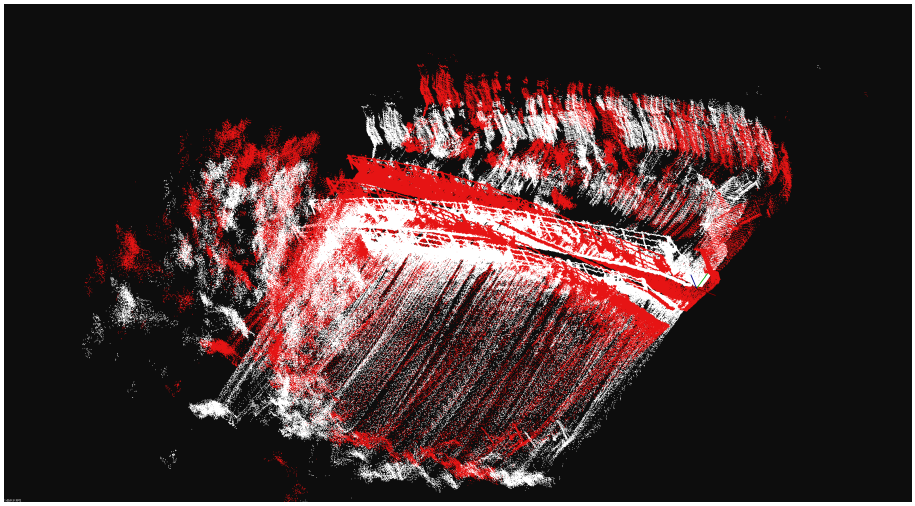


Figure 5.10: Segmentation Algorithm - Affine Transformation: `cloud_transformed` in red and the original `cloud` in white.

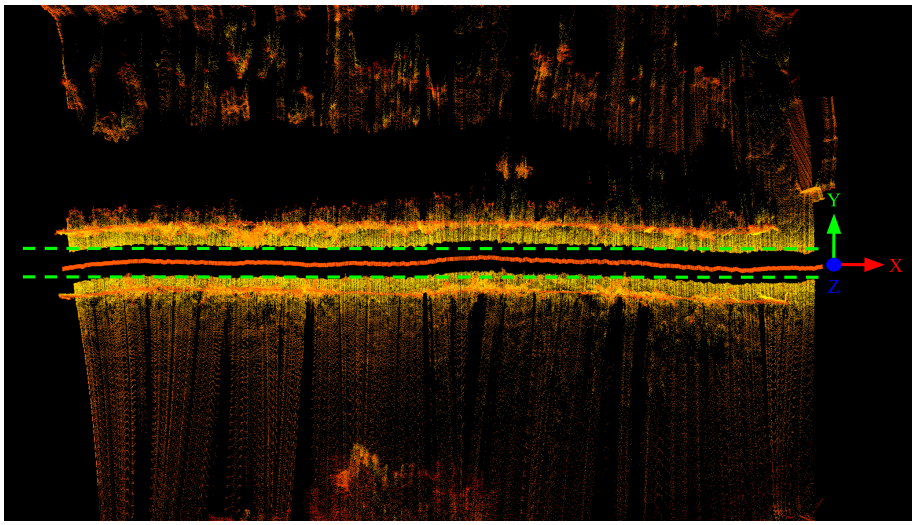


Figure 5.11: Here we show how the central trace filtering works. After fixing a convenience reference system that has the y-axis perpendicular to the vineyard row sides, we filter out all points that have their y coordinate comprised in a certain range. This range is the one between the green dashed lines.

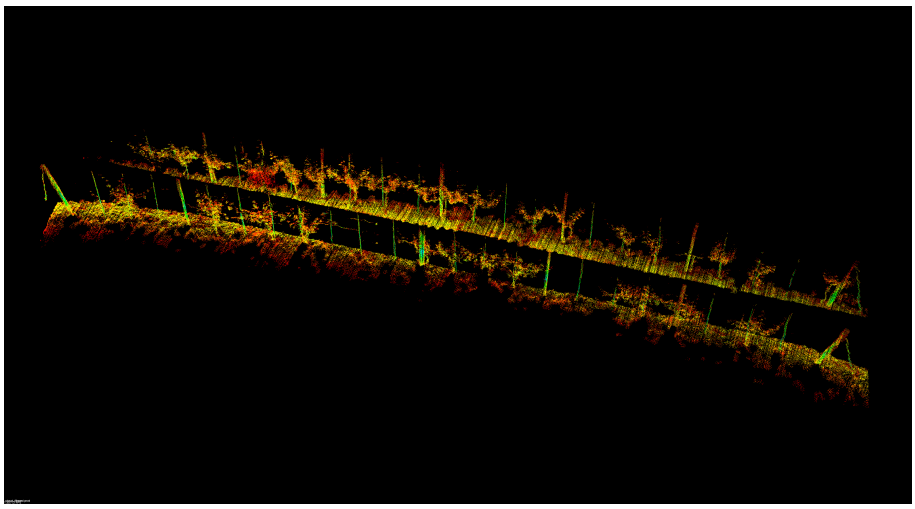


Figure 5.12: Segmentation Algorithm: `cloud` point cloud.

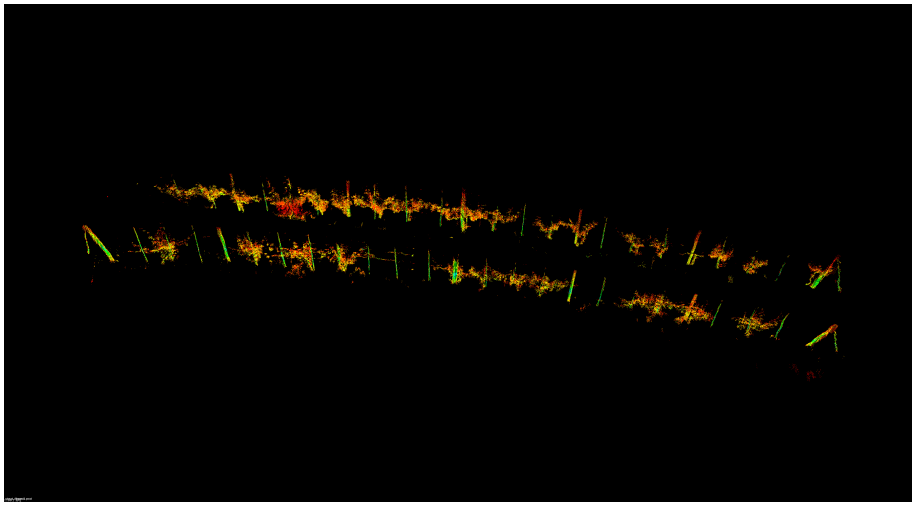


Figure 5.13: Segmentation Algorithm - Ground Removal: `cloud_filtered2` point cloud.

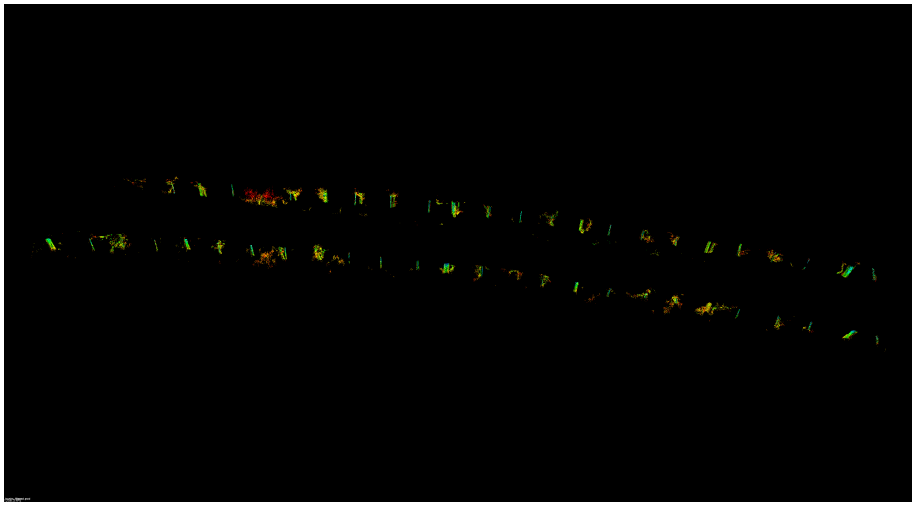


Figure 5.14: Segmentation Algorithm - Z Filtering: `trunks_filtered` point cloud.

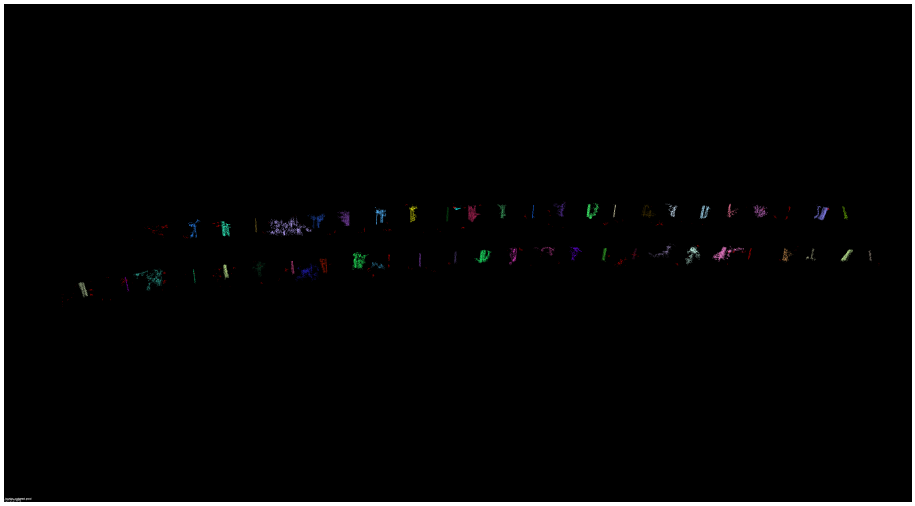


Figure 5.15: Segmentation Algorithm - Trunks Clustering: trunks and poles clusters are shown in different colors. Nonclustered points are shown in red.

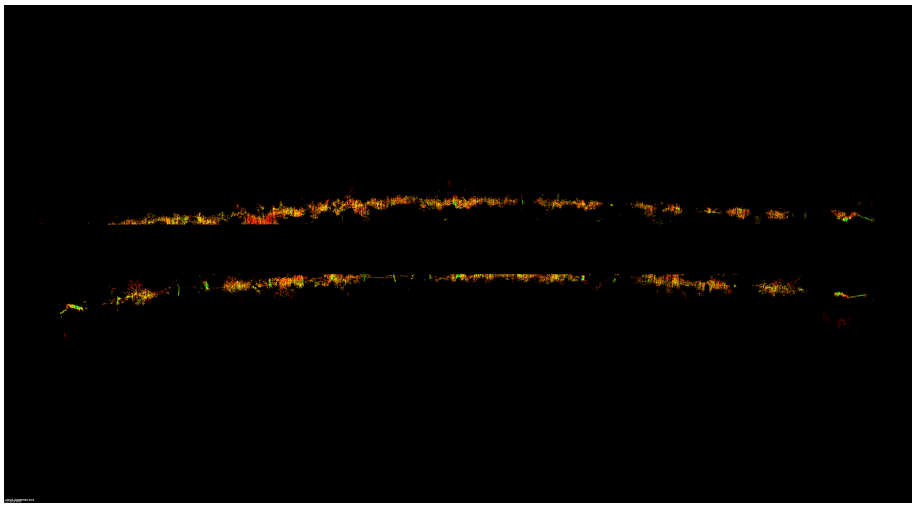


Figure 5.16: Segmentation Algorithm - Cloud Projection: `cloud_projected` point cloud.

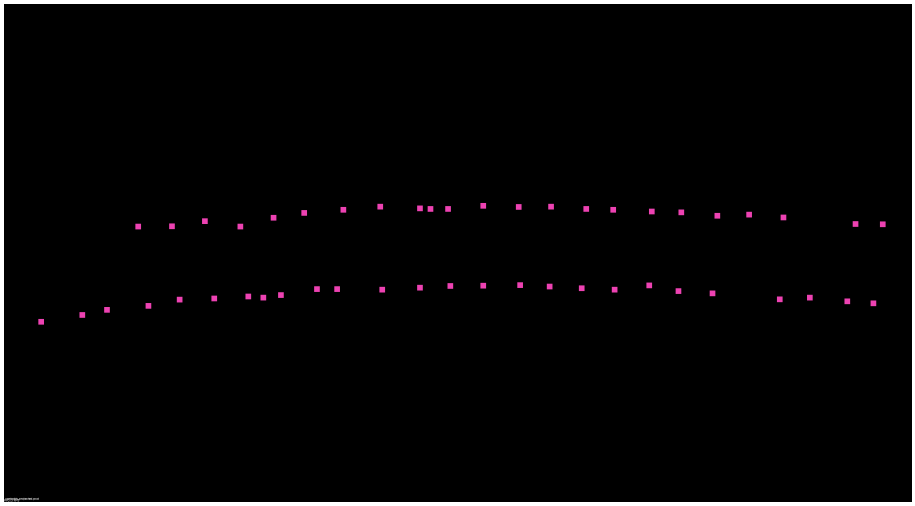


Figure 5.17: Segmentation Algorithm - Centroids Projection: `centroids_projected` point cloud. Points are enlarged in size for ease of visualization.

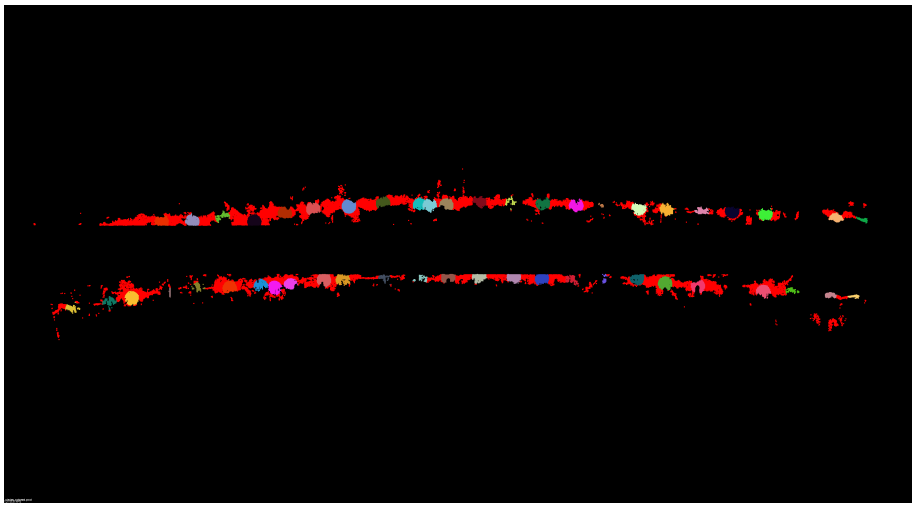


Figure 5.18: Segmentation Algorithm - Radius Search: circle cluster are represented in different colors. Nonclustered points are shown in red.

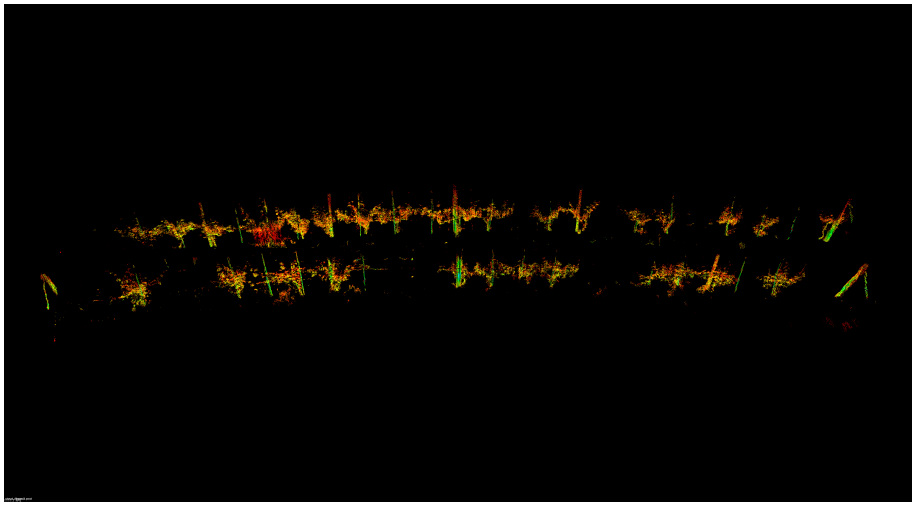


Figure 5.19: Segmentation Algorithm - Variance Filtering: `cloud_filtered3` point cloud. Note that poles have been filtered out.

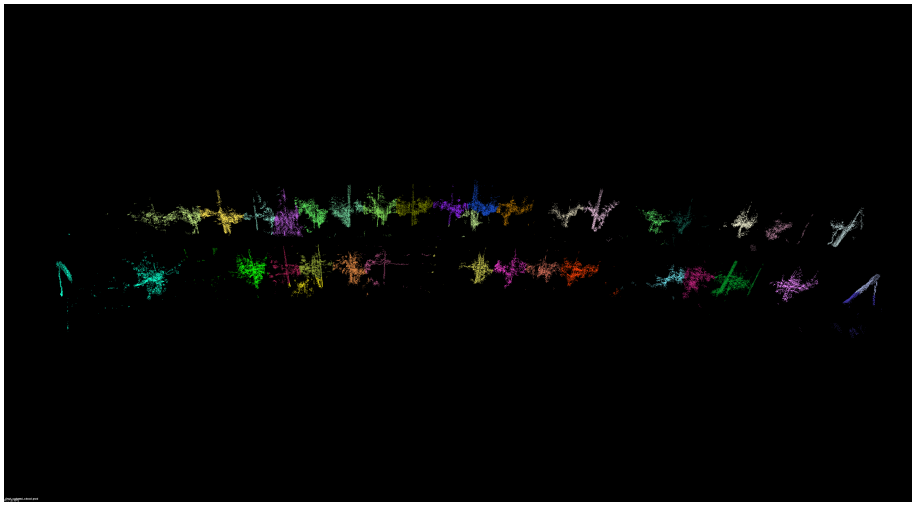


Figure 5.20: Segmentation Algorithm - K-means Clustering: plants clusters are shown in different colors.

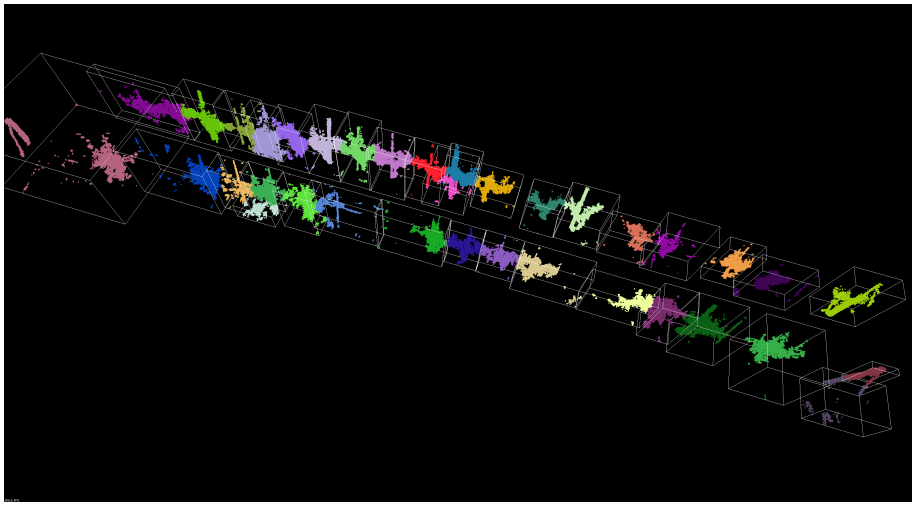


Figure 5.21: Segmentation Algorithm - Bounding Boxes: plants clusters with bounding boxes. Points are enlarged in size for ease of visualization.

Chapter 6

Experimental Results And Discussion

In this Chapter, we discuss the results of the experimental activity at a botanical garden near our university. In Section 6.1, we describe the experimental field, with a focus on a vineyard row. In Section 6.2, we present evaluations of the various odometry approaches proposed in Chapter 5. In Section 6.3, we draw conclusions about the two types of reconstruction performed by our system, that is, Visual Reconstruction and LiDAR Reconstruction. Finally, in Section 6.4, we evaluate and discuss our Segmentation algorithm.

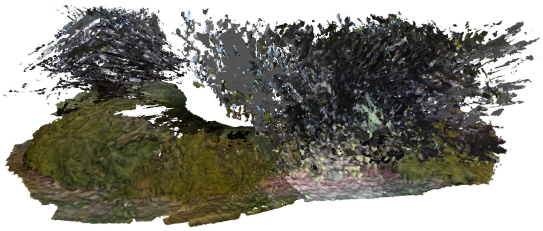
6.1 Experiments Field

We performed outdoor experiments at the botanical garden "Orto Botanico Città Studi" located in Milan (45°28'30.4"N 9°14'03.8"E). There, we scanned different plant species with a focus on the vine (*Vitis vinifera*). Experiments have been performed from March 2019 to June 2019 during morning hours (from 9 am to 12 am). Figure 6.1 shows a picture of our system manually pulled during data collection. Figures 6.2 and 6.3 show pictures of some natural environment, with trees and bushes, and their respective 3D colored visual reconstructions.

Above all, we focused our attention on the vineyard of Figure 6.4. This picture has been taken in September 2019. The one of Figure 6.5 has been taken at the beginning of the season in March 2019; there, trunks and poles can be clearly recognized. The vineyard is a single row composed by two sides. The supporting structure is constituted of wood poles and metal wires. It is 26 m long, and it has an average width of 2.18 m. Poles have an average height of 1.87 m. There are 25 poles for each side.

Figure 6.1: A typical session of outdoor data collection.

(a) The picture of the scanned subject.



(b) The 3D colored mesh.

Figure 6.2: A visual reconstruction of bushes.

(a) The picture of the scanned subject.

(b) The 3D colored mesh.

Figure 6.3: A visual reconstruction of an hedge.

Figure 6.4: The vineyard in September.

Figure 6.5: The vineyard in March, at the beginning of the season. It is possible to clearly recognize poles and trunks.

6.2 Odometry Evaluation

Our system can perform odometry by various combinations of sensors (see Subsection 5.2.1). To understand which is the best combination, we performed experimental tests evaluating different subsets of sensors. Thus, we first collected sensors data at the botanical garden, memorizing them in ROS bag files. Then, we worked offline playing back the recorded bag files. We played them at a lower frequency than real-time to have the guarantee the odometry node processed all frames. Then we launched our system with different odometry settings, and we visually inspected the colored and LiDAR point clouds and the odometry trajectories.

We purposely did not make any loop closure during experiments. For this reason, we deactivated loop closure to avoid false positive detections and to speed up the computation. Our experimental field was too small to generalize the effectiveness of loop closures to real farming settings. In the future, we expect to bring our system in farming crops to perform large-scale experiments. Consequently, it will worth trying to understand the feasibility of loop closure and their effectiveness compared to other alternatives.

In this Section, we also discuss how we modified the default parameters of each ROS node. We want to point out that all the parameters have been tuned by visual inspection of the resulting point clouds and odometry trajectories. Obtaining a ground truth for outdoor settings is hard. The GPS signal is usually employed as ground truth for outdoor experiments, but in our case, it was not possible as we used the GPS as an odometry source. Since we did not have any ground truth, the only information we could exploit is the fact that the vineyard row sides are reasonably straight and parallel. In the following, we list the various odometry combinations we tried discussing the strength and weakness points of each approach.

D435 Visual Odometry A possible odometry setting is the one that employes just the camera data to perform Visual Odometry. The architecture functioning principle is the following. The `rtabmap rgbd_odometry` node provides a local odometry source and writes it on the `odom->base_link` transform. The global correction written on the transform `map->odom` should be in charge of the `rtabmap` node as result of loop closures. In this setting, as the loop closure detection is deactivated, we do not have any global correction. Thus, the transform `map->base_link` is just the identity.

It is still worth to analyze the data coming from plain visual odometry to understand its accuracy as a local reference. Indeed, it is not possible to use plain visual odometry for large environments because of the accumulated drift. However, for a small-scale environment like the one we scanned, it is still pretty accurate.

The modified parameters of the node `rgbd_odometry` are listed in the following:

- `Odom/FilteringStrategy` set to "Kalman Filtering" to make the odometry smoother. Indeed, the visual odometry is given as input to a Kalman Filter and weighted with a motion model

- `Odom/ResetCountdown` set to 5. This is the number of frames the odometry update can fail before the odometry reset. The reset causes the odometry to restart from the latest computed pose

- the `Odom/Holonomic` parameter was leaved with the default value of `true`. However, it is worth to explain this choice as our robot is nonholonomic, but it is characterized by a tricycle kinematic. Due to the cart structure, when the platform is pulled on uneven outdoor terrains, it produces low-frequency wide oscillations in the y-axis direction (the direction perpendicular to the one of movement, and parallel to the ground). Hence, as those oscillations are so pronounced, we can not assume our robot to be nonholonomic

- the `Vis/MaxDepth` parameter set to 6 m and `Vis/MinDepth` set to 0.15 m so to exclude too far and too close image points that could be affected by significant noise

Even if the visual odometry approach showed good quality (Figure 6.6), it was not free of problems. Indeed, it got lost repeated times during a single scanning session. For us, this was not a severe problem as our robot was moving straight constrained by vineyard row. Thus, the odometry reset did not affect too much the accuracy. The loss of odometry was caused by a too much low number of inliers after PnP RANSAC estimation or after Bundle Adjustment.

Further investigation is needed to understand the cause of such a low number of inliers. A hypothesis is that the loss of odometry is caused by the light changes produced by the auto exposure. Indeed, it is known that the feature matching task is widely affected by light variations. In the future, it will be better to disable the auto exposure, or to auto expose a limited Region Of Interest of the entire image. Another idea, to enhance the feature matching, thus possibly highering the number of inliers, is to provide to the odometry node the IR images instead of the RGB ones. This could help the feature matching task as the D435 camera enriches its IR images employing an IR projector.

Finally, it will be of fundamental importance to investigate how the camera angle position affects the odometry. At this moment, the cameras are placed parallel to the scanned subjects. In the future, it will be useful trying to point the cameras more toward the center of the vineyard row. This will extend their view on the row sides, hence having more overlapped scenes, and possibly helping the odometry task.

In Figure 6.6 it is showed the odometry trajectory by a blue line. Since there is not global correction, the trajectory, and in turn the reconstruction, presents a moderate curvature. However, note the smoothness and continuity of the trajectory. The computed odometry leads to the assembled LiDAR point cloud of Figure 6.7. The reconstruction has no holes, and it is coherent with the real vineyard. This is due to the continuity of the local odometry with the additional smooth of the Kalman Filtering. Still, let note how wavy is the trace left in the center of the row from the occlusion of the vertical LiDAR. This is exactly due to the lateral oscillations of the cart. In Figure 6.8 it is possible to see a section of the reconstructed colored point cloud of a row side.

D435 and IMU fusion Even if the integration of the IMU as an odometry source has not been our priority, it is still interesting to look at the result of a basic sensor fusion. As already stated, the IMU is necessary for our systems since the GPS node requires magnetometer data. So, we tried a fusion with the D435 Visual Odometry and the STM-NUCLEO IMU. The data collected from the D435i IMU are unavailable since the multiple

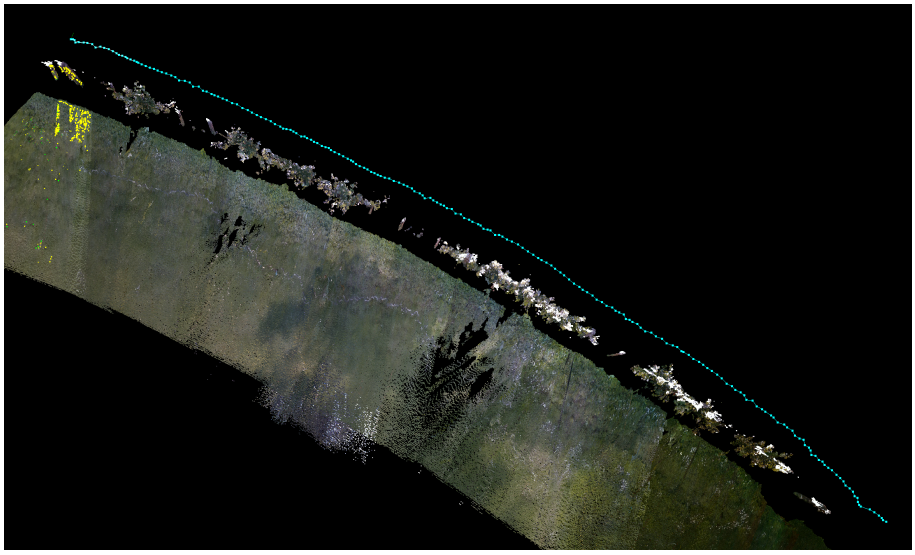


Figure 6.6: D435 Visual Odometry experimental odometry.

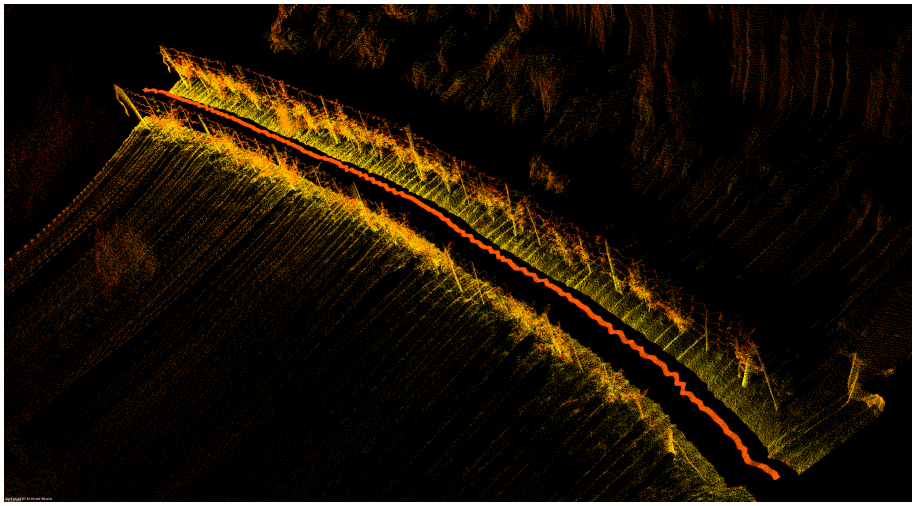


Figure 6.7: D435 Visual Odometry experimental LiDAR point cloud.

cameras setting created conflict problems. Further investigation is needed to understand how to integrate multiple D435 cameras in the same system.

We performed the fusion with an Extended Kalman Filter from the ROS package `robot_localization`. We had the visual odometry node publishing the local odometry as in the above case. This time, the global correction was produced by the odometry fusion coming from the Kalman Filter. The experimental results show just a little improvement compared to plain Visual Odometry.

In Figure 6.9 it is possible to see that the resulting odometry is smooth a pretty straight. The LiDAR point cloud showed in Figure 6.10 seems a little better of the preceding one. Note that the central row trace is less wavy compared to the one of Figure 6.7. An inspection of the colored point cloud of Figure 6.11 shows a result similar to the preceding odometry.

The only thing we can add is that the colored reconstruction presents more unmatched

Figure 6.8: D435 Visual Odometry experimental colored point cloud.

surfaces, especially the ones of the turf adjacent the vineyard row. Thus, having comparable results to the preceding odometry, we can not draw any conclusion on the usefulness of the IMU integration. However, it is reasonable to think that the IMU could be of help, especially with oscillations; thus, further research is needed. Here we list some ideas for a better IMU integration:

- the IMU needs a sound calibration. The accuracy of the manual measurements is probably not enough to exploit the IMU information. Thus, automatic calibration is required. A possibility is to perform camera-IMU calibration via the well-known Kalibr calibration package[1], as described in [47]. Another possibility is to exploit the IMU embedded in the D435i camera since it comes already calibrated

- the `robot_localization` requires a proper covariance matrix for each input to work correctly. However, for this IMU, covariance data was not specified in the datasheet. In this case, algorithmic ways should be investigated to estimate the IMU covariance matrix

- it is worth trying purposely designed algorithm for the fusion of visual odometry and IMU data, like OKVIS [48] or MSCKF [49]

The modified parameters are the same as D435 Visual Odometry, except for the `Odom/FilteringStrategy` that has been disabled since the odometry is already filtered by the `robot_localization` Kalman Filter.

LiDAR Odometry Another odometry approach we tried is the one with LiDAR data as input of the `rtabmap icp_odometry` node (see Subsection 5.2.1). In our case, the node can take as input the scans from the LMS100 or the LD-MRS LiDAR sensors and outputs

[1] `https://github.com/ethz-asl/kalibr`

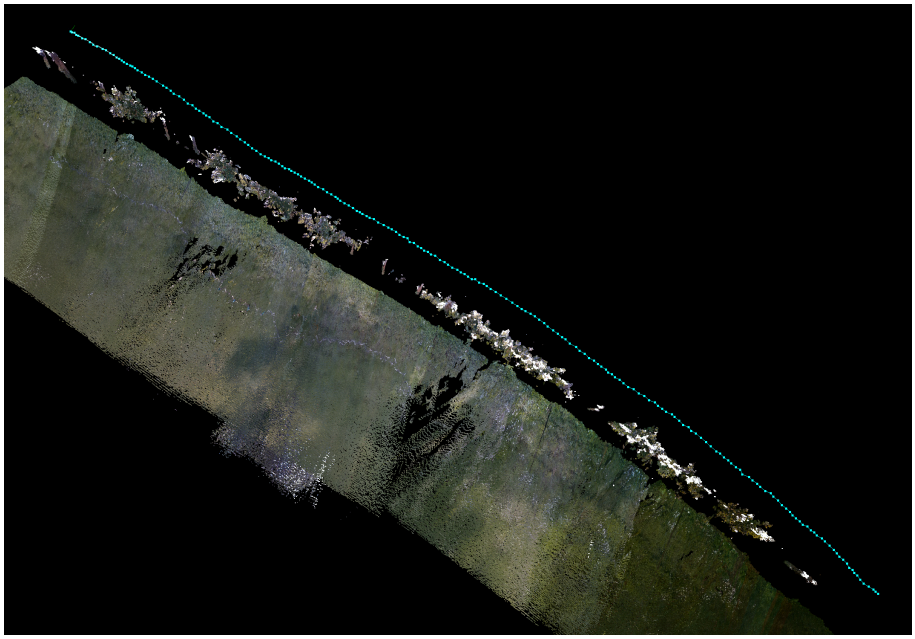


Figure 6.9: D435 and IMU fusion experimental odometry.

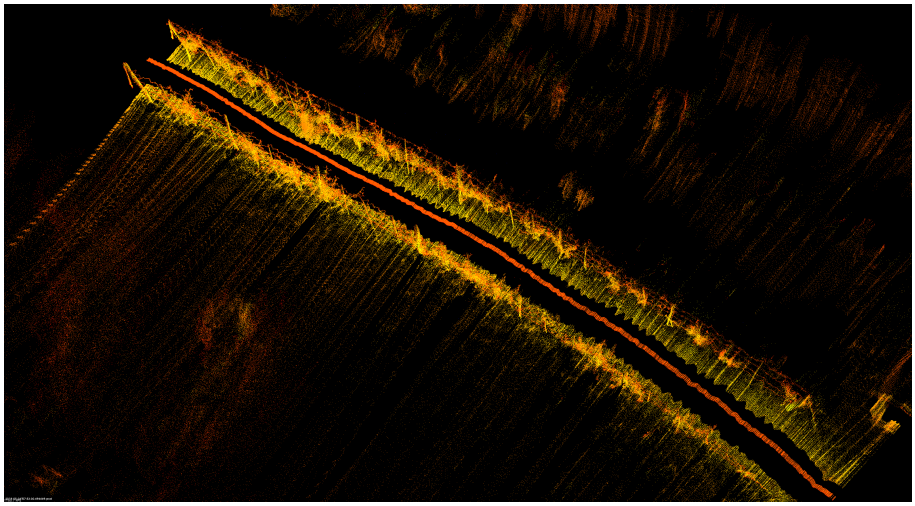


Figure 6.10: D435 and IMU fusion experimental LiDAR point cloud.

an odometry message named `odom_icp`. The working principle is the one of ICP scan matching, as explained in Subsection 3.2.1.

Successive scans can be matched if they share overlapping scanned surfaces. Thus, in case of a LiDAR with single or few scanning planes, it is crucial to have them as parallel to the ground as possible. In the extreme case of a single vertical plane, ICP odometry will not give any information on motion along the moving direction. Starting from the second sensor setup (see Section 4.3), we mounted the LMS100 LiDAR vertically. Here we provide an evaluation of LiDAR odometry performed with the LD-MRS four planes sensor. As in the case of plain Visual Odometry, the `icp_odometry` is used as a local reference, and no global correction is performed.

From visual inspection of the results, we can state that LiDAR odometry demonstrated

Figure 6.11: D435 and IMU fusion experimental colored point cloud.

the worst quality compared to other approaches. In Figure 6.12 it is possible to see a higher view of the odometry trajectory and the trace left by the LiDAR scans (the blue points). The odometry is very poor when computed at the beginning of the vineyard row, and gets a little better when the robot has traversed part of the row. This can be explained looking at what points are used for ICP matching in those different moments.

At the beginning of the row, the only points perceived are the farthest ones, highlighted with a red ellipse in the cited Figure. While the robot goes forward, far points do not change their position; thus, successive scans will be almost identical. This problem is accentuated by the restricted horizontal angular aperture of the LD-MRS sensor. Then, as the robot proceeds further in the vineyard, the LiDAR starts to see points on the lateral sides of the row, that is, the ones highlighted with the green ellipse in the same cited Figure. After some time, only lateral points will be detected, hence making the ICP matching more accurate.

In Figure 6.13 it is possible to see that the assembled LiDAR point cloud is of low quality. The lateral vineyard sides are wavy and crooked. Finally, in Figure 6.14 it is showed the respective colored point cloud.

The modified parameters are:

- `Icp/PointToPlane` set to `false` as we are not in structured human-made environments with a lot of plane surfaces. Hence, ICP registration is done using Point to Point (P2P) instead of Point to Plane (P2P) correspondences

- `scan_voxel_size` set to 0.04 m to enable voxel grid filtering. Voxel grid filtering works by superimposing a cubes grid on a point cloud. The cubes have a parametrized fixed edge length. Then, all the points that fall in a cubic cell are merged into a single point. Different merging strategies are possible, such as the centroid of the points or the center of the cell. This kind of filtering has been enabled on the point cloud produced by the LD-MRS sensor to ease the registration

task and to reduce noise that can affect odometry accuracy. From a visual inspection, this parameter seems fundamental to not make the odometry broke completely. In fact, in Figure 6.15, it is possible to see the resulting odometry computed without voxel filtering. However, as usual, we can not draw strong conclusions about the effect of voxel grid filtering for the lack of a sound statistical analysis

- `Icp/CorrespondenceRatio` has been lowered from 0.1 to 0.08 for accounting the increased noise of outdoor environments compared to indoor ones
- `Odom/FilteringStrategy`, `Odom/ResetCountdown`, `Kp/MaxFeatures`, and `Rtabmap/StartNewMapOnLoopClosure` parameters have been set as in the Visual Odometry approach

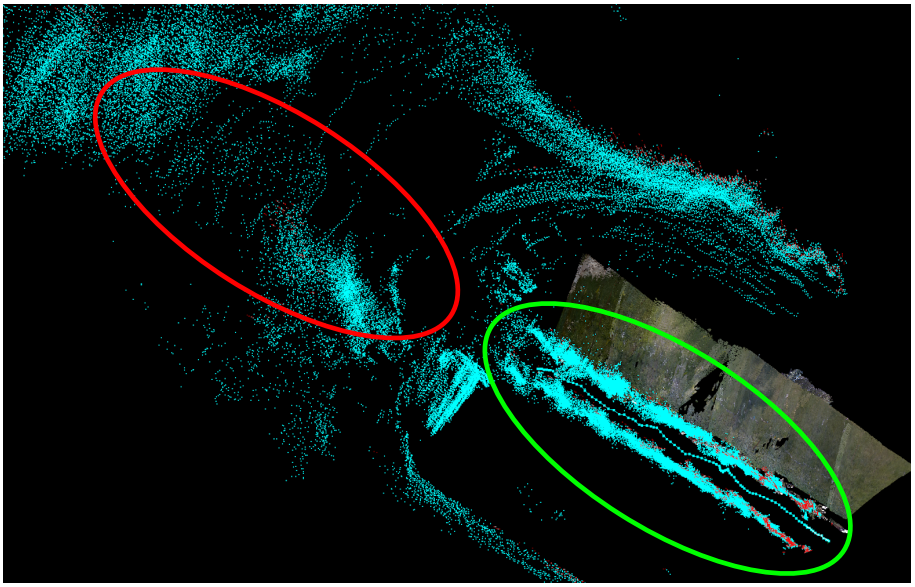


Figure 6.12: LD-MRS experimental LiDAR Odometry. In red, the farthest points as captured by the LiDAR sensors at the beginning of the vineyard row. In green, the LiDAR captured points as the robot proceeds further in the vineyard row.

D435 and GPS fusion The last presented alternative is the one that fuses Visual Odometry data with GPS. In this case, the Visual Odometry is used as a local reference and made to write the `odom->base_link` transform. The odometry fusion is computed via a Kalman Filter and used as global correction writing it on the transform `map->odom`.

The global odometry is also given as input to the `rtabmap` node responsible for the visual reconstruction. However, since the global odometry can make discreet jumps, the reconstructed map could result broken. Indeed, as loop closure detection is deactivated, no graph optimizations will be performed. Nodes are added by the `rtabmap` node based on the provided odometry; thus, jumps in the odometry will produce jumps in the graph. In the future, a more sound strategy for GPS integration should be devised.

The odometry computed with this approach is the best compared to the others. In Figure 6.16 it is possible to see a smooth and straight odometry trajectory. Figure 6.17 shows the LiDAR point cloud. In this case, the vineyard row is almost perfectly straight

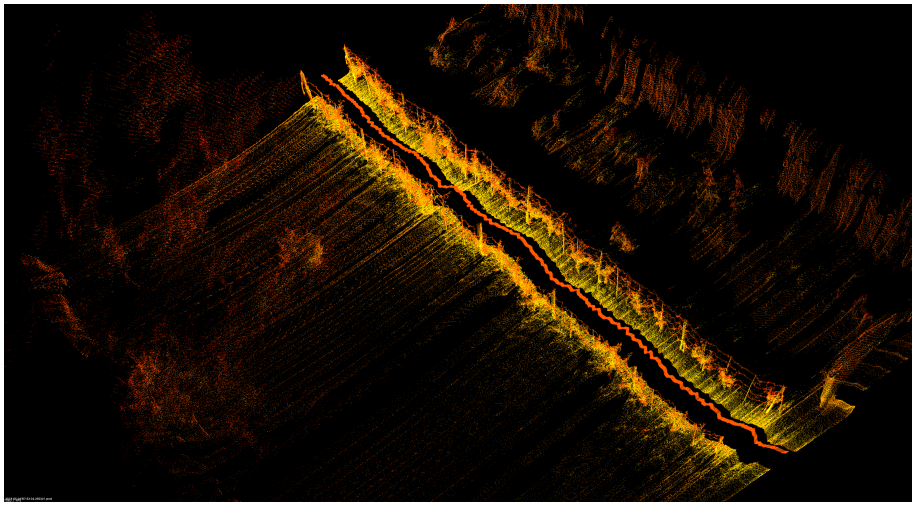


Figure 6.13: LD-MRS LiDAR Odometry experimental LiDAR point cloud.

Figure 6.14: LD-MRS LiDAR Odometry experimental colored point cloud.

and does not present the curvature that characterized the other odometry approaches. Finally, in Figure 6.18 it is showed the colored point cloud.

The parameters are the same as in the case of plain Visual Odometry, except for the `Odom/FilteringStrategy` that has been disabled since the odometry is already filtered by the `robot_localization` Kalman Filter. As for the GPS `navsat_transform_node` we set the `magnetic_declination_radians` parameter to the magnetic declination of our experiments field. As for the Kalman Filter node, we set to `true` the `two_d_mode` parameter that makes the filter to assume planar movements.

6.3 Reconstruction Evaluation

Visual Reconstruction The Visual Reconstruction is built by the `rtabmap` node as explained in Subsection 5.2.2. The `rtabmap` node uses the `map->base_link` transform to add Neighbor links between successive nodes. Every time a new node is created, its Local Map is merged with the Global Map, thus forming the generated colored point cloud.

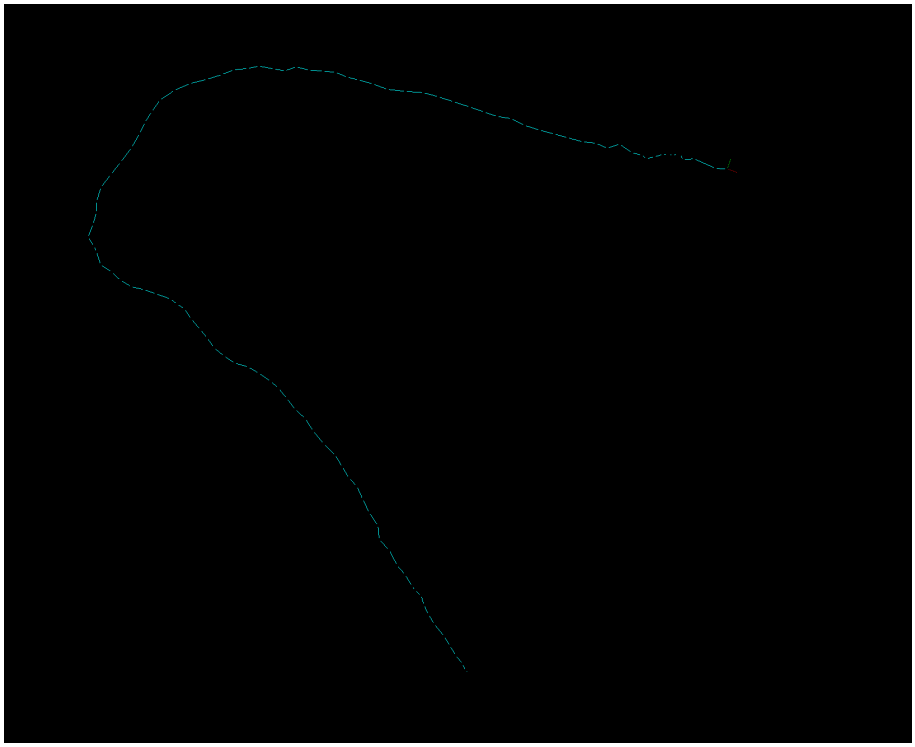


Figure 6.15: LD-MRS LiDAR Odometry broken as computed without voxel filtering.

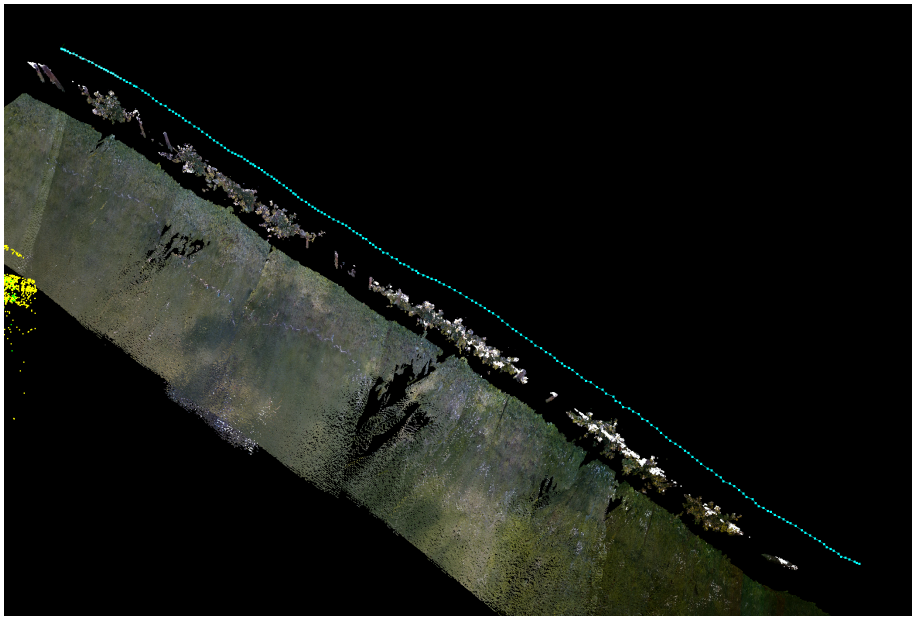


Figure 6.16: D435 and GPS fusion experimental odometry.

After experimental activity we modified some parameters of the `rtabmap` node:

- the `RGBD/NeighborLinkRefining` parameter set to `true` is crucial for an accurate Visual Reconstruction in case of an inaccurate odometry. The attentive reader will have noticed that all the previously showed colored point clouds are pretty similar to each other. Nevertheless, we have seen that some approaches have resulted in

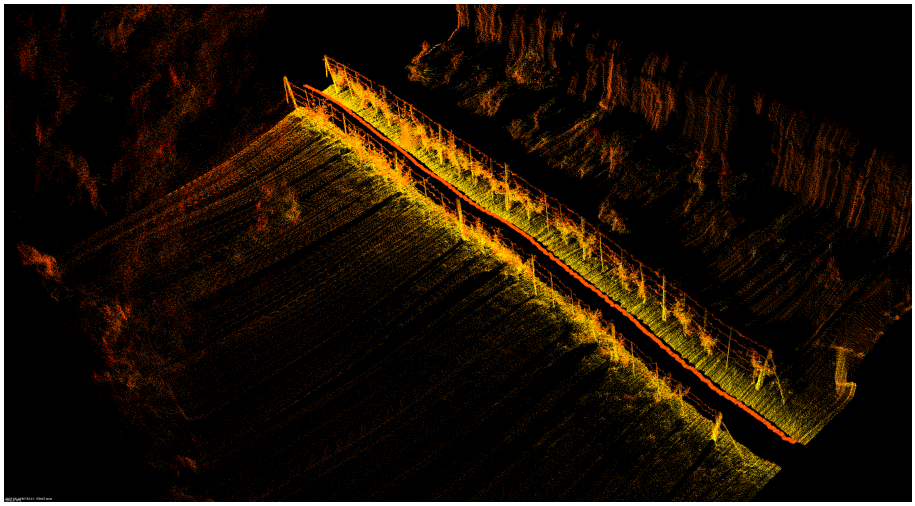


Figure 6.17: D435 and GPS fusion experimental LiDAR point cloud.

Figure 6.18: D435 and GPS fusion experimental colored point cloud.

very poor odometry, like the one with the LD-MRS sensor. Indeed, the LiDAR point cloud assembled with this latter odometry approach is of really low quality as can be seen in Figure 6.13. However, the colored point cloud of Figure 6.14 seems comparable to the one produced by the other approaches. This is possible precisely thanks to the above-mentioned parameter.

When this parameter is activated, the links added to the SLAM graph are refined. The external odometry is used just as a guess for a new visual registration between the current image frame and that of the preceding node. The result will be a newly refined transform that will be added as a neighbor graph link. In case the odometry is already accurate, the `RGBD/NeighborLinkRefining` parameter could be disabled as it leads to an extra computational load

- `publish_tf` set to `false` as we do not want the odometry correction coming from Loop Closure to be published on the `tf` tree. The correction is done via the GPS, if available

- `Kp/MaxFeatures` set to -1 to disable Loop Closure detection. Indeed, as already explained in the introduction of Section 5.2, our system performs global odometry corrections via the GPS
- in case of real-time computation it is possible to set the `Rtabmap/TimeThr` parameter to 200 ms and `Rtabmap/MemoryThr` parameter to 300 nodes, for activating the memory-management feature of `rtabmap`. In our case, these parameters are disabled since we make an offline processing
- `wait_for_transform_duration` from 0.1 s (default) to 0.3 s to be less restrictive on the waiting time for a transform
- as we are processing offline, we increased the `Rtabmap/DetectionRate` parameter to 4 Hz without worries about memory usage. This is the rate at which `rtabmap` creates new nodes in the SLAM graph

In Figure 6.19 it is shown the colored Visual Reconstruction of an entire vineyard row side.

Figure 6.19: A colored Visual Reconstruction of an entire vineyard row side.

LiDAR Reconstruction The LiDAR point cloud is assembled by the `laser_scan_assembler` node as already explained in Subsection 5.2.3. The software architecture for this type of reconstruction is straightforward as it just uses the odometry poses to assemble single scans into a point cloud. Thus, the quality of the point cloud is directly related to the odometry approach employed. The LiDAR Reconstruction code does not require any particular parameter to be set. In Figure 6.20 is shown the same point cloud of Figure 6.17 but with a closer view.

6.4 Segmentation Algorithm Evaluation

In this Section, we evaluate the accuracy of the Segmentation algorithm that has the purpose of isolating single plants from the vineyard LiDAR reconstruction. Experimental tests have been performed using data recorded in a scanning session on the 24th May 2019, around 10 am. We have then reconstructed LiDAR point clouds with the different odometry approaches presented above, and we have run the segmentation algorithm on them.

The vineyard row is constituted by two sides, that we indicate as left and right, as seen by someone facing in front of the row at the robot starting position. Both sides have 25 poles each. The left side has 11 plants, while the right side has 19 plants. However, one plant of the right side does not have any leaf or branch; thus in the following, we will

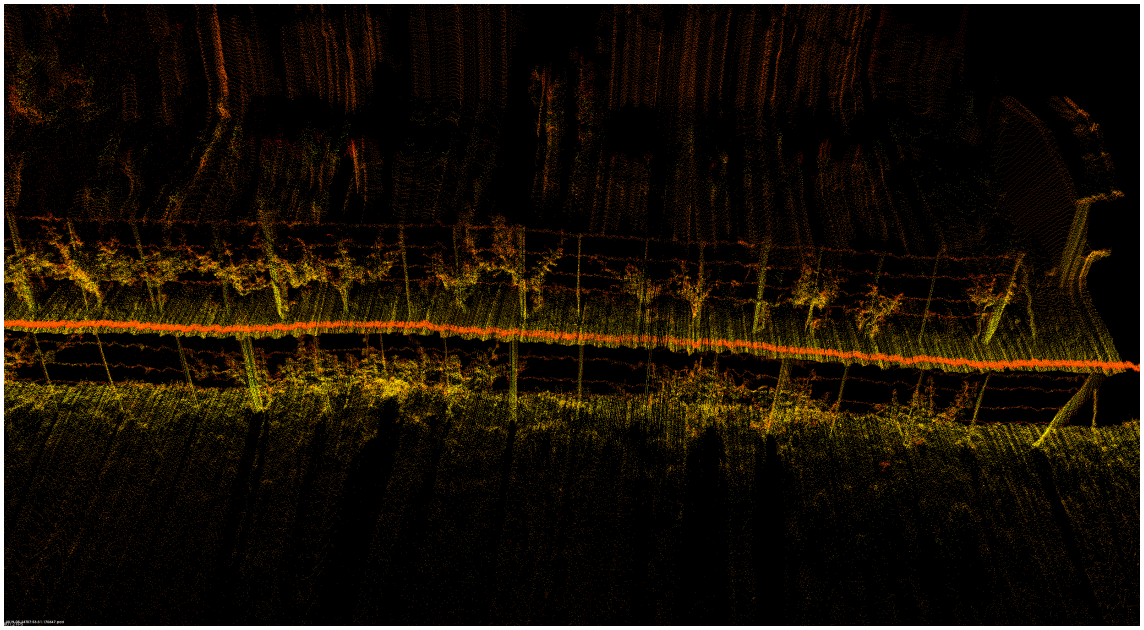


Figure 6.20: A close view to the LiDAR point cloud of Figure 6.17.

only consider 18 plants for this side. Here we list, for each kind of odometry, the number of clusters that have been found by the segmentation algorithm for each side. As it is possible to see in the corresponding Figures, a cluster can correctly contain a single plant, a plant and a pole, or parts of plants and poles.

- D435 plain Visual Odometry (Figure 6.23): left side - 17 clsuters; right side - 19 clusters
- D435 and IMU fusion (Figure 6.24): left side - 17 clusters; right side - 22 clusters
- LiDAR Odometry (Figure 6.25): left side - 20 clusters; right side - 23 clusters
- D435 and GPS fusion (Figure 6.26): left side - 17 clusters; right side - 21 clusters

Quantitative Evaluation To quantitatively evaluate the segmentation algorithm, it should be seen as a binary classification algorithm. Indeed, each element of our dataset is classified as plant or non-plant (thus, as a pole). However, our dataset is a point cloud; thus, single elements are not defined. So, our algorithm is in charge not only of the classification but also of the identification of the single elements. Thus, it should group a bunch of points and see them as a single element. Once the dataset has been divided into single elements, it possible to proceed to the classification and tell if those elements are plants or not. Our algorithm determines single elements by the trunks clustering procedure. Indeed, each cluster will give birth to a centroid, and by our assumption, a centroid could only belong to a pole or a plant trunk.

If we see our algorithm as a binary classification, we can compute for each dataset the confusion matrix. The concept of confusion matrix is represented in Table 6.1. The actual class is the class of the element as attributed by a human. The predicted class is the one given by the classification algorithm. As the algorithm is binary, the possible classes

are plant and non-plant. Thus, after the algorithm has classified each element, we can proceed to count the True Positives, False Positives, False Negatives, and True Negatives. True Positive is the number of elements that are plants and are correctly predicted as a plant. False Positive is the number of elements that are non-plant but are incorrectly classified as a plant. False Negative is the number of elements that are plants but are incorrectly classified as non-plant. Negative is the number of elements that are non-plant and are correctly classified as non-plant. However, the confusion matrix presented here is modified to take into account the peculiarities of our task. Indeed, we define and additional count called False Positive$^+$. Its meaning is explained below.

Thus, the idea is to compute confusion matrices for the point clouds computed with each different odometry approach. Figure 6.21 shows a schema of the vineyard where we manually tagged each element, assigning it to the plant class (PL), or the non-plant class (NPL). Then, we define some rules for assigning the predicted class:

- if a plant, or part of it, belongs to a bounding box, it is considered a True Positive (TP). If the plant is only covered in part by a bounding box, each other bounding box covering the remaining part is considered an additional False Positive (FP^+)
- if a pole, or part of it, belongs to a bounding box, it considered a False Positive (FP). If the pole is only covered in part by a bounding box, each other bounding box covering the remaining part is considered an additional False Positive (FP^+)
- if a pole has been correctly filtered out by the Variance Filtering procedure, it is considered as True Negative (TN)
- if a plant has been incorrectly filtered out by the variance filtering procedure, it is considered as False Negative (FN)
- *if a bounding box comprises parts of plants or poles, it is considered as an additional False Positive. Thus we have* $FP^+ = FP + \#additional\_bounding\_boxes$

The last rule is highlighted as it makes the confusion matrix computation tricky. Indeed, as previously said, our algorithm is in charge of determining the elements composing the dataset. However, it can get wrong by putting bounding boxes in places where they should not be (see Figure 6.22). For example, inside other bounding boxes or comprising parts of multiple plants or poles. Thus, these additional bounding boxes are added to the total number of False Positives giving birth to the FP^+ count. Finally, it is still true that $TP + FP + FN + TN = P + N$ (where P are Positives, and N are Negatives), but $TP + FP^+ + FN + TN \neq P + N$. Considering the additional False Positives is fundamental to correctly assessing the quality of the segmentation algorithm as we are expecting bounding boxes to comprise single and entire plants. If a bounding box comprises parts of plants or poles, it is a synonym of bad quality.

Moreover, it should be taken into consideration that a bounding box could contain multiple plants. Following the above rules, all the plants belonging to a bounding box are considered True Positive. Thus, a single bounding box covering all the plants will maximize the number of True Positives. Thus, a quality index should inversely proportional weight the number of True Positives by the number of bounding boxes detected.

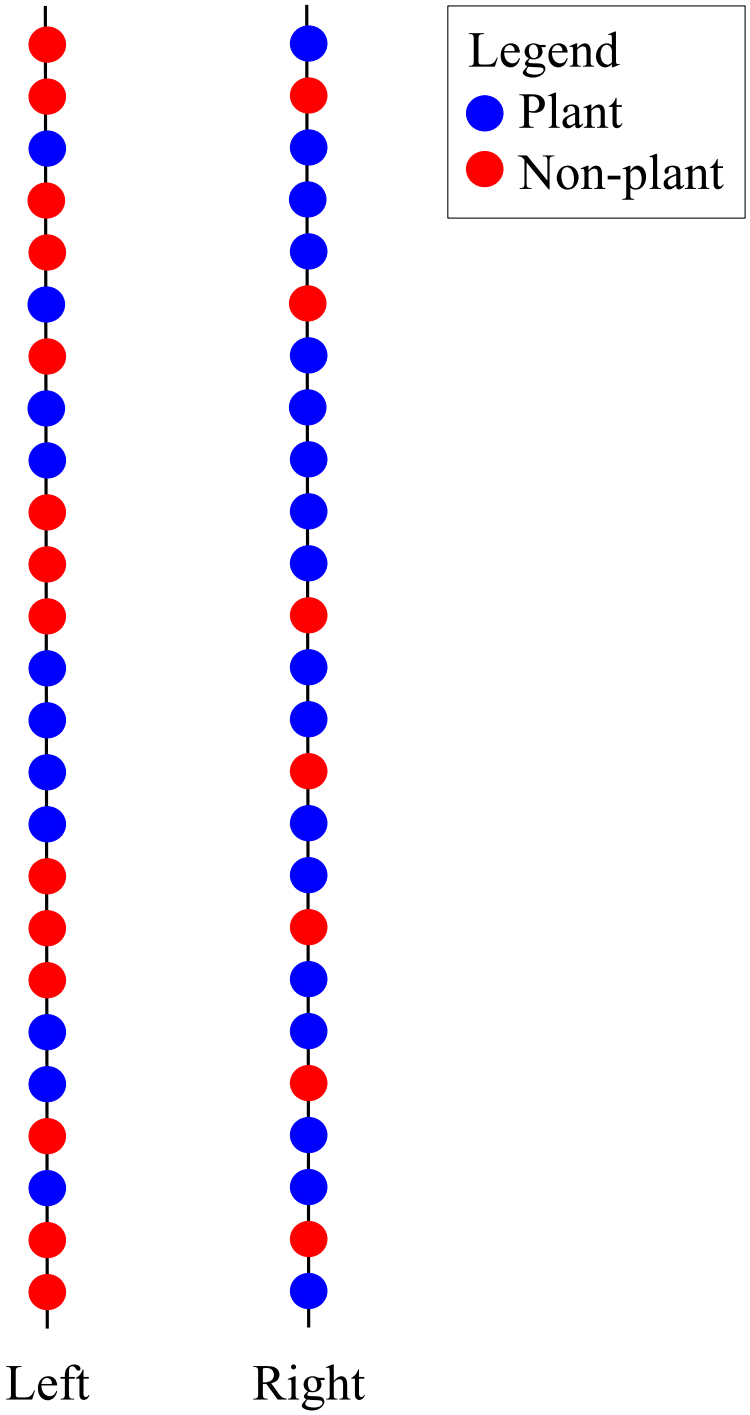


Figure 6.21: Vineyard ground truth.

Finally, classical indices such as True Positive Rate (TP/P), True Negative Rate (TN/N), Accuracy, etc. can not be computed as they are. Indeed, it is not clear how to correctly use the FP^+ count. Further research is needed.

Tables 6.2, 6.3, 6.4, and 6.5 show the confusion matrices of the point clouds from the four different odometry approaches.

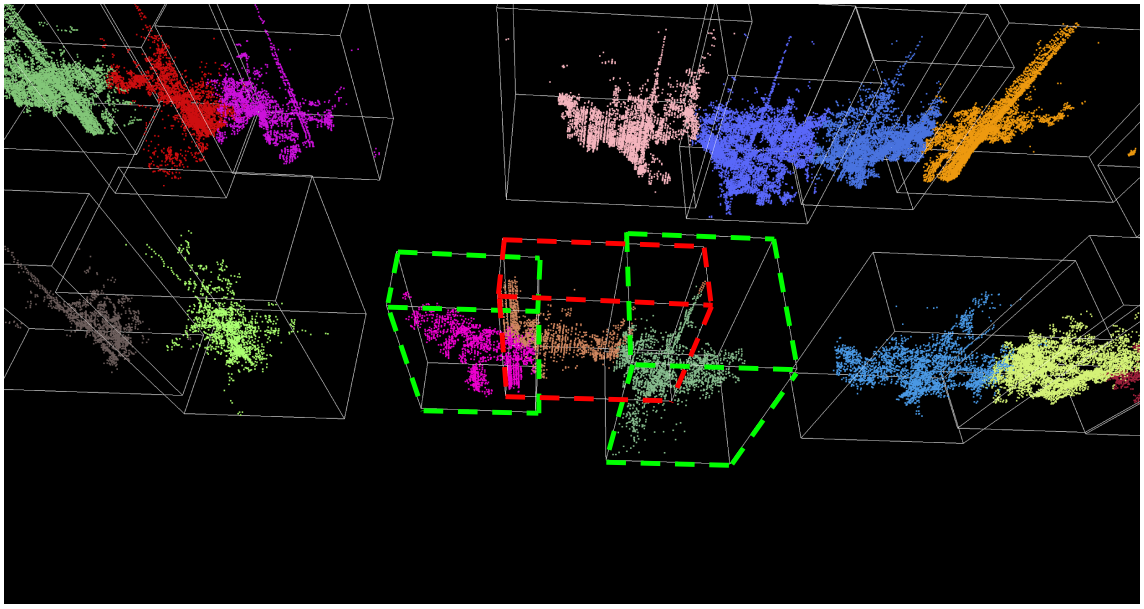


Figure 6.22: In the Figure are highlighted in green two correctly placed bounding boxes around two plants. However, an additional bounding box, highlighted in red, is placed in between of the two plants. This additional bounding box is counted separately as False Positive (FP^+).

		Actual class	
		Plant	Non-plant
Predicted class	Plant	True Positive	False Positive(+ #BoundingBoxes)
	Non-plant	False Negative	True Negative

Table 6.1: A general confusion matrix.

		Actual class	
		Plant	Non-plant
Predicted class	Plant	26/29	12/21(+1 BBs)
	Non-plant	3/29	9/21

Table 6.2: Confusion matrix of the Segmentation algorithm applied to the point cloud from D435 Visual Odometry.

Parameters Tuning The experimental activity required us to tuned some parameters for each different point cloud. The parameters have been tuned via a trial-and-error procedure, and they are listed in the following:

- the angle `theta` of the rotation of the Affine Transformation. This angle depends on the robot starting position that differs for every run of odometry. However, the tuning of this parameter can be automated as explained is Section 5.3

		Actual class	
		Plant	Non-plant
Predicted class	Plant	29/29	13/21(+2 BBs)
	Non-plant	0/29	8/21

Table 6.3: Confusion matrix of the Segmentation algorithm applied to the point cloud from D435 and IMU fusion odometry.

		Actual class	
		Plant	Non-plant
Predicted class	Plant	28/29	9/21(+9 BBs)
	Non-plant	1/29	12/21

Table 6.4: Confusion matrix of the Segmentation algorithm applied to the point cloud from LD-MRS LiDAR Odometry.

		Actual class	
		Plant	Non-plant
Predicted class	Plant	29/29	7/21(+3 BBs)
	Non-plant	0/29	14/21

Table 6.5: Confusion matrix of the Segmentation algorithm applied to the point cloud from D435 and GPS fusion odometry.

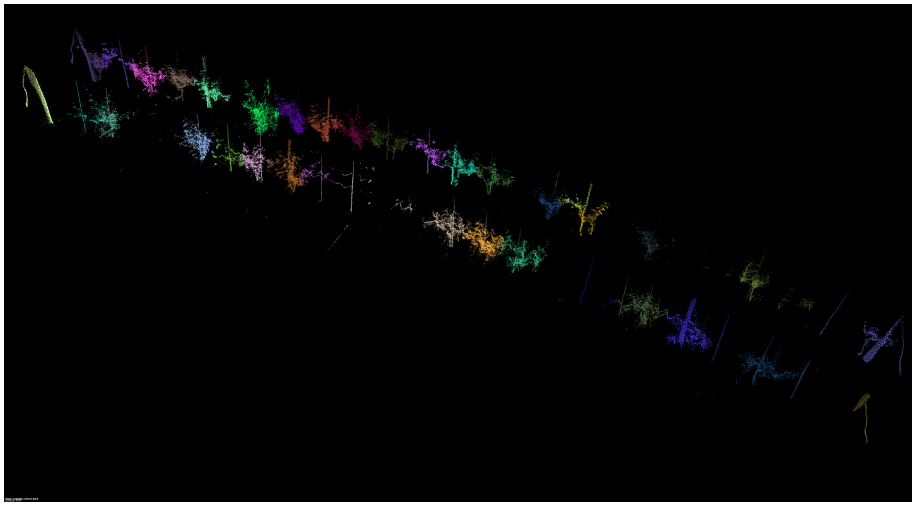


Figure 6.23: D435 Visual Odometry segmented LiDAR point cloud.

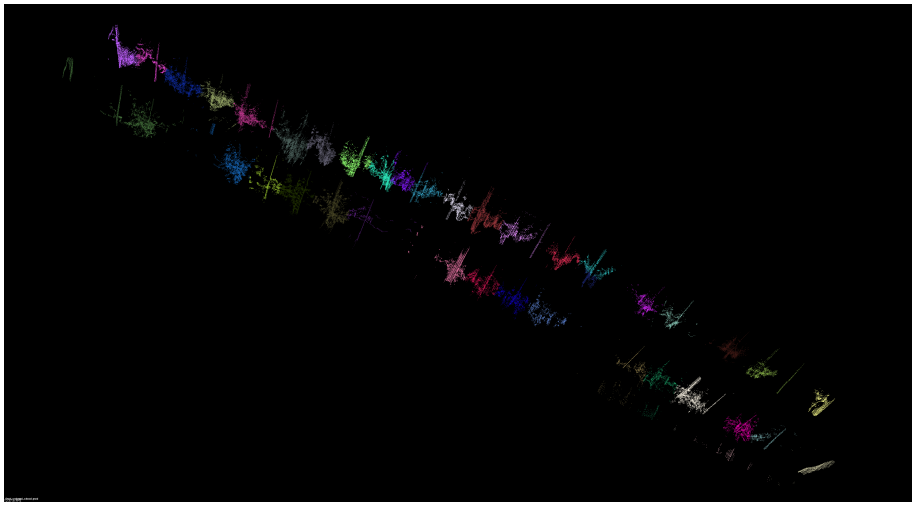


Figure 6.24: D435 and IMU fusion odometry segmented LiDAR point cloud.

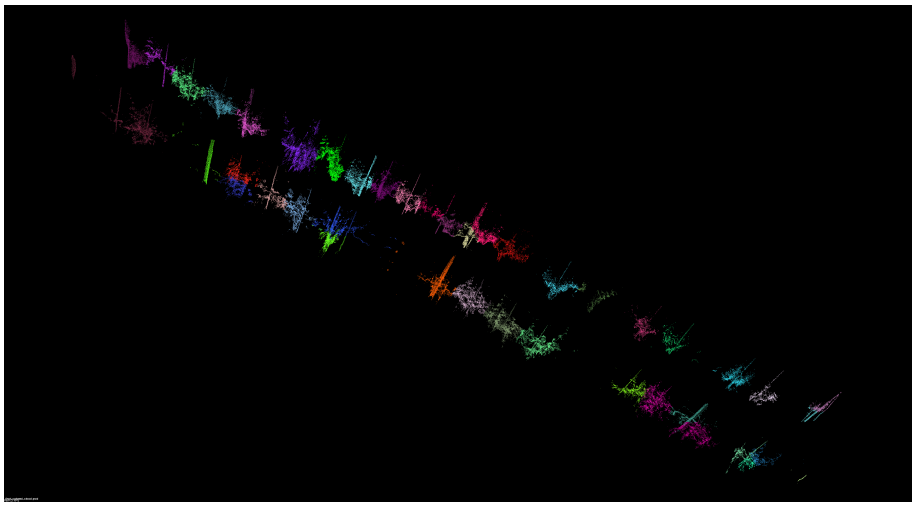


Figure 6.25: LD-MRS LiDAR odometry segmented LiDAR point cloud.

- also the Z filtering thresholds depend on the robot starting position, thus requiring to be modified. However, even the tuning of this parameter could be automated. It suffices to fix the low threshold at the level of the fitted ground plane; then the upper threshold is just calculated as a fixed amount over the lower threshold

- two Euclidean Clustering parameters need to be tuned: the minimum number of elements in a set of points to be considered a cluster, and the search radius. These parameters vary upon the point cloud section resulting from the preceding mentioned Z filtering

Segmentation Algorithm Conclusions Our segmentation algorithm showed excellent performance to recognize plants as can be seen by the number of True Positives that is almost the maximum for all odometry approaches. The higher accuracy has been obtained with the most accurate sources of odometry, that is, the D435 plain Visual Odometry and its fusion with the GPS. The segmentation of the left side is less accurate than the right

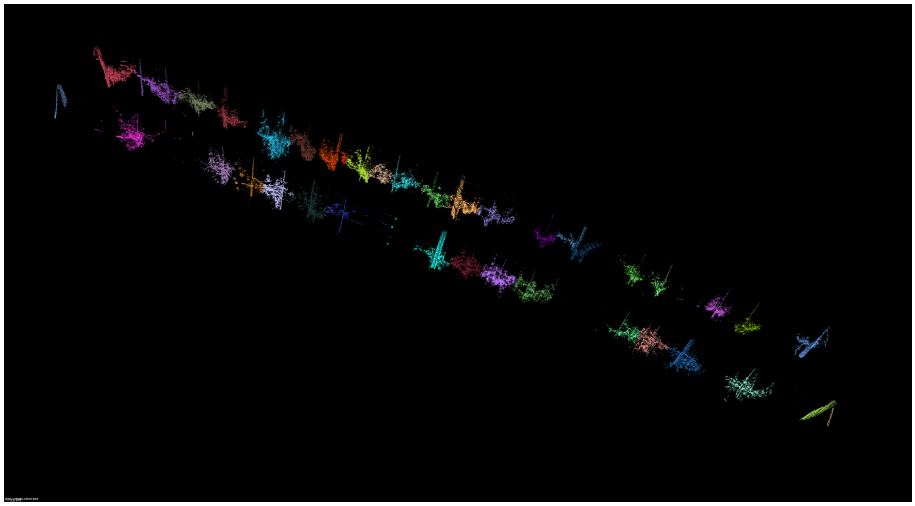


Figure 6.26: D435 and GPS fusion odometry segmented LiDAR point cloud.

side. Indeed, the left side has fewer plants than the right side, 11 plants versus 18 plants (over 25 poles). This means that the poles filtering procedure is not that effective to remove poles. This is caused by the fact that the assumption of empty space around a pole does not always hold. Especially in vineyard farmings, it happens that plants climb along wires, thus covering neighbor poles. However, it is even difficult for a human that sees the point cloud to understand if, in a specific position, there is a plant, or it is just another plant extending over a neighbor pole. Moreover, it is not true that all circles containing poles projections have less variance than one of the plants since there are poles with big diameter. These considerations are reflected in the low number of True Negatives of the confusion matrices.

The trunks clustering procedure is the most critical part of the algorithm. The plants' segmentation accuracy depends on the majority by the accuracy of the trunks clustering. Our trunks detection algorithm failed in some cases to discriminate between trunks and poles due to the assumption that each cluster is either a pole or a trunk. Indeed, grass bushes and plant branches could be seen as an additional cluster, thus, as an additional centroid. Finally, each vine is planted nearby a pole, and if the plant trunk is sufficiently distant by the respective pole, they are correctly detected as two clusters. However, this will lead to two centroids for the K-means algorithm while it should be only one. These additional centroids will finally randomly cover parts of pants leading to the above-explained count of additional False Positives (FP^{+}).

For these reasons, it is of fundamental importance to develop a new trunks detection procedure that exploits semantic information such as the color.

Chapter 7

Conclusions And Future Work

7.1 Conclusions

This work has been motivated by the lack of automatic systems that could isolate single plants from a tridimensional crop model for phenotyping purposes. Having the model of a crop at the level of every single plant is of fundamental importance to track the plant growth and to target localized treatments. We presented the first building block for an innovative data analysis and modeling framework for the field of automatic plant phenotyping. The final goal is to plot the phenotyping parameters and analyze their evolution during a plant entire lifetime. Increasing the granularity of phenotyping data, from groups of plants to individual plants, will bring new information to be exploited in phenomics studies.

We started our work investigating the literature to understand what are the typically measured phenotyping parameters and commonly adopted sensors. Furthermore, we reviewed the different kinds of platforms that could be exploited for automatic plant phenotyping. As we did not find a complete and standard classification of the various approaches, we proposed a new one. We explained the rationale behind our taxonomy with the hope that the scientific community will widely adopt it.

Then, we gathered a set of cheap, compact, and flexible devices for 3D imaging. We aimed to discover the minimal and cheapest combination of sensors that can still provide useful information. Thus, we tested both indoor and outdoor a variety of sensors. Then, we made a qualitative comparison based on quantitative technical details and experimental results. Thus, after some refinements, we came up with a flexible sensors suite that can be easily mounted on any platform, from robots to tractors. Our system enables high-throughput phenotyping as the only limit is the sensors working frequency.

After sensors evaluation, we put developed a modular software design to which it is possible to plug in and plug out components easily. Thus, the resulting software architecture is flexible and quickly expandable and adaptable to different needs. We realized a structure constituted of three main components: the Data Collection component responsible for recording data; the Reconstruction component, further divided in three subcomponents (Odometry, Visual Reconstruction, and LiDAR Reconstruction); and the Segmentation component.

The odometry subcomponent has been designed with the possibility to test different sources of odometry. Thus, we evaluated four odometry approaches in outdoor natural environments. After visual inspection of experimental results, we declared the Visual Odometry approaches (the plain one and the fusion with the GPS) as the most accurate. Surprisingly, we observed the poor quality of laser odometry, although it is usually considered the most accurate in indoor settings. Likely, the lack of human-made flat surfaces in outdoor natural environments causes degradation of performance.

The other two subcomponents, namely, the Visual Reconstruction and the LiDAR Reconstruction, are responsible for building colored and laser scans point clouds, respectively. The Visual Reconstruction module employes a graph-based SLAM algorithm to build the environment map, while the LiDAR Reconstruction module assembles laser scans based on odometry poses. Thus, the quality of the LiDAR assembled point clouds directly depends on the odometry accuracy. Indeed, Visual Odometry approaches led to high-fidelity point clouds. Likewise, the colored point clouds from the Visual Reconstruction module, accurately represent the plants in term of size, shape, and colors.

Finally, we developed an algorithm that, taken as input a LiDAR point cloud, can isolate single plants. The algorithm showed good performance to recognize plants, while it needs improvements on the detection of non-plant elements. The working principle is to segment plants via K-means clustering algorithm. The clustering algorithm initialization widely influences the accuracy of the segmentation task. We initialized the algorithm with trunks locations. Indeed, we developed a trunks detection procedure exploiting geometric information. Moreover, we devised a strategy to filter out the plants supporting poles. However, the trunks detection procedure can be misled by the presence of extraneous elements as grass bushes or plants branches. Indeed, it is too strict assuming that, at a lower height from the ground, only poles and trunks are present. Thus, we concluded highlighting the importance of exploiting semantic information.

7.2 Future Work

In this Section, we list the short-term tasks that are the natural consequence of this work (Subsection 7.2.1). Then, in Subsection 7.2.2, we provide to the reader a long-term vision about the future of automatic plant phenotyping.

7.2.1 Short-Term Work

- cameras placement could be improved. Orienting the cameras more toward the center of the row will likely ease the odometry task, thus resulting in higher accuracy. Another advantage of this reposition will be the wider view on the vineyard, thus possibly having a colored reconstruction complete of both canopy and trunks. Finally, an overlapping view of both cameras would be advantageous for camera-to-camera calibration
- the STM Nucleo IMU should be calibrated with more sophisticated methods. An alternative is the camera-IMU calibration method implemented by the Kalibr ROS package[1], as described in [47]. Moreover, a way to estimate the IMU covariance

[1] `https://github.com/ethz-asl/kalibr`

matrix should be investigated. Otherwise, the already calibrated IMU embedded in the D435i camera can be exploited. Finally, it is worth to test purposely designed algorithms for visual and IMU odometry fusion, like OKVIS [48] or MSCKF [49]

- to merge visual and lidar reconstructions, cameras need to be calibrated with the 2D LiDAR. The task is complicated by non overlapping views. However, in [50], the authors proposed a method to automatically estimating the relative pose between a push-broom LIDAR and a camera without the need of an overlapping field of view
- a better integration method for GPS locations in the SLAM graph should be investigated. It is crucial to find a method to add GPS constraints in such a way the graph can be optimized, thus avoiding discreet jumps
- the feasibility of loop closure in large-scale natural environments should be assessed. Loop closures are convenient as they allow the system to be completely independent of the GPS signal
- it should be investigated the feasibility of using the new Intel RealSense T265 camera[2] as the principal source of odometry. Preliminary indoor tests revealed a great potential of this device
- the accuracy of the various odometry approaches should be estimated using an RTK-GPS as ground truth
- as for the Segmentation algorithm is of primary relevance to improve the trunks detection procedure exploiting semantic information like the color
- a more sophisticated algorithm that distinguishes leaves from trunks and branches could be devised
- a merging strategy of LiDAR and colored point clouds should be studied. In this way, the advantages of both reconstructions will be exploited. The resulting point cloud will benefit from the LiDAR reconstruction high accuracy and the additional information of color and increased density of visual reconstruction
- to allow comparisons with state-of-the-art phenotyping methods is crucial to perform a quantitative evaluation of the system. It is fundamental to estimate the throughput and accuracy of the phenotyping parameters and the segmentation algorithm. Moreover, it should be assessed the odometry quality in large-scale natural environments

7.2.2 Long-Term Vision

With the advent of Precision Agriculture we are going in the direction of a fourth agricultural revolution. We envision a future where crop works will be entirely automated by intelligent machines that will carry out all operations, from sowing to harvesting, leading to more efficient use of resources (like water and nutrients) and contamination reduction via a fine-tuned spraying of pesticides. One of the requirements for agricultural machines

[2] `https://www.intelrealsense.com/tracking-camera-t265/`

to accomplish in-field localized operations is to have a digital model of each single crop plant. This framework leads to the idea of *Plant Digital Twin*, that is, the computerized counterpart of a real plant.

Autonomous physical agents and smart sensors will collect data aiming at a 3D geo-referenced model of the entire crop at the level of each single plant. The model will evolve as more passages throughout the crop will be performed, and it will have, associated with each single point (or voxel), information such as color, depth, temperature, spectral information, and fluorescence. The data will be analyzed to recognize and track single plants so to monitor them over time. A semantic segmentation will be possible with the joint use of computer vision and machine learning algorithms, e.g., via [51]. Finally, the variables that prove to be relevant from phenomics studies [52] will be inferred from the reconstruction. Having the digitalized representation of a plant, i.e., the Digital Twin, will allow us to monitor and follow it during its entire life giving birth to a new concept that is the agricultural counterpart of the Internet of Things, the Internet of Plants. Think about the potential of a network of plants that share information. What if an infested plant could alert its neighbors to receive treatments hence blocking a pest diffusion?

I foresee critical challenges for years to come. We need to find a way to manage the massive quantity of phenotyping data that will be produced. New modeling techniques and data management approaches should be investigated. Moreover, new advanced analysis tools and prediction models, even beyond the usual statistical methods, are necessary [2]. Great effort should be made to understand the socio-economic impact derived from the adoption of these technologies. The phenotyping data can be used as input for a data-driven DSS (Decision Support System) supporting farmers in the decision-making process. With the help of such information, farmers can optimize returns on inputs usage. For example, a variable-rate application of pesticides and fertilizers can reduce the cost deriving from the inputs and have a benefit on the environment, limiting harmful runoff into waterways. Therefore, it is of fundamental importance to investigate the relationships among the adoption of phenotyping technologies, the economics aspects affecting farms and the environmental impact consequences [53].

Breakthroughs in automatic plant phenotyping will be possible only with the interaction and collaboration with other fields' experts. Multi-disciplinary teams gathering together engineers, mathematicians, economists, biologists, and geneticists need to be formed. The results that will be produced are a crucial step in the fourth agriculture revolution that will be able to satisfy the increase in food demand.

References

[1] R. T. Furbank and M. Tester, "Phenomics–technologies to relieve the phenotyping bottleneck," *Trends in plant science*, vol. 16, no. 12, pp. 635–644, 2011.

[2] J. L. Araus and J. E. Cairns, "Field high-throughput phenotyping: the new crop breeding frontier," *Trends in plant science*, vol. 19, no. 1, pp. 52–61, 2014.

[3] Food and A. O. of the United Nations, "Global agriculture towards 2050," 2009.

[4] S. M. Pedersen, S. Fountas, H. Have, and B. Blackmore, "Agricultural robots—system analysis and economic feasibility," *Precision agriculture*, vol. 7, no. 4, pp. 295–308, 2006.

[5] R. Qiu, S. Wei, M. Zhang, H. Li, H. Sun, G. Liu, and M. Li, "Sensors for measuring plant phenotyping: A review," *International Journal of Agricultural and Biological Engineering*, vol. 11, no. 2, pp. 1–17, 2018.

[6] J. W. White, P. Andrade-Sanchez, M. A. Gore, K. F. Bronson, T. A. Coffelt, M. M. Conley, K. A. Feldmann, A. N. French, J. T. Heun, D. J. Hunsaker, *et al.*, "Field-based phenomics for plant genetics research," *Field Crops Research*, vol. 133, pp. 101–112, 2012.

[7] N. Virlet, K. Sabermanesh, P. Sadeghi-Tehran, and M. J. Hawkesford, "Field scanalyzer: An automated robotic field phenotyping platform for detailed crop monitoring," *Functional Plant Biology*, vol. 44, no. 1, pp. 143–153, 2017.

[8] T. Mueller-Sim, M. Jenkins, J. Abel, and G. Kantor, "The robotanist: a ground-based agricultural robot for high-throughput crop phenotyping," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3634–3639, IEEE, 2017.

[9] S. Madec, F. Baret, B. De Solan, S. Thomas, D. Dutartre, S. Jezequel, M. Hemmerlé, G. Colombeau, and A. Comar, "High-throughput phenotyping of plant height: comparing unmanned aerial vehicles and ground lidar estimates," *Frontiers in plant science*, vol. 8, p. 2002, 2017.

[10] G. Yang, J. Liu, C. Zhao, Z. Li, Y. Huang, H. Yu, B. Xu, X. Yang, D. Zhu, X. Zhang, *et al.*, "Unmanned aerial vehicle remote sensing for field-based crop phenotyping: current status and perspectives," *Frontiers in plant science*, vol. 8, p. 1111, 2017.

[11] Q. Qiu, N. Sun, Y. Wang, Z. Fan, Z. Meng, B. Li, and Y. Cong, "Field-based high-throughput phenotyping for maize plant using 3d lidar point cloud generated with a "phenomobile"," *Frontiers in plant science*, vol. 10, p. 554, 2019.

[12] A. Shafiekhani, S. Kadam, F. Fritschi, and G. DeSouza, "Vinobot and vinoculer: Two robotic platforms for high-throughput field phenotyping," *Sensors*, vol. 17, no. 1, p. 214, 2017.

[13] A. Kicherer, K. Herzog, M. Pflanz, M. Wieland, P. Rüger, S. Kecke, H. Kuhlmann, and R. Töpfer, "An automated field phenotyping pipeline for application in grapevine research," *Sensors*, vol. 15, no. 3, pp. 4823–4836, 2015.

[14] K. Herzog *et al.*, "Initial steps for high-throughput phenotyping in vineyards," *Australian and New Zealand Grapegrower and Winemaker*, no. 603, p. 54, 2014.

[15] J. Barker III, N. Zhang, J. Sharon, R. Steeves, X. Wang, Y. Wei, and J. Poland, "Development of a field-based high-throughput mobile phenotyping platform," *Computers and Electronics in Agriculture*, vol. 122, pp. 74–85, 2016.

[16] D. Deery, J. Jimenez-Berni, H. Jones, X. Sirault, and R. Furbank, "Proximal remote sensing buggies and potential applications for field-based phenotyping," *Agronomy*, vol. 4, no. 3, pp. 349–379, 2014.

[17] P. Andrade-Sanchez, M. A. Gore, J. T. Heun, K. R. Thorp, A. E. Carmo-Silva, A. N. French, M. E. Salvucci, and J. W. White, "Development and evaluation of a field-based high-throughput phenotyping platform," *Functional Plant Biology*, vol. 41, no. 1, pp. 68–79, 2014.

[18] R. Sanz, J. Rosell, J. Llorens, E. Gil, and S. Planas, "Relationship between tree row lidar-volume and leaf area density for fruit orchards and vineyards obtained with a lidar 3d dynamic measurement system," *Agricultural and forest meteorology*, vol. 171, pp. 153–162, 2013.

[19] L. Busemeyer, D. Mentrup, K. Möller, E. Wunder, K. Alheit, V. Hahn, H. Maurer, J. Reif, T. Würschum, J. Müller, *et al.*, "Breedvision—a multi-sensor platform for non-destructive field-based phenotyping in plant breeding," *Sensors*, vol. 13, no. 3, pp. 2830–2847, 2013.

[20] R. Sui, D. K. Fisher, and K. N. Reddy, "Cotton yield assessment using plant height mapping system," *Journal of Agricultural Science*, vol. 5, no. 1, p. 23, 2013.

[21] A. Comar, P. Burger, B. de Solan, F. Baret, F. Daumard, and J.-F. Hanocq, "A semi-automatic system for high throughput phenotyping wheat cultivars in-field conditions: description and first results," *Functional Plant Biology*, vol. 39, no. 11, pp. 914–924, 2012.

[22] J. Llorens, E. Gil, J. Llop, and M. Queraltó, "Georeferenced lidar 3d vine plantation map generation," *Sensors*, vol. 11, no. 6, pp. 6237–6256, 2011.

[23] V. Vadez, J. Kholová, G. Hummel, U. Zhokhavets, S. Gupta, and C. T. Hash, "Leasyscan: a novel concept combining 3d imaging and lysimetry for high-throughput phenotyping of traits controlling plant water budget," *Journal of Experimental Botany*, vol. 66, no. 18, pp. 5581–5593, 2015.

[24] J. W. White and M. M. Conley, "A flexible, low-cost cart for proximal sensing," *Crop Science*, vol. 53, no. 4, pp. 1646–1649, 2013.

[25] C. J. Bronick and R. Lal, "Soil structure and management: a review," *Geoderma*, vol. 124, no. 1-2, pp. 3–22, 2005.

[26] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pp. 1–6, April 2013.

[27] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[28] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Intelligent autonomous systems 13*, pp. 335–348, Springer, 2016.

[29] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[30] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[31] 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.

[32] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[33] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[34] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling," *Remote Sensing*, vol. 7, no. 10, pp. 13070–13097, 2015.

[35] A. Corti, S. Giancola, G. Mainetti, and R. Sala, "A metrological characterization of the kinect v2 time-of-flight camera," *Robotics and Autonomous Systems*, vol. 75, pp. 584–594, 2016.

[36] T. Bell, B. Li, and S. Zhang, "Structured light techniques and applications," *Wiley Encyclopedia of Electrical and Electronics Engineering*, pp. 1–24, 1999.

[37] C. Nock, O. Taugourdeau, S. Delagrange, and C. Messier, "Assessing the potential of low-cost 3d cameras for the rapid measurement of plant woody structure," *Sensors*, vol. 13, no. 12, pp. 16216–16233, 2013.

[38] J. Llorens, E. Gil, J. Llop, *et al.*, "Ultrasonic and lidar sensors for electronic canopy characterization in vineyards: Advances to improve pesticide application methods," *Sensors*, vol. 11, no. 2, pp. 2177–2194, 2011.

[39] W. Kazmi, S. Foix, G. Alenyà, and H. J. Andersen, "Indoor and outdoor depth imaging of leaves with time-of-flight and stereo vision sensors: Analysis and comparison," *ISPRS journal of photogrammetry and remote sensing*, vol. 88, pp. 128–146, 2014.

[40] A. Vit and G. Shani, "Comparing rgb-d sensors for close range outdoor agricultural phenotyping," *Sensors*, vol. 18, no. 12, p. 4413, 2018.

[41] M. Vázquez-Arellano, H. Griepentrog, D. Reiser, and D. Paraforos, "3-d imaging systems for agricultural applications—a review," *Sensors*, vol. 16, no. 5, p. 618, 2016.

[42] L. Li, Q. Zhang, and D. Huang, "A review of imaging techniques for plant phenotyping," *Sensors*, vol. 14, no. 11, pp. 20078–20111, 2014.

[43] P. Roy, W. Dong, and V. Isler, "Registering reconstructions of the two sides of fruit tree rows," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018.

[44] J. Shi and C. Tomasi, "Good features to track," tech. rep., Cornell University, 1993.

[45] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, IEEE, 2011.

[46] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing icp variants on real-world data sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[47] T. Pire, M. Mujica, J. Civera, and E. Kofman, "The rosario dataset: Multisensor data for localization and mapping in agricultural environments," *The International Journal of Robotics Research*, vol. 38, no. 6, pp. 633–641, 2019.

[48] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[49] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[50] A. Napier, P. Corke, and P. Newman, "Cross-calibration of push-broom 2d lidars and cameras in natural scenes," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3679–3684, IEEE, 2013.

[51] F. Visin, M. Ciccone, A. Romero, K. Kastner, K. Cho, Y. Bengio, M. Matteucci, and A. Courville, "Reseg: A recurrent neural network-based model for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 41–48, 2016.

[52] F. Fiorani and U. Schurr, "Future scenarios for plant phenotyping," *Annual review of plant biology*, vol. 64, pp. 267–291, 2013.

[53] J. Schieffer and C. Dillon, "The economic and environmental impacts of precision agriculture and interactions with agro-environmental policy," *Precision agriculture*, vol. 16, no. 1, pp. 46–61, 2015.

Appendix A

Sensors Technical Specifications And Settings

In this Appendix we provide tables with the technical specifications of the sensors listed in Section 4.1. When different values of a parameter are available, we write the one adopted in the final configuration in bold italic font (e.g., Angular Resolution: ***0.25***°, 0.5°)

ZED Camera	
Name	Value
Frame Rate	15, 30, 60 fps
Video Resolution	***1280x720***
Depth Range	0.5 - 20 m
Stereo Baseline	120mm
Motion 6-axis Pose Accuracy	Position: +/- 1mm; Orientation: 0.1°
Field of View	90°(H) x 60°(V) x 110°(D) max
Sensor Resolution	4M pixels per sensor
Shutter Sync	Electronic Synchronized Rolling Shutter
Interface	USB 3.0
Power	USB 5V / 380mA
Dimensions	175 (W) x 30 (H) x 33 (D) mm
Weight	159g
OS Compatibility	Windows 7,8,10 and Linux

Table A.1: ZED Camera technical specifications

Kinect One	
Name	Value
Frame Rate	30 fps
RGB Image Resolution	1920x1280
Depth Image Resolution	512x424
Depth Range	0.5 - 4.5 m
Field of View	70°(H) x 60°(V)
Interface	USB 3.0
Input Voltage	12V
Dimensions	249 (W) x 66 (H) x 67 (D) mm
Weight	1400g
Operating Environment	Indoor

Table A.2: Kinect One technical specifications

XtionPRO Live	
Name	Value
Frame Rate	***30 fps (VGA)***, 60 fps (QVGA)
RGB Image Resolution	1280x1024 (SXGA)
Depth Image Resolution	320x240 (QVGA), ***640x480 (VGA)***
Depth Range	0.8 - 3.5 m
Field of View	58°(H) x 45°(V) x 70°(D)
Interface	USB 2.0
Input Voltage	5V
Power Consumption	< 2.5 W
Dimensions	177.8 (W) x 48.2 (H) x 38.1 (D) mm
Weight	540g
Operating Environment	Indoor

Table A.3: XtionPRO Live technical specifications

Intel RealSense D435(i) Camera	
Name	Value
RGB Frame Rate	up to 30 fps ***(30 fps)***
Depth Frame Rate	up to 90 fps ***(30 fps)***
RGB Image Resolution	1920x1080
Depth Image Resolution	1280x720
Depth Range	0.105 - 10 m
RGB Field of View	69.4°(H) x 42.5°(V) x 77°(D)
Depth Field of View	87°(H) x 58°(V) x 95°(D)
RGB Sensors Technology	Global Shutter
Interface	USB 3.0
Input Voltage	5V
Dimensions	90 (W) x 25 (H) x 25 (D) mm
Operating Environment	Indoor and Oudoor
D435i Additional Feature	BMI055 IMU (Inertial Measurement Unit)

Table A.4: Intel RealSense D435(i) Camera technical specifications

Sick LMS100-1000 2D LiDAR	
Name	Value
Light Source	Infrared (905 nm)
Aperture Angle	270°
Scanning Frequency	25 Hz, ***50 Hz***
Angular Resolution	0.25°, ***0.5***°
Working Range	0.5 - 20 m
Fog Correction	Yes
Interfaces	***Ethernet TCP/IP***; Serial RS-232; CAN
Input Voltage	10.8 V ... 30 V ***(12 V)***
Power Consumption	8 W
Enclosure Rating	IP65
Dimensions	102 (W) x 152 (H) x 105 (D) mm
Weight	1100g
Operating Environment	Indoor
Ambient Light Immunity	40000 lx

Table A.5: Sick LMS100-1000 2D LiDAR technical specifications

Sick LD-MRS400001S01 3D LiDAR	
Name	Value
Light Source	Infrared (905 nm)
Horizontal Aperture Angle	85°(4 planes), ***110° (2 planes)***
Vertical Aperture Angle	3.2°
Scanning Frequency	***12.5 Hz*** ... 50 Hz
Angular Resolution	0,125°, ***0.25***°, 0.5°
Working Range	0.5 - 300 m
Interfaces	***Ethernet TCP/IP***; Serial RS-232; CAN
Input Voltage	9 V ... 27 V ***(12 V)***
Power Consumption	8 W
Enclosure Rating	IP69K
Dimensions	165 (W) x 88 (H) x 94 (D) mm
Weight	1000g
Operating Environment	Outdoor

Table A.6: Sick LD-MRS400001S01 3D LiDAR technical specifications

Trimble 5700 GPS Receiver	
Name	Value
Measurement Rate	1 Hz, 2 Hz, ***5 Hz***
Interfaces	***Serial RS-232***; USB (only for data download)
Input Voltage	10.5 V ... 28 V ***(12 V)***
Power Consumption	5.9 W
Enclosure Rating	IP67
Dimensions	135 (W) x 85 (H) x 240 (D) mm (receiver only)
Weight	1400g (receiver only)
Operating Environment	Outdoor

Table A.7: Trimble 5700 GPS Receiver technical specifications

X-NUCLEO-IKS01A1 Multi-Sensor Board	
Name	Value
IMU LSM6DS0 Acceleration Range	±2/±4/±8 g
IMU LSM6DS0 Angular Range	±245/±500/±2000 dps
Magnetometer LIS3MDL Magnetic Full Scales	±4/±8/±12/±16 gauss

Table A.8: X-NUCLEO-IKS01A1 technical specifications