POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in
Computer Science and Engineering

Crop Rows Constraints for Agricultural SLAM

Advisor: PROF. MATTEO MATTEUCCI
Co-advisor: ENG. RICCARDO BERTOGLIO

Master Graduation Thesis by:

CHIARA RELANDINI
Student Id n. 914815

Academic Year 2019-2020

ABSTRACT

Agricultural Simultaneous Localization And Mapping (SLAM) is the process of robot self-localization and field mapping in agriculture. Robot localization is the prerequisite for any autonomous robot task. It can exploit a prior environment map if available, in order to extract reference points. Reference points are observable characteristics of the environment that are distinguishable from each other, thus allowing to determine a unique robot position. If a prior map is not available, the robot needs to build a map concurrently with localization, which is the case addressed in this thesis. Besides being a support for robot localization, agricultural mapping can be exploited for several tasks such as crops phenotyping and targeted herbicides spraying.

Localization in the agricultural field is a complex task for systems using both absolute and relative sensor measurements. Absolute localization methods can fail because of signal lack, while relative localization systems have to deal with drift accumulation due to the uneven ground and uniform visual appearance of the surroundings. Uniform visual appearance makes the identification of reference points extremely difficult, especially if we consider that landscape is highly time-varying due to the plants growth and therefore reference points are difficult to track over time. This is the reason why prior maps are seldom available.

In this thesis, we addressed the localization problem by fusing absolute and relative sensor measurements. In particular, GPS messages were used to get absolute position measurements, while wheel odometry and an RGB camera were used for relative trajectory estimation. The RGB camera images were used to detect crop plants Stem Emerging Point (SEP) as reference points for the relative estimation system. We aimed at partially replacing the need for a precise GPS sensor, by providing a reliable trajectory estimation with visual sensor fusion.

Mapping was handled by storing the estimated coordinates of crops SEP in the GPS absolute frame. In addition to single crop plants position, the crop row direction was estimated in the robot local frame. Crop rows constraints were created by interpolating individual crop plants lying along a line, because of the specific domain assumption of crop plants sowed in a row. The mentioned constraints can be used to correct the estimate of the robot trajectory, which must lie at a given distance from the crop row that the robot is following. In particular, for each RGB image we can build a constraint on the distance between the robot pose and the plant centroid obtained by the 3D reconstruction of the SEP identified in the image. For identifying the SEPs in images, we developed a specific image processing component.

Finally, we implemented an error computation function to assess the localization results without absolute information. The method computes the best transformation to align the robot trajectory to the ground truth, before the error computation. This was done in order to evaluate the trajectory optimization results which did not fuse GPS measurements in the estimation and therefore are not absolutely referenced.

SOMMARIO

La localizzazione e la mappatura simultanea (in inglese è abbreviata con l'acronimo SLAM) per i robot agricoli è il processo di auto-localizzazione del robot nel campo agricolo senza una mappa a priori dell'ambiente. I robot autonomi affrontano il problema dell'auto-localizzazione mentre svolgono la maggior parte dei loro compiti. I robot possono utilizzare una mappa a priori o l'identificazione di punti di riferimento mentre si muovono. I punti di riferimento sono caratteristiche dell'ambiente osservabili e distinguibili tra loro, che consentono di determinare una posizione univoca del robot. L'ambiente agricolo pone il problema dell'indisponibilità di punti di riferimento da sottoporre direttamente al sistema di localizzazione e mappatura. Il campo è spesso visivamente ripetitivo e omogeneo, ed è anche mutevole nel tempo, quindi è difficile estrarre punti di riferimento da identificare in modo univoco nel tempo.

In questa tesi, affrontiamo il problema della localizzazione di un robot autonomo in campo agricolo senza una mappa a priori, nel contesto di SLAM, dal momento che l'ambiente è sconosciuto. Valutiamo la possibilità di utilizzare colture e filari di piante come punti di riferimento nel sistema di localizzazione e mappatura. Proponiamo un metodo per identificare le singole piante coltivare, che possono essere utilizzate come riferimento invariante nel tempo se adeguatamente elaborate. Il metodo consiste nell'estrazione del punto emergente del fusto (abbreviato con l'acronimo SEP in inglese) della pianta, caratteristica invariante nel tempo anche quando essa cresce. Le immagini del campo vengono inviate a un sistema che esegue i tre seguenti passaggi: segmentazione delle immagini, clustering e calcolo del centro di massa per ottenere i SEP delle piante coltivate.

Abbiamo progettato un algoritmo di SLAM che prende come input i punti di riferimento costituiti dalle piante coltivate per migliorare la stima della localizzazione e produrre una mappa delle caratteristiche dell'ambiente. Il sistema fonde le misurazioni del GPS, i punti di riferimento delle piante e la stima iniziale della traiettoria data dalla wheel odometry. Come risultato produce una traiettoria ottimizzata, sia online che offline. L'ottimizzazione della traiettoria può funzionare anche senza informazioni GPS, tuttavia il risultato non può essere considerato orientato in modo assoluto. Per valutare le prestazioni del sistema in assenza di informazioni assolute, abbiamo implementato una funzione ad hoc per calcolare l'errore, che gestisce l'allineamento della traiettoria calcolata con il ground truth prima dell'effettivo calcolo dell'errore.

La struttura della tesi è la seguente:

- Il Capitolo 2 presenta la revisione della letteratura dei sistemi che eseguono la localizzazione e la mappatura per i robot autonomi e dei metodi di identificazione dei punti di riferimento che possono essere applicati al campo agricolo. Affronta anche la motivazione di questo lavoro e i passi da compiere per progredire nel campo dello SLAM agricolo;

- Il Capitolo 3 fornisce il contesto sia teorico che tecnico per l'implementazione della metodologia proposta da questa tesi;

- Il Capitolo 4 descrive l'insieme di dati che abbiamo usato nei nostri esperimenti, sia per l'addestramento della rete neurale che per la simulazione di dati reali nell'algoritmo di SLAM. Include anche la descrizione degli strumenti software che abbiamo utilizzato per sviluppare l'algoritmo;

- Il Capitolo 5 fornisce una panoramica dell'architettura del sistema e spiega in dettaglio tutti i componenti;

- Il Capitolo 6 riporta i risultati ottenuti dal componente di identificazione dei centroidi e dal componente di SLAM;

- Il Capitolo 7 conclude la tesi riassumendo i risultati ottenuti e proponendo dei possibili miglioramenti.

RINGRAZIAMENTI

Sono giunta alla fine del mio percorso accademico, e guardandomi indietro vorrei ringraziare alcune figure che hanno avuto un ruolo fondamentale nella mia crescita personale e accademica.

Innanzitutto, vorrei ringraziare il professor Matteucci che mi ha guidata in questa tesi e mi ha permesso di approfondire un campo in parte nuovo per me e al contempo di mettere in pratica quei concetti di deep learning che mi affascinano e che mi hanno spinto a scegliere questo percorso. Ringrazio anche Riccardo, che mi ha supportato nel lavoro di tesi e in cui ho visto la passione accademica per la ricerca come una fonte di ispirazione.

Già prima del percorso universitario, ho avuto la fortuna di incontrare insegnanti appassionati che mi hanno trasmesso l'entusiasmo per lo studio. La prima ad avere un ruolo fondamentale sulle mie scelte è stata senz'altro la maestra Angela, facendomi capire che la matematica è un gioco, ma anche accrescendo quella voglia di mettermi in gioco e di dare il meglio che mi ha accompagnato nella crescita. Penso in modo particolare anche agli insegnanti del liceo, che hanno nutrito la mia personalità e mi hanno aiutato a sviluppare un pensiero critico.

Fuori dal mondo accademico, devo senz'altro ringraziare la mia famiglia, senza la quale non sarei la persona che sono oggi e che mi ha sempre incoraggiato e sostenuto in ogni mia scelta. Ringrazio i miei genitori, Luca e Bettina, che mi sono sempre accanto e mi hanno permesso di vivere ogni esperienza. Loro mi hanno insegnato a intraprendere la strada che ritengo giusta, anche se difficile, e mi hanno fatto capire che non lo farò mai da sola. Mia sorella Gaia che mi ha sempre ascoltato e tirato su quando ne avevo bisogno, mi ha accolto a Milano, e alla fine abbiamo anche imparato ad andare d'accordo. Fin da piccole, ci siamo modellate a vicenda, ed è la persona con la quale so di poter condividere tutto senza aver paura del suo giudizio. Ringrazio i miei nonni, quelli che ci sono e quelli che non ci sono più ma che mi sono sempre stati accanto, il cui affetto e orgoglio mi ha spinto a dare il meglio in ogni situazione.

Nella mia esperienza, ho incontrato tante persone che hanno lasciato il segno sul mio modo di ragionare e di vedere il mondo. Mi piace pensare che mi abbiano reso più aperta e più libera nell'affrontare la vita. A partire dai miei colleghi di corso, in particolare Iacopo a Torino, Daniele e Alessandra a Milano, con cui ho superato i momenti difficili e ho festeggiato gli avvenimenti importanti. Un pensiero particolare va a Flavia, che ha condiviso con me il percorso completo, il trasferimento da Roma, la prima volta fuori di casa e in un'altra città. Ringrazio tutte le amiche che mi sono state accanto in

questo percorso, Sara, Sara e Gaia, la cui amicizia è stata fondamentale per superare le difficoltà di questo periodo storico. Sara e Marzia, che mi hanno riportato a quando eravamo bambine, sostenendomi nonostante la distanza. Elisabetta, con la quale ho riscoperto un'amicizia importante anche se lontane e con la quale mi sono potuta confidare come se fossimo sempre state vicine.

Infine voglio ringraziare le persone che forse più di tutte mi sono state accanto durante e alla fine di questo percorso. Ringrazio in modo particolare Paloma, nella quale ho trovato un'amica sincera con cui discutere di qualsiasi argomento, e che mi ha aiutato a superare le mie paure e anche ad affrontare serenamente il mondo del lavoro. E ringrazio Alberto, che ha condiviso questo percorso con me dal primo giorno, mi ha sostenuto e aiutato a sfruttare a pieno i momenti di svago.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

SLAM	Simultaneous Localization And Mapping
BA	Bundle Adjustment
METRICS	Metrological Evaluation and Testing of Robots in International Competitions
ACRE	Agri-food Competition for Robot Evaluation
MRF	Markov Random Field
WO	Wheel Odometry
VO	Visual Odometry
EKF	Extended Kalman Filter
ICP	Multi View Stereo
MVS	Multi View Stereo
SVM	Support Vector Machine
CNN	Convolutiona Neural Network
SEP	Stem Emerging Point
PSEP	Plant Stem Emerging Point
NDVI	Normalized Difference Vegetation Index
RTK	Real Time Kinematic
PPP	Precise Point Positioning
RMSE	Root Mean Square Error
IMU	Inertial Measurement Unit

1 INTRODUCTION

Simultaneous Localization And Mapping (SLAM) for agricultural robots is the process of robot self localization in the agricultural field without a prior map of the environment. Autonomous robots face the self-localization problem while performing most of their tasks. Robots can employ a prior map or the identification of reference points while they move. Reference points are observable features of the environment that are distinguishable from each other, thus they allow to determine a unique robot position. The agricultural environment raises the issue of the unavailability of reference points to be directly fed to the localization and mapping system. The field is often visually repetitive and homogeneous, and it is also mutable in time, so it is difficult to extract landmarks to be uniquely identified over time.

In this thesis, we address the issue of localizing an autonomous robot in the agricultural field without a prior map. The context is the SLAM problem, since the environment is unknown. We explore the possibility of using crop plants and crop rows as landmarks in the localization and mapping system. We propose an identification method for single crops, which can be used as time-invariant landmarks if adequately processed. It consists in extracting the Stem Emerging Point (SEP) of the plant, which is a time-invariant feature even when the crop grows. Field images are fed to a system performing a pipeline of image segmentation, clustering and center of mass computation to obtain the crops SEP.

We designed a SLAM algorithm taking as input the crops landmarks to improve the localization estimation and produce a landmark map of the environment. The system fuses GPS measurements, crops landmarks and the raw trajectory estimation from wheel odometry. As a result, it produces an optimized trajectory, both online and offline. The trajectory optimization can also work without GPS information, however, the result cannot be considered absolutely referenced. To assess the performance of the system in absence of absolute information, we implemented an error function, which handles the prior alignment of the computed trajectory with the ground truth and finally computes the error.

The thesis structure is the following:

- Chapter 2 provides the literature review of systems performing localization and mapping for autonomous robots and of landmarks identification methods which can be applied to the agricultural field. It also addresses the motivation of this work and the steps which are to be made to progress in the agricultural SLAM field;

- Chapter 3 provides both theoretical and technical background for the implementation of the methodology proposed by this thesis;
- Chapter 4 describes the dataset we used in our experiments, both for the neural network training and for simulating real data in the SLAM algorithm. It also includes the description of the software tools we used to develop the algorithm;
- Chapter 5 gives an overview of the architecture of the system and explains in detail all components;
- Chapter 6 reports the results obtained by the centroids identification and the SLAM components;
- Chapter 7 concludes the thesis by summarizing the obtained results and proposing some future works.

2 STATE OF THE ART

In this chapter, we describe the state of the art of Agricultural SLAM. We start by considering the context of the thesis and then analyze some existing projects in localization, mapping and SLAM. Most of them cannot be directly applied to the agricultural framework due to the extremely different environment characteristics, but they represent the starting point for the method proposed in the next chapters.

2.1 MOTIVATION

Robotics is currently being object of great interest. An example of the worldwide attention to this area is the METRICS [5] project. METRICS was designed to organize robotics competitions in four domains: Healthcare, Inspection and Maintenance, Agri-Food, Agile Production, with the goal of competitions being reproducible and having an objective evaluation. The aim of this project is to structure the European robotics research around the four areas and to draw the attention of economics on robotics. It introduces a methodology to evaluate results in the four fields. Reliability is assessed through an evaluation framework based on measurable quantitative metrics. Competitions have both a physical field evaluation and a virtual assessment on datasets.

In particular, the current work finds its place in the context of the **ACRE** (Agri-Food Competition for Robot Evaluation) [6] competition, where autonomous robots perform various agricultural jobs, such as removing weeds. The purpose is to enhance precision agriculture and bring it to Agriculture 4.0, by improving efficiency and sustainability. In addition to performing repetitive and laborious tasks 24/7, robots allow farmers to reduce waste in resources, such as water and fertilizers. ACRE is a benchmarking competition for both researchers and companies, testing the evaluation in the real-world and documenting the processes to obtain repeatable results.

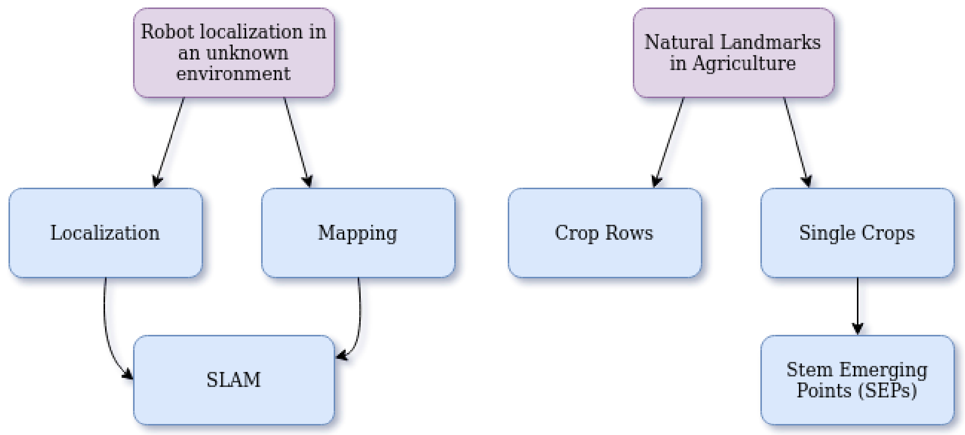

Robot localization in an unknown environment
Localization
Mapping
SLAM
Natural Landmarks in Agriculture
Crop Rows
Single Crops
Stem Emerging Points (SEPs)

Figure 2.1: Subjects investigated in the study of the state of the art. On the left: Localization and Mapping can be performed independently or concurrently, thus building a SLAM system. On the right: In order to create a representation of the unknown agricultural environment, we exploit natural landmarks. These can be either crop rows or single crops, which are well represented by their stem emerging point.

2.2 LITERATURE REVIEW

In this section, we analyze the state of the art in localization, mapping, and SLAM (Simultaneous Localization and Mapping) for mobile robots in the outdoor environment (see Figure 2.1-left). We focus on agriculture oriented works, but also include some papers that explore features that may be considered of interest for the agricultural domain. The related works considered are classified in three categories that may possibly overlap (Localization, Mapping, SLAM), hence our classification takes into account what we consider to be the main focus of the paper and how it could be applied to our use case, even if more features are implemented.

Moreover, we analyze the state of the art for landmarks detection in the agricultural field. The almost complete lack of reference points in the agricultural scenario makes the localization and mapping tasks extremely challenging. Reference points are important because they act as landmarks in the pose optimization problem, raising the accuracy of the estimation. To satisfy the need for reference points, natural landmarks, which can be either the entire crop rows or the individual crops (see Figure 2.1-right), can be identified in the field. The decision on whether considering the entire row or the single plant depends on the application. On the one hand, crop rows are used when guidance lines for navigation systems are necessary. On the other hand, single crop plants are useful in field mapping, plant phenotyping, or herbicide spraying tasks, where crops must be separated from weeds. A key aspect of

Title	Global sensors	Local sensors	Optimization method	Goal	Environment
"An effective multi-cue positioning system for agricultural robotics," [16]	GPS, IMU, digital elevation model	WO, VO, point clouds	g2o	localization	agriculture
"Automatic guidance of a farm tractor along curved paths, using a unique CP-DGPS," [26]	GPS	-	Kalman filter	localization (guidance)	agriculture
"Guidance of a forage harvester with GPS," [27]	RTK-GPS	-	control system experimentally adjusted	localization (path planning)	agriculture
"Accurate localization by fusing images and GPS signals," [31]	GPS	contour action camera	VO: bag of words, custom function	localization	outdoor
"Visual odometry and map fusion for GPS navigation assistance," [37]	GPS	stereo camera	Kalman filter	localization	outdoor
"Agri-Cost-Maps-Integration of Environmental Constraints into Navigation Systems for Agricultural Robots," [32]	GPS	camera/ 2D laser scans	global and local planners	localization (navigation system)	agriculture
"Deployment of a point and line feature localization system for an outdoor agriculture vehicle," [44]	-	WO, laser range finde	EKF with a prior map	localization	agriculture
"Plant detection and mapping for agricultural robots using a 3D LIDAR sensor," [34]	RTK-GPS	3D lidar, WO	transformation from camera poses	mapping	agriculture
"3D plant modeling: localization, mapping and segmentation for plant phenotyping using a single hand-held camera," [38]	-	stereo camera	projective geometry	mapping (phenotyping)	agriculture

"6D SLAM—3D mapping outdoor environments," [39]	-	3D laser range scans, WO	ICP alignment	SLAM (3D Map)	outdoor
"Real-time hierarchical outdoor SLAM based on stereovision and GPS fusion," [36]	GPS	stereo vision	EKF	SLAM (sparse map of landmarks)	outdoor
"An Efficient Multi-Robot 3D SLAM Algorithm," [41]	-	RGB-D images (Kinect)	ORB SLAM (for single robot) + g2o	multi-robot SLAM	outdoor
"Localization and mapping for autonomous navigation in outdoor terrains: A stereo vision approach," [48]	GPS, IMU	stereo vision, WO	linear filter	SLAM	outdoor
"A high efficient 3D SLAM algorithm based on PCA," [40]	-	RGB-D images	(bag of words) RTAB-Map + g2o	SLAM	indoor
"Improved KinectFusion based on graph-based optimization and large loop model," [45]	-	Kinect	ICP + g2o	SLAM	indoor
"Keyframe-based rgb-d slam for mobile robots with visual odometry in indoor environments using graph optimization," [46]	-	RGB-D images	FOVIS + g2o	SLAM	indoor
"Online 3D Mapping and Localization System for Agricultural Robots," [49]	IMU	3D lidar	feature matching on lidar scans	SLAM	agriculture
"SLAM in the Field: An Evaluation of Monocular Mapping and Localization on Challenging Dynamic Agricultural Environment," [52]	GPS	monocular VO	MVS Reconstruction + OpenVSLAM	SLAM	agriculture
"Autonomous navigation with ROS for a mobile robot in agricultural fields," [56]	IMU, GPS	WO, stereo camera, laser scanner	Hector SLAM + EKF + RTAB-Map	SLAM (navigation system)	agriculture

"4D crop monitoring: Spatio-temporal reconstruction for agriculture," [58]	RTK-GPS, IMU	color camera	factor graph (Max A Posteriori)	4D SLAM	agriculture
"Robust visual SLAM with point and line features," [51]	GPS	velodyne, stereo camera	line features based ORB-SLAM	SLAM and BA	outdoor
"Building a 3-D line-based map using stereo SLAM," [60]	-	stereo vision	MRF graph based with lines as landmarks	SLAM and BA	indoor/outdoor

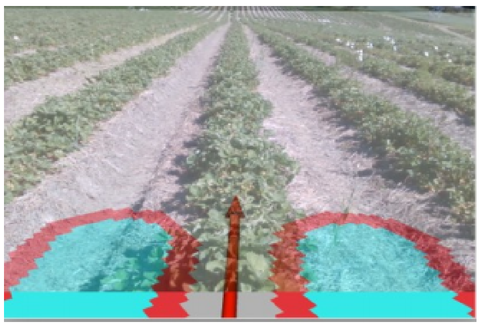
Table 2.1: Analysis of basic features of localization and mapping methods included in the study.

landmarks detection is the visual variance over time. Since both crop plants and crop rows are usually tracked by vision sensors, their visual aspect is the main feature to recognize them. In principle, both methods are extremely time-varying due to the nature of the features, as crops grow over time. However, we note that with some derivations, one method can be considered much more reliable. Crop rows can indeed be easily tracked when the growth stage is advanced, but it becomes difficult to distinguish the line from bare soil when crops have just been sown, although prior knowledge about spatial distribution of crop rows can be exploited. On the contrary, the second approach relies on deriving the stem emerging points (SEP) of the crops in order to identify every single plant, yelding remarkably time-invariant results. The choice of landmarks and application type influence where the camera is placed: if it looks downwards, individual crops are more easily identified, while if it is in front of the robot, it naturally captures the geometric pattern of the rows.

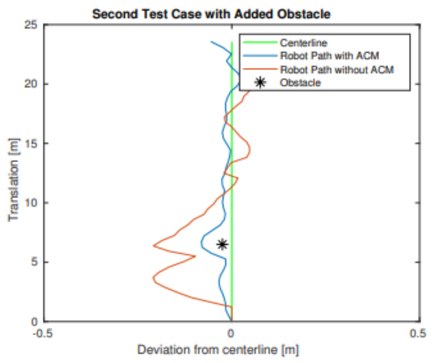
2.2.1 Localization

We distinguish two types of approaches for the localization of mobile robots, based on the type of sensors used. The first relies on absolute information, such as GPS readings. The second is the vision based localization method, which takes advantage of relative information with respect to some reference points. Two different problems arise in these approaches: absolute information provided by GPS measurements is accurate enough only with the more expensive RTK-GPS system, while calculations for relative methods always depend on the previous measurement and the first reference accuracy, and thus are subject to possible drift accumulation. Therefore, a trade-off is necessary to balance the advantages and disadvantages of both methods. On the one hand, global position measurements alone do not allow obstacles to be taken into account, whereas visual information naturally includes knowledge for obstacle avoidance. On the other hand, when visual similarities would lead vision based methods to incorrectly close the loop (roughly, matching a new position with a previously identified one), only absolute information can disambiguate it, even if provided by a consumer-grade GPS, which might be slightly inaccurate but still unambiguous. In this section, we only mention the works relevant to this thesis and provide information on what kind of sensors they use and how they perform trajectory optimization.

The method proposed in [16] described the 3D global pose estimation of a mobile robot in an agricultural field as a pose graph optimization. In particular, the following measurements and estimations were used: GPS and Inertial Measurement Units (IMU) measurements, wheel odometry, visual odometry, point clouds registration. Some domain assumptions were also considered as additional constraints: Ackermann motion model, elevation constraints between adjacent nodes, digital elevation model. They were rearranged in the form of a graph which was fed to the g2o [15] graph optimization framework, which is examined in detail in the next chapter. This approach deserves particular attention because it is one of the few to apply a sensor fusion of both absolute and relative information in a localization algorithm.



(a) Lane detection by local planner.



(b) Trajectory estimation results.

Figure 2.2: Results from paper [32] which shows how the fusion between global (GPS) and local (vision) information obtains a better result with respect to only global information.

As described earlier, GPS localization methods are widely used, as they provide a globally referenced and fairly accurate position estimation. An example is the one in [26] which described the development of a localization method used in order to elaborate a guidance system for an agricultural robot based on GPS to follow curved paths. It used a Kalman gain for correcting the current trajectory with respect to the guidance path. The authors of [27] also used an RTK-GPS in order to precisely locate an agricultural robot and correct the current path. The two just described methods provide interesting examples of localization in the agricultural field, however, the goal of this thesis is to investigate new positioning estimations methods to integrate or substitute the GPS readings, especially focusing on the role of images to correct the relative transformations among robot poses, but few cases are available in the agricultural environment. Therefore, we include an analysis of GPS localization systems used in different outdoor environments which fuse GPS localization with visual measurements. The lack of examples in the agricultural field is due to the particularly challenging environment for visual odometry.

The method used in [31] is an example of using images to correct absolute GPS readings, while using GPS to disambiguate wrong image frames association in case of visual similarities. Visual odometry was implemented as a bag-of-words image feature retrieval problem, searching for correspondences among frames that lied within a given radius from the query image. Namely, each image was GPS tagged in order to restrict the search to only meaningful associations. Also the authors of [37] used stereo camera images to assist GPS localization, by estimating the relative displacement between two 3D frames, especially when the reliability of GPS was considered to be insufficient. Going back to applications in the agricultural domain, we consider the work

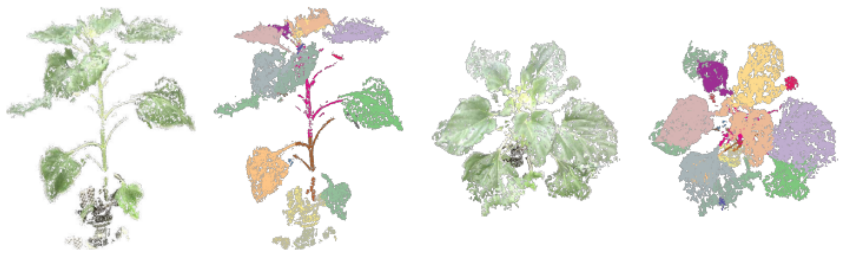


Figure 2.3: Spectral segmentation of reconstructed crop model, side and top views respectively, from [38].

presented in [32] as an important contribution. It implemented a navigation system working on two levels, first developing a global plan based on a static map which used GPS positioning to reach the goal. Then, while navigating, it used a local planner based on vision in order to modify the global plan if obstacles were encountered. The local planner used images in order to detect crop lanes (Figure 2.2a) and fuse direction estimation in the motion commands computation. Results are shown in Figure 2.2b.

Finally, it is interesting to mention the work proposed in [44], where the authors presented a localization method based on both point and line features. They only used a wheel odometry prediction step for EKF (Estended Kalman Filter) and a laser range finder update step. The update step exploited both point and line features. In particular, line features were the tree rows detected in the environment, which acted as natural landmarks. However, the method requires an *a priori* map of the features position, which reduces the generality of the application.

2.2.2 Mapping

In the literature, we can find two representative examples that are specific for the agricultural sector and therefore relevant to this thesis. As we said, methods cannot be sharply classified in one category, but we consider the main purpose of the next presented papers to be the mapping of individual crops, while performing GPS localization mainly to position the plants in the map. The purposes of mapping individual crops in agriculture are multiple: developing a navigation plan without harming plants, using natural landmarks for robot localization, or developing phenotyping applications. The method described in [34] used a 3D LIDAR sensor to reconstruct point clouds of the environment. The method consisted first in detecting the ground to segment the point cloud into soil and other objects, which were then clustered into plants using a prior model of the crop row. Thanks to the fusion of an RTK-GPS, an odometer and an inertial measurement unit, an accuracy of

2 cm was achieved in locating plants. Plants were stored in a global map, after being transformed from robot coordinate frame. The second example of agricultural mapping is described in [38], where the authors used a stereo camera to reconstruct point clouds, instead of a LIDAR sensor. By using a fixed-plant-moving-camera approach, multiple-view stereo images of the plants were collected. Visual local features on plants surface were detected and matched across frames, then they were used to predict the camera pose and the plant's 3D position using projective geometry. Figure 2.3 shows the reconstructed and segmented crop that was later used for phenotyping.

2.2.3 SLAM

There is an extended literature on Simultaneous Localization And Mapping (SLAM), however, application to agriculture is difficult because of the lack of reference points and the uneven ground highly affecting the odometry estimation. Most of them rely on GPS measurements to have accurate position estimates. Here, we analyze both agricultural and non-agricultural SLAM systems, as some methods suggest features that could be applied to our work. SLAM consists in estimating the robot poses while concurrently building a representation of the unknown environment around. In the following paragraphs, we start with outdoor SLAM systems, mention some indoor methods, and conclude with agricultural-specific ones.

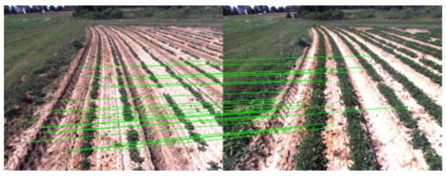
The method in [39] proposed a SLAM system for estimating robot poses as a set of position and attitude, while building a 3D map of the outdoor environment. For mapping, 3D laser range scans, aligned with the ICP (Iterative Closest Point) algorithm and optimized by loop closures were used, whereas odometry estimations were used for robot poses estimation. The system in [36] localized the mobile robot outdoor by sensors fusion between GPS and stereo vision, while creating a sparse map of landmarks. The authors proposed a hierarchical SLAM method, where local submaps (called fingerprints) were created for each reference point. A fingerprint was associated to a mobile robot pose, which defined the local reference frame, and stored visual information for loop closure detection. Local submaps were based on vehicle dead reckoning estimations and were optimized by an Extended Kalman Filter SLAM module. Constraints among fingerprints were eventually considered in the higher level SLAM.

In [41] the authors proposed a multi-robot 3D SLAM, where each robot created its local map that eventually contributed to build the global map. Single robots performed ORB-SLAM [65] with RGB-D images from a Kinect sensor. A local pose graph was built and co-visible portions were optimized by the g2o framework, constraining relative transformation among different robots. At the end, the local maps were fused into a single one. In [48] local and global information fusion was performed, which again used a stereo vision sensor to compute a visual odometry estimation in outdoor terrain, later integrated with an inertial measurement unit and a GPS for global consistency.

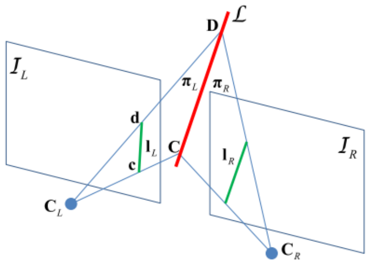
In [40] authors proposed the employment of PCA (Principal Component Analysis) to speed up the feature extraction from RGB-D images. Even if the method is not directly applicable to agriculture because it was proposed for indoor environment, it represents a good inspiration to take from. Extracted features were employed in a bag-of-words algorithm and loop closure was eventually much faster. The SLAM problem was constructed following the RGB-D graph based RTAB-Map (Real-Time Appearance-Based Mapping [69]) approach and optimized using the g2o framework. Visualization data were stored inside the graph nodes. The method in [45] used a Kinect sensor with the ICP algorithm for point cloud alignment and scene reconstruction in indoor environments. Camera poses were predicted by a dense 3D mapping system and compared to those of the ICP method. A g2o framework was used to optimize position and attitude of the robot. Finally, we mention one last indoor SLAM system, [46], which performed graph optimization employing the g2o framework. It collected RGB-D data to create point clouds to be fed to a FOVIS visual odometry [47] to estimate robot poses. Loop closure, through comparison of ORB features, was eventually needed in order to eliminate the accumulated drift from the estimation.

As for agricultural SLAM, a valuable contribution is [49], which described a method for localization and mapping using a 3D LIDAR and an inertial measurement unit. Points in the reconstructed point cloud were preprocessed to separate ground and non-ground points. Non-ground points were clustered, with each cluster representing an object. Pose estimation and loop closure were applied by matching features extracted from LIDAR scans. As an alternative to LIDAR scans, [52] used visual information, by linking a monocular visual SLAM system and a real-time Multi View Stereo (MVS) reconstruction algorithm [67] to build a dense point cloud. Multi-view stereo reconstruction is a process obtaining a 3D representation of objects and scenes models from multiple photographs, in the form of a point cloud or a polygonal mesh, starting from a set of uncalibrated images. For images, depth was predicted in order to simulate an RGB-D sensor. Feature-based visual SLAM was carried out by OpenVSLAM [66]. Additionally, GPS was used for global estimation alignment.

Another example of agricultural SLAM is [56], describing an autonomous navigation system in an agricultural field which simultaneously builds a map of the environment. The localization process used the odometry estimation and the fusion of the measurements from an inertial measurement unit (IMU), a stero camera, a laser scanner and a GPS. Laser scans were converted into odometry estimation by Hector-SLAM [68], while a visual odometry extracted information from a stereo camera images. An Extended Kalman



(a) Data association results between the same rows at 10 days time difference, in Dong et al.



(b) Initialization of line feature by matching two frames, in Zhang et al.

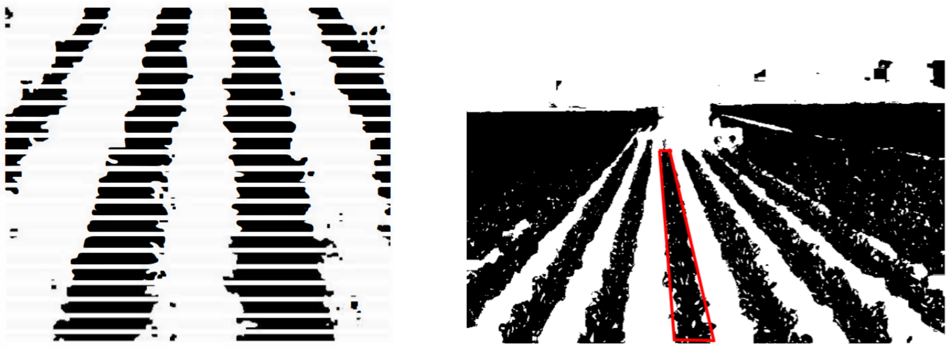
Figure 2.4: Results from different SLAM approaches.

Filter fused all odometry estimations with the IMU, then RTAB-Map used the global odometry information to build a representation of the environment. Optimization of the odometry was carried out only when loop closures occurred.

Finally, [58] is a SLAM system which took care of reconstructing the agricultural field in 4 dimensions. The fourth considered dimension was time, since crop different growth stages implied extremely different visual appearance. The method used visual landmarks for the environment mapping, represented as SIFT [70] features extracted from images. It performed SLAM along each single crop row to estimate robot poses and reconstruct the field structure, by fusing visual information, GPS and IMU. A robust data association was implemented for frame matching at different view points and time instants, as in Figure 2.4a.

Now, we present two works using line features in SLAM system, even if they are not directly applied to agricultural fields, since we are investigating a possible way to use crop lines in agricultural environments as reference points and hence parametrize them as line features. [51] used ORB-SLAM for performing loop closures, local mapping and global bundle adjustment on camera poses and visual landmarks. The method proposed a unified cost function integrating the reprojection errors of both points and lines. It is interesting to notice that lines had two different parametrization: in front-end they were represented by Plücker coordinates, while in back-end the minimal orthonormal representation was preferable. Also, the analytic derivation of the Jacobian of the reprojection error was provided, in order to speed up the computation.

Another method performing SLAM using line features is the one proposed in [60]. Lines were favored over points since they were more informative about the structure of the environment. Again, we notice that lines had a double parametrization for front-end and back-end. An example of line feature detection is shown in Figure 2.4b. It is worth to underline that since



(a) Image divided in strips is then used to cluster the boundary points to detect the lines [54].

(b) Crop rows are detected as quadrangles because of perspective distortion [53].

Figure 2.5: Examples of crop rows detection.

lines were used as features in a visual SLAM and bundle adjustment system, they need to be directly detected in the environment, thus in agricultural environments the interpolation of single crop rows cannot account for a line feature. However, we can process the re-constructed row and feed it to the SLAM optimization in a different way that is more deeply examined in next chapters.

2.2.4 Crop Rows as Landmarks

In the following, we present some methods handling the identification of crop rows to help the mobile robot in the localization task, which are summarized in Table 2.2. Crop rows can be tracked in two ways: by using the 3D structure of the environment or with image based techniques. Crop rows detection methods can be used only when the plants have grown enough to be visible. The first step for most of the systems is image preprocessing or segmentation. Image preprocessing is important in order to highlight the parts of the image which most likely belong to a plant. By analyzing properties such as the green index of each pixel, it is possible to produce a roughly segmented binary image highlighting areas belonging to soil or vegetation.

The system proposed in [30] is a vision based guidance approach for assisting navigation in agriculture by tracking the dominant parallel texture. Assuming crops were planted in roughly straight lines on flat ground, images were segmented into soil and vegetation, then lines were fitted to the binary images. Finally, both camera pose and row pattern were estimated. Camera poses were estimated by means of a Kalman Filter that processed IMU measurements and the visual horizon tracking. While the row heading angle and offset were estimated using a crop template. The authors of [54] described a multi-crop-rows detection method based on strip analysis, which worked

paper	sensors	binary image	identification method	detection
"Vision based guidance for robot navigation in agriculture," [30]	vision	periodic variation of grayscale image	crop row template	row heading and offset
"Multi-crop-row detection based on strip analysis," [54]	vision	threshold on 2G-R-B color space	strip analysis + clustering + least square regression	multi crop rows
"An efficient crop row detection method for agriculture robots," [53]	vision	threshold on 2G-R-B color space	quadrangles as boundary boxes	multi crop rows
"Crop row detection on tiny plants with the pattern hough transform," [61]	laser or RGB images	threshold on height or triangular greeness index	Hough Transform	angle, lateral offset and crop rows spacing
"Guidance line identification for agricultural mobile robot based on machine vision," [57]	monocular vision	threshold on 2G-R-B color space	genetic algorithm	guidance line
"From plants to landmarks: Time-invariant plant localization that uses deep pose regression in agricultural fields," [62]	RGB+NIR images	-	fully convolutional neural network: pose regression	SEP: likelihood map
"Estimating the plant stem emerging points (PSEPs) of sugar beets at early growth stages," [29]	RGB+NIR images	threshold on NDVI	local maxima identification as leaves intersection point	SEP: intra row mechanical weed control
"Optimized EIF-SLAM algorithm for precision agriculture mapping based on stems detection," [33]	range laser + monocular vision	-	SVM	olive stems
"Plant stem detection and position estimation using machine vision," [35]	multispectral images	threshold on NDVI	sliding window Random Forest Classifier	SEP

"Fast and accurate crop and weed identification with summarized train sets for precision agriculture," [42]	RGB+NIR images	CNN: soil vs vegetation	CNN: 3 classes + blobwise voting	crops
"Crop recognition under weedy conditions based on 3D imaging for robotic weed control," [43]	3D-time-of-flight camera	-	point cloud features processing	crops
"Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs," [64]	RGB+NIR images	-	CNN with background knowledge	crops
"An effective classification system for separating sugar beets and weeds for precision farming applications," [59]	RGB+NIR images	-	Random Forest Classifier + MRF smoothing	crops
"Design and development of automatic weed detection and smart herbicide sprayer robot," [55]	RGB	color threshold	pattern recognition on different leaves	crops and weeds (smart herbicide)
"Fuzzy Landmark Detection in Simultaneous Localisation and Mapping for Agricultural Robots," [63]	RGB	-	fuzzy system	landmarks for SLAM

Table 2.2: Analysis of basic features of localization and mapping methods included in the study.

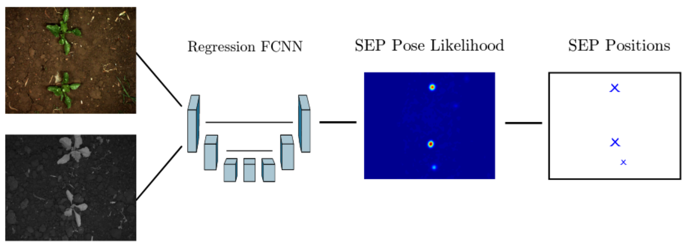
without prior information on world coordinate conversion nor intra-row spacing. The following steps were applied. First, the image was binarized, by setting a threshold on the 2G-R-B color space. Then, it was divided into horizontal strips (Figure 2.5a), the boundary points were clustered and the middle points were chosen to represent the row by least square regression. The same binarization method was also used by the vision based navigation system proposed by the authors of [53]. Based on crop rows detection, it included the image binarization and the usage of boundary boxes to identify the rows. Boundary boxes appeared as quadrangles because of the perspective distortion, Figure 2.5b.

Compared to most crop rows detection methods which rely on the prior knowledge of the field parameters such as the row spacing and heading, the Pattern Hough Transform, presented by the authors of [61], is even more valuable, since it did not need prior information to estimate crop rows pattern. It worked either with laser or RGB data and estimated all parameters, including angle, lateral offset and crop rows spacing, at the same time. The system applied Hough Transform on top of the semantic segmentation of the images of the field, in order to identify the lines corresponding to crop rows. It adapted the Hough Transform for line detection to recognize the entire pattern of parallel equidistant lines. The resulting crop rows pattern was a feature map representing the vegetation probability, defined in 3D local robot coordinates, to be used for robot localization.

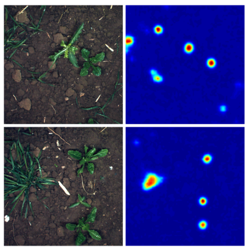
The last crop rows detection method here presented is based on a genetic algorithm to identify the guidance line for an agricultural robot [57]. By using monocular vision system, it used images to randomly select possible solutions (chromosomes) as two points belonging to image top and image bottom, respectively, which would form a line when connected. The chromosome with the highest fitting was selected as crop center line and hence used as guidance line.

2.2.5 Single Crops as Landmarks

An alternative way to identify landmarks is to extract each single crop plant and use it as a reference point for robot localization. In this section we focus on crops and weed segmentation and features extrapolation to recognize landmarks, as summarized in Table 2.2. The most efficient way to represent and locate a single crop as a time-invariant feature is its Stem Emerging Point (SEP). Indeed, crops are subject to comparatively fast growing and since agricultural robots must be able to perform their tasks during all growing stages, they cannot rely on time-varying reference points. Solution is to employ crops SEP as time-invariant points into the localization process. The authors of [62] described a SEPs identification method in the agricultural field to be used as landmarks for localization. The goal is to avoid the usage of GPS while dealing with visual variance of vegetation for achieving high accuracy. SEPs were learned by a fully convolutional network taking as input RGB+NIR images (Figure 2.6a). The network performed pose regression generating a plant location likelihood map (Figure 2.6b), where the plants centroid could be computed as the center of mass.



(a) System architecture: RGB and NIR images are fed to a fully convolutional neural network to produce a SEP pose likelihood map, from which SEP positions are extracted.



(b) Examples of SEP likelihood map for two input images.

Figure 2.6: Plants to landmarks approach described in [62].

A similar approach was depicted by the authors of [29]. Plant Stem Emerging Points (PSEPs) in a sugar beets field were estimated to develop an intra-row mechanical weed control. Precise location of crops was identified by first detecting the leaves, then building the relative PSEP model and finally predicting the true PSEP based on detected leaves. The image was segmented by using the NDVI (Normal Difference Vegetation Index) computed from red and near infrared channels. From the thresholded result, a binary image was produced. Convex regions were identified as candidate leaves and connected components were used to represent plants. Leaves intersection coincided with local maxima (concave regions) in plant boundary, which therefore were assumed to be the PSEP locations.

The authors of [33] developed an Extended Information Filter (EIF) SLAM method for precision agriculture mapping. The main goal was to create a map of the environment through olive stems detection by a range laser sensor for distance information and a monocular vision system. On top of the vision system, a support vector machine for image classification was applied to identify stems. The distance and the angle measurements from the olive stems were fed to the SLAM algorithm to estimate the position of the mobile robot and build a map of the environment. The authors of [35] used a sliding window classifier to predict whether each region in the image contained a plant stem. Only multi-spectral images were used, with no prior segmentation. First, data were acquired and background was removed by NDVI thresholding, then sliding window feature were extracted. The image classification was performed through a random forest classifier to produce a plant stem probability map. Finally, the stem position was estimated, after filtering the probability map with a Gaussian filter.

In order to complete the discussion on landmarks identification, we mention some existing approaches on recognizing crops and weeds. Although they do not include the complete identification of the stem emerging point, we consider them to be a starting point for the SEP identification, which is only a further computation on top of segmentation results. The authors of

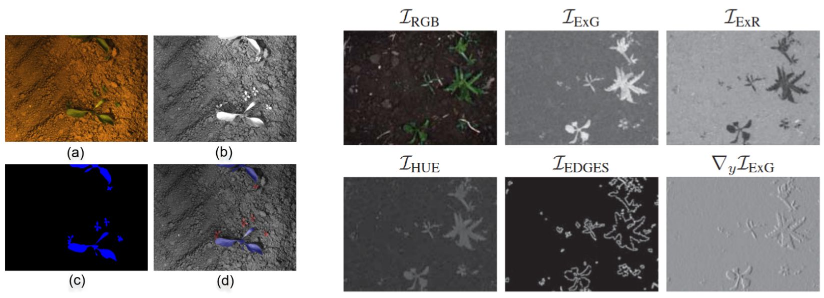

(a)
(b)
(c)
(d)
$\mathcal{I}_{\mathrm{RGB}}$
$\mathcal{I}_{\mathrm{ExG}}$
$\mathcal{I}_{\mathrm{ExR}}$
$\mathcal{I}_{\mathrm{HUE}}$
$\mathcal{I}_{\mathrm{EDGES}}$
$\nabla_y \mathcal{I}_{\mathrm{ExG}}$

Figure 2.7: On the left: Crops images segmented by [42]: (a) RGB image, (b) NIR image, (c) segmentation into vegetation and soil, (d) segmentation into crops, weeds and soil . On the right: Vegetation indexes used as background knowledge fed as additional channels to the convolutional neural network [64].

[42] proposed an accurate crop and weed identification, feeding RGB and NIR images to two convolutional neural network (CNN) in cascade. The first CNN produced a binary image which distinguished soil and vegetation (see Figure 2.7-left). The second CNN produced a 3-classes image segmentation in crops, weeds and soil (to prune the remaining soil pixels). Blob-wise voting was then applied to the classified pixels in order to smooth the result. Classification occurred in real-time. The authors of [43] proposed a crop recognition system under weedy conditions through a 3D time-of-flight camera. From the produced point cloud, 2D and 3D features were extracted and used to identify the crop with almost complete shape in real time. The method in [64] consisted in a real time semantic segmentation of crops and weed leveraging background knowledge such as vegetation indexes computed on RGB images (see Figure 2.7-right). A convolutional network performed the semantic segmentation using shared indexes for upooling layers in order to lower the number of parameters, thus re-training could be performed with relatively few data and in a short time.

Approaches not based on convolutional neural networks also exist. [59] described a random forest classification system for separating weeds and sugar beets based of features computed from RGB and NIR images. Later, the output was smoothed through Markov random field to improve the individual pixel classification. [55] developed a smart herbicide targeting weeds after having separated them from crops. First, vegetation was identified by setting a threshold on the image pixel values, and then, the identification of weeds against crops was possible thanks to the assumption that weeds had broader leaves compared to crops (specific domain assumption), so pattern recognition could be applied. Last, we cite the method in [63] which presented a fuzzy system evaluating landmarks based on measurement error, amount of dynamic nature and the need of a landmark. The result was the extraction of the landmark degree of uncertainty, which was used for identifying the best landmarks to be fed to a Kalman Filter in a localization and mapping system.

2.3 GAP ANALYSIS

Several systems on localization and mapping have already been studied and implemented. The goal of this thesis is to focus on the particular solution that the agricultural environment needs in order to address the specific domain problems. Typical issues that mobile robots need to solve in order to perform outdoor localization are: the identification of recognizable features in the environment, and the potential loss of GPS signal. Indeed, in the most successful systems, data from RTK (Real Time Kinematic) GPS is usually treated as the most reliable and accurate position estimation. Nevertheless, it is an expensive localization method, thus other GPS positioning methods, such as PPP (Precise Point Positioning), tend to be more commonly used.

As regards the identification of reference points, the localization in agricultural fields faces a challenging task. The visually uniform and almost homogeneous environment, and the remarkably tiny dimension of crops in early growing stages make the identification of landmarks very difficult. At the same time, crop landmarks are time-varying even in a small period of time, which makes the chance to re-use the identified landmarks difficult without further processing. Another issue is that odometry estimation of mobile robots is highly affected by noise due to the uneven ground. If the measurement is not corrected by a more accurate estimation, the cumulative drift grows and makes the estimated position unreliable.

For these reasons, there is need to find a new way to improve the mobile robots position estimation. An alternative way to correct the odometry estimation is needed, in order to substitute the expensive RTK GPS sensors. PPP GPS can be used in its place, if used in conjunction with other localization information. Domain assumptions can be inserted in the system in order to enhance the estimation. For example, in our work we exploited the fact that crops are sowed along approximately parallel lines, thus the trajectory of the robot should be rectified by taking the crop rows as reference points.

Furthermore, plants as landmarks deserve a more in-depth investigation. As the analysis of literature review has shown, the most successful method consists in using the stem emerging point of each plant in order to uniquely identify the landmark. However, the optimization of the localization and mapping estimation requires the possibility of recognizing the same landmark when observing it a second time. For this purpose, a visual object recognition component would be needed in the system, that can recognize the same plant when going back at it.

3 BACKGROUND

3.1 THEORETICAL BACKGROUND

In this section, we give same theoretical background for the thesis. First, we describe the graph-based Simultaneous Localization and Mapping (SLAM) as it is the general optimization framework used for the state of the art of robot SLAM. Then we briefly describe the bundle adjustment technique, which can be used to estimate the camera pose when vision sensors are used. Therefore, when the camera is mounted on a robot the technique can be employed to refine the localization estimated by the robot SLAM approach. We also include theoretical background on the pinhole camera model in order to explain how images were transformed into 3D measurements to be integrated in the robot localization estimation. Finally, theory on projective geometry is discussed, since we used it to describe points and lines in the 3D space.

3.1.1 Graph-Based SLAM

Simultaneous Localization and Mapping (SLAM) [2] is a typical issue that mobile robots face in order to operate in unknown environments. In the SLAM framework, robots explore the surroundings without a known map, while building a representation of the map and their relative position and orientation (pose). Prior maps are seldom available, since the environment might change over time, both indoor and outdoor. This is the reason why we need to estimate the map concurrently with localization. One first solution to the localization problem might be the employment of a GPS sensor. However, consumer-grade GPS are rather imprecise. Therefore when centimeter accuracy is needed we have either to employ a sophisticated RTK GPS or to fuse different sensors measurements in the SLAM framework.

There exist two SLAM approaches: filtering and smoothing. Filtering refers to the online state estimation, where the state consists of the current robot pose and the map, while smoothing includes the robot's full trajectory and relies on least square error optimization. Nowadays, graph-based formulation of the SLAM problem is considered the state of the art. It was first formulated by Lu and Milos [1] in the form of building a map by globally optimizing the system to reduce the error given by constraints. It is now feasible and efficient thanks to the advancements in sparse linear algebra which fit the structure of the problem. Sensor measurements insert noise in the system, therefore

SLAM problem formulations needs a probabilistic description. Hereafter, we describe the typical SLAM problem formulation and its assumptions.

Solving the SLAM problem means estimating the robot trajectory and the map of the environment as the robot moves, also said as learning a map under pose uncertainty. We assume that the environment is unknown and that the trajectory can be described by a series of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$. The robot collects a series of odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, ..., \mathbf{u}_T\}$ and perceptions of the environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, ..., \mathbf{z}_T\}$. The full SLAM solution is the posterior probability of the map $\mathbf{m}$ and the trajectory $\mathbf{x}_{1:T}$, given all measurements and the initial position $\mathbf{x}_0$:

$$p\left(\mathbf{x}_{1:T}, \mathbf{m} | \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0\right) \tag{3.1}$$

Poses and odometry are usually represented as transformations in the SE(2) or SE(3) groups, the latter for 3D SLAM. Among the several ways to represent a map, we used the landmark map, which is a collection of landmarks position in the environment, measured with respect to the robot poses.

In order to keep the formulation simple and accurate, we adopted the static world assumption and the Markov assumption. The Markov assumption states that if the robot location is known, future measurements are independent of past ones (and vice versa). In other words, the robot location is the only state in the environment, and knowing it is all what we need to know about the past to predict future data. The static world assumption is needed since the Markov assumption is valid only if the environment just contains static objects beyond the robot.

In the graph-based formulation, nodes are robot poses and landmarks, while edges are spatial constraints derived from observations and odometry measurements. A constraint is a probability distribution over the relative transformations between two poses. Transformations can be the odometry measurements between consecutive poses or the alignment between the observations at those poses.

Once the graph is built, we seek for the configuration that maximizes the compliance to the constraints. Thus we can see the problem as divided in two phases: graph construction and graph optimization. These tasks are usually solved by front-end and back-end, respectively, but they need to be interleaved in order to improve the intermediate results of one another. We describe the back-end in the following paragraphs.

The graph building phase consists in assigning the measurements to the edges and setting poses as the nodes. Edges are labeled with virtual measurements which represent the probability distribution over the relative locations of the two connected poses, built from the raw sensor measurement. To avoid combinatorial explosion, only the most likely constraint from an observation is selected. This task is known as data association.

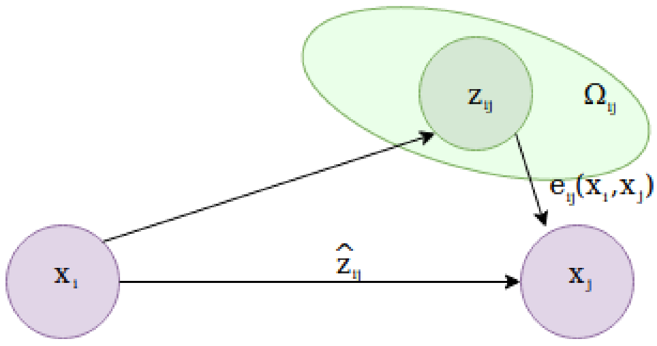


Figure 3.1: Example of measurement among robot poses represented by $\mathbf{x}_i$ and $\mathbf{x}_j$. The expected measurement is $\hat{\mathbf{z}}_{ij}$, while $\mathbf{z}_{ij}$ is the actual measurement and $\mathbf{e}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is the error among them. Ω_{ij} is the information matrix of the constraint.

In the following analysis, we assume that observations are locally affected by Gaussian noise and data association is known. The goal is to find the configuration of the nodes that maximizes the likelihood of the observations, hence to compute the Gaussian approximation mean of the posterior over the robot trajectory.

Now, we describe the graph in more detail. It is composed by a set of nodes: each node i representing the pose $\mathbf{x}_i$, as shown in Figure 3.1. For each measurement we have the mean $\mathbf{z}_{ij}$, the information matrix Ω_{ij} between node i and node j, and the corresponding predicted measurement $\hat{\mathbf{z}}_{ij}$. Hence the error is computed as the difference between actual measurement and expected measurement:

$$\mathbf{e}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j\right), \tag{3.2}$$

and the log-likelihood as:

$$l_{ij} \propto \mathbf{e}_{ij}^T\left(\mathbf{x}_i, \mathbf{x}_j\right) \Omega_{ij} \mathbf{e}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{3.3}$$

which is maximized when we minimize the negative log-likelihood of all observations:

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j\rangle \in C} \mathbf{e}_{ij}^T\left(\mathbf{x}_i, \mathbf{x}_j\right) \Omega_{ij} \mathbf{e}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{3.4}$$

and solve the following equation:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}). \tag{3.5}$$

As it is described in [2] the solution to the SLAM problem is found with an iterative error minimization approach via local linearization. Starting from an initial solution $\breve{\mathbf{x}}$ of the robot poses, we apply Gauss-Newton algorithm, or its Levenberg-Marquardat variation. The error is linearized by its first order Taylor expansion around $\breve{\mathbf{x}}$

$$\mathbf{e}_{ij}\left(\breve{\mathbf{x}}_i + \Delta\mathbf{x}_i, \breve{\mathbf{x}}_j + \Delta\mathbf{x}_j\right) = \mathbf{e}_{ij}\left(\breve{\mathbf{x}} + \Delta\mathbf{x}\right) \simeq \mathbf{e}_{ij}\left(\breve{\mathbf{x}}\right) + \mathbf{J}_{ij}\Delta\mathbf{x}, \tag{3.6}$$

where $\mathbf{J}_{ij}$ is the Jacobian of $\mathbf{e}_{ij}\left(\breve{\mathbf{x}}\right)$. Thus the error terms of $\mathbf{F}\left(\mathbf{x}\right)$ in (3.4) have a quadratic form, and we can rewrite the function as

$$\mathbf{F}\left(\breve{\mathbf{x}} + \Delta\mathbf{x}\right) \simeq c + 2\mathbf{b}^{\mathsf{T}}\Delta\mathbf{x} + \Delta\mathbf{x}^{\mathsf{T}}\mathbf{H}\Delta\mathbf{x}, \tag{3.7}$$

which can be solved as

$$\mathbf{H}\Delta\mathbf{x}^{*} = -\mathbf{b}. \tag{3.8}$$

The matrix $\mathbf{H}$ is the information matrix of the system and it is sparse by construction. The only non-zero blocks are the ones relative to the variables connected by some constraints. Therefore we can solve the system by Cholesky factorization.

The linearized solution is

$$\mathbf{x}^{*} = \breve{\mathbf{x}} + \Delta\mathbf{x}^{*}. \tag{3.9}$$

The Gauss-Newton algorithm iterates this linearization procedure and uses the previous solution as initial guess for the next iteration. Levenberg-Marquardat instead includes a damping factor to control the convergence of the solution.

An additional consideration is that not only the matrix $\mathbf{H}$ is sparse, but even the blocks $\mathbf{H}_{ij}$ corresponding to the constraints are sparse. In fact, they are computed starting from the Jacobian matrix $\mathbf{J}_{ij}$ of each edge, which is sparse itself, as the error only depends on the two involved nodes and its derivative goes to zero for all other nodes.

3.1.2 Bundle Adjustment

Bundle Adjustment (BA) is defined as refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameters (camera pose and/or calibration) estimates. Camera calibration is already given, so we refer to BA only to reconstruct the position of the landmarks while optimizing the robot poses at the same time. Bundle Adjustment is a nonlinear least squares method and can be solved by means of the same tools that solve the SLAM problem (g2o) and applying the same Gauss-Newton algorithm. Thus, we do not enter in details.

To have a visual example, we provide the explanation of a bundle adjustment example joined with a SLAM problem in Figure 3.2.

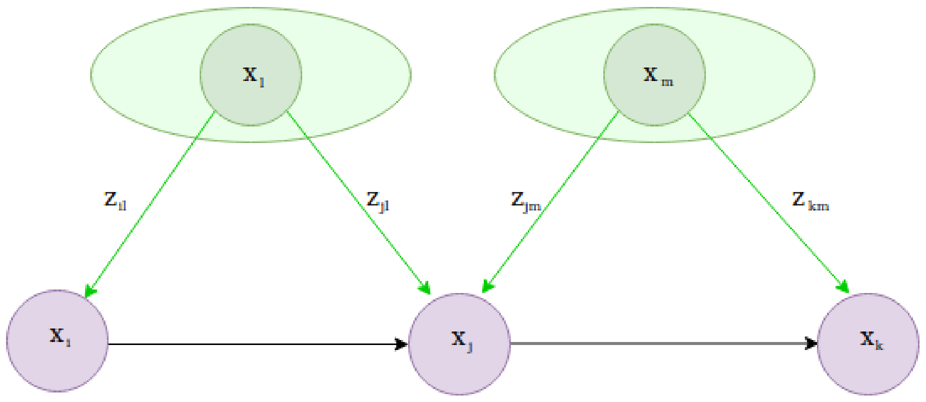


Figure 3.2: Example of bundle adjustment represented as a graph. The camera poses are the nodes in violet ($\mathbf{x}_i$, $\mathbf{x}_j$, $\mathbf{x}_k$), while landmarks are represented as green nodes ($\mathbf{x}_l$, $\mathbf{x}_m$). The measurement $\mathbf{z}_{il}$, $\mathbf{z}_{jl}$, $\mathbf{z}_{jm}$, $\mathbf{z}_{km}$ are the projection to image plane of the 3D lanmark points. The expected measurement is the corresponding pixel in the image. The error is the difference between them. In classical bundle adjustment the camera poses are not constrained among them, but since we are doing SLAM at the same time, those edges correspond to the odometry constraints once the camera pose is transformed to robot pose.

3.1.3 Camera Model

In this section, we describe the camera model of a pinhole camera, which is the one we used in our work. The pinhole camera [4] projects 3D point onto a 2D image plane trough a point **n**, which we call camera center. This model is called central projection. The transformation from a 3D point to pixel coordinates is described by the projection model which is encoded into a 3x4 matrix. By multiplying the matrix by the homogeneous coordinates of the 3D point we have the corresponding homogeneous coordinates of the 2D point on the image plane. This transformation is called forward projection, while the reverse is called backward projection.

The projection is applied in the camera centered coordinate frame, (see Figure 3.3). It is described by three axes that intersect at the camera center, the first two lying on the principal plane which is parallel to the image plane and contains the camera center. The third one, which is called the principal axis, is normal to the principal plane and goes through the camera center and the principal point which is the projection of the camera center on the image plane. The distance between the image plane and the principal plane is the focal length, and the projection of the camera center on the image plane is the principal point.

The projection matrix (3.10) stores the intrinsic parameters, which are determined by the calibration procedure. These are the focal length f, the 2D

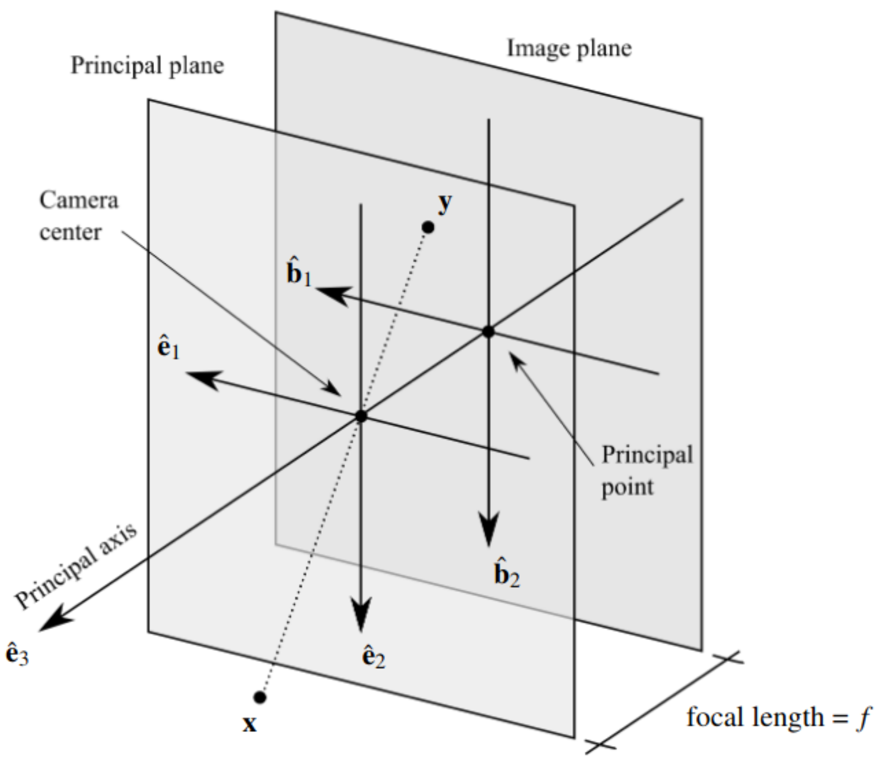


Figure 3.3: The camera centered coordinate system from [4].

coordinates of the principal point o_x, o_y, the scale factors s_x, s_y, and some pixel noise γ.

$$\mathbf{K} = \begin{bmatrix} f/s_x & \gamma & o_x & 0 \\ 0 & f/s_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.10}$$

While the forward projection is a simple matrix multiplication, the backward projection is a more elaborated process. Indeed, if we invert the projection matrix and we multiply it by the homogeneous coordinates of the 2D pixel point, we obtain a 3D point at infinity (we see more in next section). The point at infinity represents the direction, along which the 3D point corresponding to the pixel is located. The line represented by this direction and going through the camera center is called viewing ray (Figure 3.4) and its Plücker matrix (3.16) can be obtained easily in projective geometry. To obtain the precise coordinates of the 3D point, the viewing ray needs to be intersected with the ground plane. The procedure is needed because the model has only one observation point, so triangulation cannot be used as in stereo cameras.

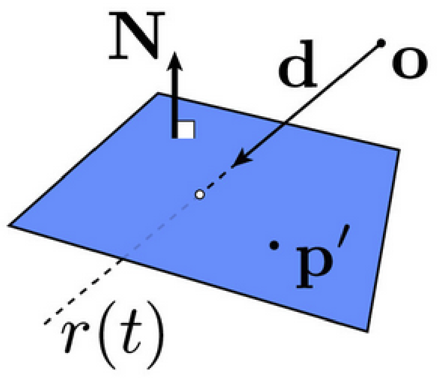


Figure 3.4: The viewing ray has direction **d**. **N** is the normal vector representing the ground plane. We are looking for the point of intersection between **r**(**t**) and **p**′, where **r**(**t**) is the viewing ray starting from the camera center **o** and going through the pixel point and **p**′ is the ground plane where the 3D object lies. (Image from [22])

The 3D point obtained by backward projection is in camera centered coordinates system. The extrinsic matrix (3.11) encodes the change of coordinate frames for points in world coordinates to camera coordinates. The vector **t** can be interpreted as the position of the world origin in camera coordinates, and the columns of **R** represent the directions of the world-axes in camera coordinates.

$$\begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{o}^T & 1 \end{bmatrix} \tag{3.11}$$

3.1.4 Projective Geometry

Homogeneous Coordinates in 3D

The homogeneous coordinates are a system of coordinates used in projective geometry, as the Cartesian coordinates are used in Euclidean space. A point (x, y) in Euclidean coordinates can be represented by a set of homogeneous coordinates in the form $(wx, wy, w), w \in \mathbb{R}, w \neq 0$. For example, given a point $\bar{\mathbf{y}}$ in the Euclidean space, the homogeneous coordinates in canonical form are

$$\mathbf{y} \sim \begin{pmatrix} \bar{\mathbf{y}} \\ 1 \end{pmatrix}. \tag{3.12}$$

This is actually an equivalence class. In fact, the homogeneous coordinates can be multiplied by any non zero scalar, still representing the same point in Euclidean space. Points in projective geometry can be classified as finite points or points at infinity. The advantage of using this kind of representation is that even points at infinity can be represented by finite coordinates. A point at infinity has its last element equal to '0'. It is the representation of a tangent vector, or equivalently of a direction, and thus a line.

To obtain the canonical form of an homogeneous point, we apply P-Normalization. It consists in dividing all elements of the homogeneous vector by the last element, in order to obtain a '1' in the last position. From the P-normalized homogeneous coordinates, it is sufficient to cut the last element in order to get the original Cartesian coordinates of the point.

The canonical dual homogeneous coordinates of a plane in projective geometry are defined based on the plane characterization in $\mathbb{E}^3$, which is defined by a point lying on the plane and the normal vector to that point. If we call the normal vector $\hat{\mathbf{p}}$ and assume the distance of the point from the origin is Δ, the dueal homogeneous coordinates of the planes are:

$$\mathbf{p} \sim \begin{pmatrix} \hat{\mathbf{p}} \\ -\Delta \end{pmatrix} \tag{3.13}$$

If a point $\mathbf{x}$ belongs to a plane $\mathbf{p}$, we can write $\mathbf{x} \cdot \mathbf{p} = \mathbf{x}^T \mathbf{p} = 0$.

To obtain the canonical form of the dual homogeneous coordinates of a plane, we apply D-Normalization. Given a vector $\mathbf{p} \in \mathbb{R}^4$, it consists in making explicit the parameters representing the normal vector $\hat{\mathbf{p}}$ and the distance Δ from the origin. It is computed as:

$$\text{norm}_D \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \frac{-\text{sign}(p_4)}{\sqrt{p_1^2 + p_2^2 + p_3^2}} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} \tag{3.14}$$

Plücker coordinates

A line in $\mathbb{E}^3$ is defined uniquely by two points in the space. In projective geometry, we can allow one of the two points to be at infinity and in fact represent the direction of the line. Therefore, we can write the parametric representation of the 3D line as

$$\mathbf{x}(s) \sim \mathbf{x}_0 + s\mathbf{t} = \begin{pmatrix} \bar{\mathbf{x}}_0 \\ 1 \end{pmatrix} + s \begin{pmatrix} \hat{\mathbf{t}} \\ 0 \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{x}}_0 + s\hat{\mathbf{t}} \\ 1 \end{pmatrix}. \tag{3.15}$$

An alternative representation of a 3D line is given by the Plücker coordinates. They represent the 2-dimensional subspace of $\mathbb{R}^4$ spanned by the homogeneous coordinates of all points that lie on the line [4]. First, we need to obtain the Plücker matrix $\mathbf{L}$, a 4x4 anti-symmetric matrix:

$$\mathbf{L} \sim \mathbf{x}_1\mathbf{x}_2^T - \mathbf{x}_2\mathbf{x}_1^T \tag{3.16}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the homogeneous coordinates of two points on the line.

The L-Normalization is:

$$\text{norm}_L(\mathbf{L}) = \frac{\mathbf{L}}{||\mathbf{L}_{1:3,4}||} = \begin{bmatrix} \mathbf{A} & -\hat{\mathbf{t}} \\ \hat{\mathbf{t}}^T & 0 \end{bmatrix} \tag{3.17}$$

where $||\mathbf{L}_{1:3,4}||$ is the norm of the first three elements in the fourth column of $\mathbf{L}$.

Another convenient representation of the line is the Dual Plücker Matrix $\widetilde{\mathbf{L}}$ which is obtained as:

$$\widetilde{\mathbf{L}} \sim \mathbf{p}_1\mathbf{p}_2^T - \mathbf{p}_2\mathbf{p}_1^T \tag{3.18}$$

where $\mathbf{p_1}$ and $\mathbf{p_2}$ are two perpendicular planes.

In order to extract $\widetilde{\mathbf{L}}$ from $\mathbf{L}$, or vice versa, we can exploit the relation given by the following definition. Since $\mathbf{L} \cdot \widetilde{\mathbf{L}} = 0$ and

$$\mathbf{L} \sim \begin{bmatrix} 0 & a & b & c \\ -a & 0 & d & e \\ -b & -d & 0 & f \\ -c & -e & -f & 0 \end{bmatrix} \tag{3.19}$$

we obtain:

$$\widetilde{\mathbf{L}} \sim \begin{bmatrix} 0 & f & -e & d \\ -f & 0 & c & -b \\ e & -c & 0 & a \\ -d & b & -a & 0 \end{bmatrix}. \tag{3.20}$$

The DL-Normalization is:

$$\text{norm}_{DL}\left(\widetilde{\mathbf{L}}\right) = \frac{\sqrt{2}}{||\widetilde{\mathbf{L}}_{1:3,1:3}||_F}\widetilde{\mathbf{L}} = \begin{bmatrix} \mathbf{A} & -\mathbf{b} \\ \mathbf{b}^T & 0 \end{bmatrix} \tag{3.21}$$

where $||\widetilde{\mathbf{L}}_{1:3,1:3}||$ is the Frobenius norm of the upper left 3x3 block of $\widetilde{\mathbf{L}}$.

From the Plücker matrix we can finally obtain the Plücker line. By rewriting $\mathbf{L}$ as

$$\mathbf{L} \sim \begin{bmatrix} [\mathbf{n}]_\times & \mathbf{v} \\ -\mathbf{v}^T & 0 \end{bmatrix} \tag{3.22}$$

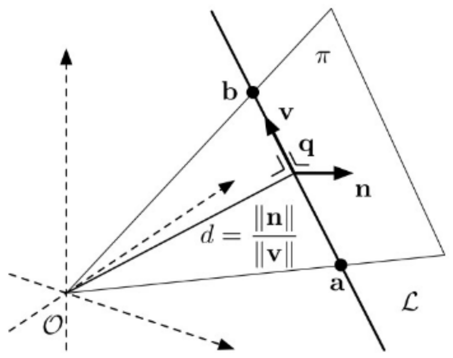


Figure 3.5: The Plücker line $(\mathbf{n}, \mathbf{v})^T$, from [23]. The line lies on the plane π with normal vector $\mathbf{n}$ starting from point $\mathbf{q}$ on the line with distance d from the origin $\mathcal{O}$. $\mathbf{a}$ and $\mathbf{b}$ are two points on the line, from which we can define $\mathbf{n} = \mathbf{a} \times \mathbf{b}$ and $\mathbf{v} = \mathbf{b} - \mathbf{a}$.

with

$$[\mathbf{n}]_\times = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}, \tag{3.23}$$

the line can be represented as a 6-dimensional vector. Setting $\mathbf{n} = \mathbf{a} \times \mathbf{b}$ and $\mathbf{v} = \mathbf{b} - \mathbf{a}$, with $\mathbf{a}$ and $\mathbf{b}$ two homogeneous points, the Plücker line is

$$\mathcal{L} \sim \begin{pmatrix} \mathbf{n} \\ \mathbf{v} \end{pmatrix} = (n_x, n_y, n_z, v_x, v_y, v_z)^T \tag{3.24}$$

The Plücker line is a very useful representation because its geometric definition can be immediately derived. $\mathbf{n}$ is a normal vector to the plane containing the line and the origin, $\mathbf{v}$ is a direction vector of line, oriented from $\mathbf{a}$ to $\mathbf{b}$. Therefore, the property $\mathbf{n} \perp \mathbf{v}$.

Transformations in 3D

In the context of SLAM, we continuously deal with transformations between the poses of the robot. A pose represents both the position in the space and the orientation of the robot. Since the goal of this thesis is 3D localization and mapping, we consider poses in 3D space. The pose is defined with respect to the origin of a specific coordinates frame, with the position being the translation from the origin and the orientation being the rotation with respect to the three axes. The robot pose is usually defined with respect to the world origin, which coincides with the first point of the trajectory. To represent the pose in a compact way, we use a 4x4 matrix $\mathbf{T}$ containing both the rotation and the translation components, which looks like this:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{o}^\mathsf{T} & 1 \end{bmatrix} \tag{3.25}$$

where $\mathbf{R}$ is a 3x3 matrix, $\mathbf{t}$ is a 3-dimensional column vector, and $\mathbf{o}^\mathsf{T}$ is a 3-dimensional row vector.

Matrix $\mathbf{T}$ can be applied to change coordinates frame. Suppose that we have a landmark $\mathbf{x}$ with position given by the 3D homogeneous point $\mathbf{x}_{\mathtt{robot}}$ in robot frame, observed by the robot pose $\mathbf{T}$ in world frame. In order to transform $\mathbf{x}_{\mathtt{robot}}$ to world frame ($\mathbf{x}_{\mathtt{world}}$), we just need to multiply the point $\mathbf{x}$ by the pose of the robot. This means applying to the point the rotation and translation of the robot. If the same point $\mathbf{x}$ is described by the Cartesian coordinates $\bar{\mathbf{x}}_{\mathtt{robot}}$, with $\mathbf{x}_{\mathtt{robot}} = (\bar{\mathbf{x}}_{\mathtt{robot}}^\mathsf{T}, 1)^\mathsf{T}$, we can describe the equivalent transformation in the Cartesian world as first rotating $\bar{\mathbf{x}}_{\mathtt{robot}}$ by $\mathbf{R}$ and then translate it by $\mathbf{t}$. In Cartesian coordinates it would be:

$$\bar{\mathbf{x}}_{\mathtt{world}} = \mathbf{R}\bar{\mathbf{x}}_{\mathtt{robot}} + \mathbf{t}, \tag{3.26}$$

and in homogeneous coordinates it is:

$$\mathbf{x}_{\mathtt{world}} = \mathbf{T}\mathbf{x}_{\mathtt{robot}} \tag{3.27}$$

where $\mathbf{x}_{\mathtt{world}} = \left(\bar{\mathbf{x}}_{\mathtt{world}}^\mathsf{T}\ 1\right)^\mathsf{T}$ and $\mathbf{x}_{\mathtt{robot}} = \left(\bar{\mathbf{x}}_{\mathtt{robot}}^\mathsf{T}\ 1\right)^\mathsf{T}$.

Another very useful application is the computation of the difference between two robot poses. If we want to compute rotation and translation from pose with transformation matrix $\mathbf{T}_1$ to pose with transformation matrix $\mathbf{T}_2$, it is sufficient to compute $\mathbf{T}_1^{-1}\mathbf{T}_2$ to obtain the odometry measurement between the two poses.

Intersection of a Line with a Plane

Given the Plücker matrix of a line $\mathbf{L}$ and the dual homogeneous coordinates of a plane $\mathbf{p}$, in order to find the incidence point $\mathbf{x}_0$ (Figure 3.6) of the line on the plane, we can write: $\mathbf{L} \sim \mathbf{x}_0\mathbf{x}_1^\mathsf{T} - \mathbf{x}_1\mathbf{x}_0^\mathsf{T}$ by equation (3.16) and assuming $\mathbf{x}_1$ to be another point on the line $\mathbf{L}$. Since $\mathbf{x}_0$ also lies on the plane, we have that $\mathbf{x}_0 \cdot \mathbf{p} = 0$ by duality property of planes and points in 3D projective geometry. Therefore we can write:

$$\mathbf{L}\mathbf{p} \sim \left(\mathbf{x}_0\mathbf{x}_1^\mathsf{T} - \mathbf{x}_1\mathbf{x}_0^\mathsf{T}\right)\mathbf{p} = \mathbf{x}_0\left(\mathbf{x}_1 \cdot \mathbf{p}\right) - \mathbf{x}_1\underbrace{\left(\mathbf{x}_0 \cdot \mathbf{p}\right)}_{=0} = \mathbf{x}_0\left(\mathbf{x}_1 \cdot \mathbf{p}\right) \sim \mathbf{x}_0. \tag{3.28}$$

In the case that also $\mathbf{x}_1$ lies on the plane, we have that the line lies on the plane too, leading to the result:

$$\text{The line } \mathbf{L} \text{ lies on the plane } \mathbf{p} \Leftrightarrow \mathbf{L}\mathbf{p} = 0. \tag{3.29}$$

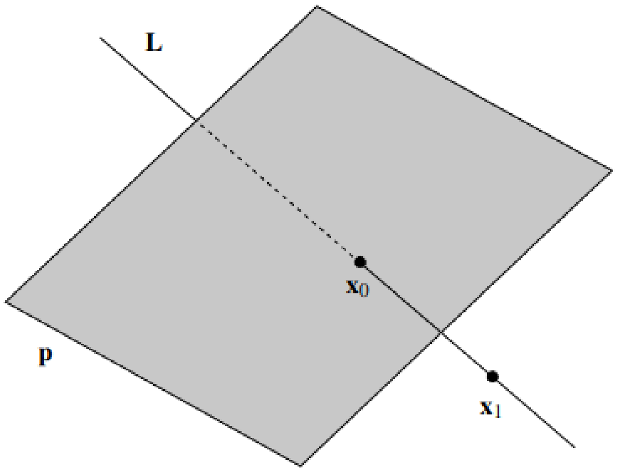


Figure 3.6: $\mathbf{x}_0$ is the incidence point of the line $\mathbf{L}$ on plane $\mathbf{p}$. (Image from [4])

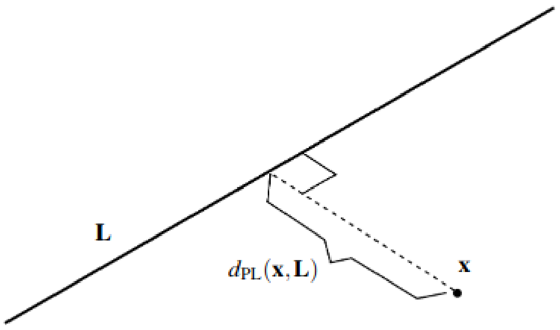


Figure 3.7: $\mathbf{d_{PL}}(\mathbf{x},\mathbf{L})$ is the distance between the line $\mathbf{L}$ and the point $\mathbf{x}$. (Image from [4])

Distance between a Point and a Line

Given a proper line $\mathbf{L}$ and a proper point $\mathbf{x}$ in $\mathbb{E}^3$, we want to compute the distance from the point to the line (Figure 3.7). $\widetilde{\mathbf{L}} \sim \mathbf{p}_1\mathbf{p}_2^\top - \mathbf{p}_2\mathbf{p}_1^\top$ is the dual Plücker matrix of $\mathbf{L}$ by equation (3.18). Since we can select $\mathbf{p}_1$ as we like, we choose it to be the plane where point $\mathbf{x}$ also lies. So we get:

$$\widetilde{\mathbf{L}}\mathbf{x} = \left(\mathbf{p}_1\mathbf{p}_2^\top - \mathbf{p}_2\mathbf{p}_1^\top\right)\mathbf{x} = \mathbf{p}_1\left(\mathbf{p}_2\cdot\mathbf{x}\right) - \mathbf{p}_2\underbrace{\left(\mathbf{p}_1\cdot\mathbf{x}\right)}_{=0} = \mathbf{p}_1\left(\mathbf{p}_2\cdot\mathbf{x}\right) \tag{3.30}$$

Then we apply DL-Normalization on $\widetilde{\mathbf{L}}$ and P-normalize $\mathbf{x}$ and we have:

$$\mathrm{norm}_{\mathrm{DL}}\left(\widetilde{\mathbf{L}}\right)\mathrm{norm}_{\mathrm{P}}\left(\mathbf{x}\right) = \mathrm{norm}_{\mathrm{D}}\left(\mathbf{p}_1\right)\underbrace{\left(\mathrm{norm}_{\mathrm{D}}\left(\mathbf{p}_2\right)\cdot\mathrm{norm}_{\mathrm{P}}\left(\mathbf{x}\right)\right)}_{=\,d} = d\cdot\mathrm{norm}_{\mathrm{D}}\left(\mathbf{p}_1\right) \tag{3.31}$$

where d is the signed distance between point $\mathbf{x}$ and plane $\mathbf{p}_2$ which is perpendicular to plane $\mathbf{p}_1$. Since $\mathbf{p}_1$ is D-normalized, the first three elements have unit norm and the point-to-line distance is

$$|d| = \left\|\left(\mathrm{norm}_{\mathrm{DL}}\left(\widetilde{\mathbf{L}}\right)\mathrm{norm}_{\mathrm{P}}\left(\mathbf{x}\right)\right)_{1:3}\right\| \tag{3.32}$$

Parallel Lines

In order to force two 3D lines to be parallel we can exploit their Plücker line representation. By (3.24) we have $\mathcal{L}_1 = \left(n_{x1}, n_{y1}, n_{z1}, v_{x1}, v_{y1}, v_{z1}\right)^T$ and $\mathcal{L}_2 = \left(n_{x2}, n_{y2}, n_{z2}, v_{x2}, v_{y2}, v_{z2}\right)^T$. Since vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ represent respectively the direction vectors of lines $\mathcal{L}_1$ and $\mathcal{L}_2$, we can use the the property of parallel vectors to have cross product equal to zero and impose the cross product of the two direction vectors to be zero. Therefore, we can write the property:

$$\mathcal{L}_1 \text{ and } \mathcal{L}_2 \text{ are parallel lines } \Leftrightarrow \begin{cases} v_{y1}v_{z2} - v_{z1}v_{y2} = 0 \\ v_{z1}v_{x2} - v_{x1}v_{z2} = 0 \\ v_{x1}v_{y2} - v_{y1}v_{x2} = 0 \end{cases} \tag{3.33}$$

.

3.2 G2O

g2o [15], which stands for General Graph Optimization, is a general framework for performing optimization of nonlinear least squares problems in the form of a graph. It can be used for graph SLAM optimization back-end and for Bundle Adjustment problems. It is particularly efficient for two reasons: it leverages the sparse structure of the graph, and it uses advanced methods to solve sparse linear systems. It is also general and extensible, as it stands out for the easiness of building a graph or implementing custom nodes and edges within the framework. Moreover, the number of already implemented nodes and edges is substantial. In the case they suit the application, it is sufficient to compute the initial estimate for nodes, and the measurement error for edges. As it turns out, even defining new nodes and edges is very easy, but we explain it after presenting the framework logic and implementation for graph optimization.

Graph optimization is a nonlinear least squares problem, which is usually solved by linearizing the system around the current state, solving, and finally iterating. Several problems in robotics and computer vision aim at finding the minimum of an objective function which can be formulated as this:

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j\rangle \in C} \underbrace{\mathbf{e}_{ij}^{T}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}\right) \Omega_{ij} \mathbf{e}_{ij}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}\right)}_{\mathbf{F}_{ij}}, \tag{3.34}$$

with the following solution:

$$\mathbf{x}^{*} = \arg\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}). \tag{3.35}$$

It can be noted that (3.34) is the same as (3.4) and that (3.35) equals (3.5). This is because the g2o framework exactly addresses the solution of the graph SLAM formulation. Here, $\mathbf{x} = \left(\mathbf{x}_1^T, ..., \mathbf{x}_n^T\right)$ is a vector of parameters representing the state of the system, $\mathbf{z}_{ij}$ and Ω_{ij} are the mean and the information matrix of the constraint between $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{e}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}\right)$ is the error vector of the constraint.

From now on, $\mathbf{e}\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}\right)$ is just called $\mathbf{e}_{ij}(\mathbf{x})$. As it was described for graph SLAM in general, the error function can be approximated by its first order Taylor expansion around the current solution $\check{\mathbf{x}}$:

$$\mathbf{e}_{ij}\left(\check{\mathbf{x}}_i + \Delta\mathbf{x}_i, \check{\mathbf{x}}_j + \Delta\mathbf{x}_j\right) \simeq \mathbf{e}_{ij}(\check{\mathbf{x}}) + \mathbf{J}_{ij}\Delta\mathbf{x} \tag{3.36}$$

with $\mathbf{J}_{ij}$ Jacobian of $\mathbf{e_{ij}}(\hat{\mathbf{x}})$. Substituting (3.36) into (3.34), we have for each error term $\mathbf{F}_{ij}$:

$$\mathbf{F}_{ij} \simeq \underbrace{\mathbf{e}_{ij}^{T}\Omega_{ij}\mathbf{e}_{ij}}_{c_{ij}} + 2\underbrace{\mathbf{e}_{ij}^{T}\Omega_{ij}\mathbf{J}_{ij}}_{\mathbf{b}_{ij}}\Delta\mathbf{x} + \Delta\mathbf{x}^{T}\underbrace{\mathbf{J}_{ij}^{T}\Omega_{ij}\mathbf{J}_{ij}}_{\mathbf{H}_{ij}}\Delta\mathbf{x} = c_{ij} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + \Delta\mathbf{x}^{T}\mathbf{H}_{ij}\Delta\mathbf{x} \tag{3.37}$$

and the quadratic objective function equal to

$$\mathbf{F}(\check{\mathbf{x}} + \Delta\mathbf{x}) = \sum_{\langle i,j\rangle \in C} \mathbf{F}_{ij}(\check{\mathbf{x}} + \Delta\mathbf{x}) \simeq c + 2\mathbf{b}^{T}\Delta\mathbf{x} + \Delta\mathbf{x}^{T}\mathbf{H}\Delta\mathbf{x}. \tag{3.38}$$

By computing

$$\mathbf{H}\Delta\mathbf{x}^{*} = -\mathbf{b} \tag{3.39}$$

we obtain the closed form solution:

$$\mathbf{x}^{*} = \check{\mathbf{x}} + \Delta\mathbf{x}^{*}. \tag{3.40}$$

The Gauss-Newton algorithm iterates these three steps: linearization (3.37), solution of quadratic system (3.39), and update step (3.40). At each iteration, the previous solution $\mathbf{x}^{*}$ is used as initial guess and linearization point. The procedure iterates until a termination criterion is satisfied. The

Levenberg-Marquardat algorithm [24] introduces a damping factor to control convergence and the solution of the quadratic system assumes the form of:

$$(\mathbf{H} + \lambda\mathbf{I})\,\Delta\mathbf{x}^* = -\mathbf{b}, \tag{3.41}$$

where λ is a damping factor that reduces the step size, and it changes over time.

Now, the structure of the linearized system is analyzed. Since the error function of each constraint only depends on at most two variables (two nodes for binary constraints, otherwise just one node for unary edges), the Jacobian matrix is zero everywhere, except for two blocks, which we call $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$. These two blocks are the derivatives of the error function with respect to the increment of the two variables $(\Delta\mathbf{x}_i, \Delta\mathbf{x}_j)$. Therefore, the matrix $\mathbf{H}_{ij} = \mathbf{J}_{ij}^T\Omega_{ij}\mathbf{J}_{ij}$ is zero everywhere except for four blocks, given by the combination of $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ with the information matrix Ω_{ij}. Similar considerations can be made for the vector $\mathbf{b}_{ij} = \mathbf{J}_{ij}^T\Omega_{ij}\mathbf{e}_{ij}$. The result is:

$$\mathbf{H}_{ij} = \begin{bmatrix} 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & & & & & & \vdots \\ & & \mathbf{A}_{ij}^T\Omega_{ij}\mathbf{A}_{ij} & \cdots & & \mathbf{A}_{ij}^T\Omega_{ij}\mathbf{B}_{ij} & & \vdots \\ \vdots & & \vdots & & & \vdots & & \vdots \\ \vdots & & \mathbf{B}_{ij}^T\Omega_{ij}\mathbf{A}_{ij} & \cdots & & \mathbf{B}_{ij}^T\Omega_{ij}\mathbf{B}_{ij} & & \vdots \\ \vdots & & & & & & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{3.42}$$

$$\mathbf{b}_{ij} = \begin{bmatrix} 0 \\ \vdots \\ \mathbf{A}_{ij}^T\Omega_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T\Omega_{ij}\mathbf{e}_{ij} \\ \vdots \\ 0 \end{bmatrix} \tag{3.43}$$

where all non-specified elements are zero.

It is interesting to note that for BA problems we can exploit an even more particular structure. Since there are two types of poses, those of the camera $\mathbf{p}$ and the landmarks $\mathbf{l}$, the matrix $\mathbf{H}$ can be re-arranged, together with the solution of the quadratic system, as:

$$\begin{bmatrix} \mathbf{H}_{pp} & \mathbf{H}_{pl} \\ \mathbf{H}_{pl}^T & \mathbf{H}_{ll} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{x}_p^* \\ \Delta\mathbf{x}_l^* \end{pmatrix} = \begin{pmatrix} -\mathbf{b}_p \\ -\mathbf{b}_l \end{pmatrix}, \tag{3.44}$$

which is exactly the same as (3.39). This is equivalent to taking the Schur complement of $\mathbf{H}$ to solve for the camera increment $\Delta\mathbf{x}_p^*$:

$$\left(\mathbf{H}_{pp} - \mathbf{H}_{pl}\mathbf{H}_{ll}^{-1}\mathbf{H}_{pl}^T\right)\Delta\mathbf{x}_p^* = -\mathbf{b}_p + \mathbf{H}_{pl}\mathbf{H}_{ll}^{-1}\mathbf{b}_l, \tag{3.45}$$

and landmarks increment is computed by:

$$\mathbf{H}_{ll}\Delta\mathbf{x}_l^* = -\mathbf{b}_l - \mathbf{H}_{pl}^T\Delta\mathbf{x}_p^*. \tag{3.46}$$

The Schur complement is a tool used for matrix decomposition in numerical analysis, statistics and matrix analysis. Having a matrix $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ the Schur complement of block D of the matrix M is $M/D = A - BD^{-1}C$ if D is invertible, and the Schur complement of block A of the matrix M is $M/A = D - BA^{-1}C$ if A is invertible.

This alternative method constitutes an important speed-up if the number of landmarks is much higher than the number of poses and the assumption of landmarks only connected to camera poses is satisfied. Otherwise, it is more convenient to use the first method.

In order to better fit the problem at hand, g2o provides three different linear solvers: two based on Cholesky decomposition (CHOLMOD and CSparse) and an iterative method based on block-Jacobi preconditioner (PCG).

3.2.1 Robust Least Squares

Least squares optimization can be optionally robustified, in order to lower the influence of outliers. The error function of each constraint has quadratic influence on the objective function, thus outliers may have a very large weight on the optimization. The solution is to substitute the quadratic error function with a more robust error function which decreases the weight of large errors. An example of robust cost function is the Huber kernel [25], which is quadratic for small errors and linear for potential outliers, with a customizable threshold. If we describe the standard quadratic cost function single term as:

$$\mathbf{F}_k = \mathbf{e}_k^T\Omega_k\mathbf{e}_k = \rho_2\left(\sqrt{\mathbf{e}_k^T\Omega_k\mathbf{e}_k}\right) \text{ with } \rho_2(\mathbf{x}) := \mathbf{x}^2, \tag{3.47}$$

then, the Huber kernel has the following form:

$$\rho_H(\mathbf{x}) := \begin{cases} \mathbf{x}^2 & \text{if} |\mathbf{x}| < \mathbf{b} \\ 2\mathbf{b}|\mathbf{x}| - \mathbf{b}^2 & \text{otherwise,} \end{cases} \tag{3.48}$$

with the advantage to remain convex anyway. During the implementation, each constraint can be optionally robustified by enabling robust least squares and selecting the kernel for the corresponding edge.

3.2.2 Implementing Custom Nodes and Edges

g2o has a large set of already implemented vertices and edges, but if the problem to solve has specific demands, custom nodes and edges can be implemented. Nodes are called vertices, and in order to implement a new vertex type, one should extend the basic template of **BaseVertex** and specify the size and type of the estimate. For example, the 3D SLAM **VertexSE**3 declaration is

class G2O_TYPES_SLAM3D_API VertexSE3 : public BaseVertex <6,
Isometry3>,

which means that it inherits the basic properties and functions of *BaseVertex* and its estimate has 6 dimensions corresponding to the 3D translation vector and the rotation in the form of the first three elements of a normalized quaternion. The dimension parameter actually refers to the dimension of the vertex in the manifold, but it could be parametrized in a more convenient way while dealing with front-end. Then, the custom edge class must overwrite the following basic functions:

- virtual bool *read*(std::istream& is);
- virtual bool *write*(std::ostream& os) const;
- virtual void *oplusImpl*(const number_t* update);
- virtual void *setToOriginImpl*();

read/write functions are needed in order to store and read the vertex id and estimate to an output file, in case that intermediate results want to be saved. The *oplusImpl* function is to apply the increment to the current estimate at each optimization iteration. The *setToOriginImpl* resets the estimate to its initialization value.

As for what concerns new edges, they must extend one of the following templates: **BaseUnaryEdge**, **BaseBinaryEdge**, **BaseMultiEdge**, depending on how many vertices they connect. For example, the basic type **EdgeSE**3 which connects two *VertexSE*3 is declared as:

class G2O_TYPES_SLAM3D_API EdgeSE3 : public BaseBinaryEdge <6,
Isometry3, VertexSE3, VertexSE3>

and it means that the measurement variable is stored as an isometry value, that is a 6-dimensional transformation, to be decomposed in a 3D translation vector and a 3x3 rotation matrix. The functions to be implemented are:

- virtual bool *read*(std::istream& is);
- virtual bool *write*(std::ostream& os) const;

- virtual void *computeError*();
- virtual void *linearizeOplus*();

read/write store and retrieve edge data from a file. Edge data include the parameters (for example, camera calibration parameters), the identification numbers of the connected vertices, the measurement, and the triangular upper part of the information matrix. *computeError* is needed in order to define the computation of the error with respect to the measurement and the updated vertices at each iteration of the optimization. *linearizeOplus* defines the Jacobian matrices for the linearization of the error with respect to the connected vertices, and it is not a mandatory function. If it is not implemented, the Jacobians are numerically computed, however it is strongly advised to analytically compute them in order to speed up the optimization.

Once a new edge or vertex has been created, the name must be registered to g2o with the G2O_REGISTER_TYPE macro. This additional step provides an identification tag for the type of element when writing or reading the graph elements from the g2o file. Finally, also custom parameters can be defined, in order to provide additional information to the edges to whom they are attached. Parameters must be registered to g2o as well.

3.3 IMAGE SEGMENTATION WITH NEURAL NETWORKS

As we faced the problem of identifying natural landmarks such as crops from pictures, it is essential to introduce the image segmentation problem. Image Segmentation is a deep learning problem that can be solved with neural networks. It essentially consists in classifying each single pixel of the input image with a label, so the output is the image itself with a label per pixel. Hereafter we consider the words 'label' and 'class' as synonyms. Most of the methods actually assign a probability to the pixel to belong to each class, then a decision is made based on frequency of each class and other influencing factors depending on the application.

Many neural network models exist already implemented and can be used as they are or adjusted to the specific context. For example, a popular model for image segmentation is the U-Net [82] architecture. It was first developed for biomedical applications, but can be easily modified to apply to different fields. It is divided into two main components: an encoder and a decoder. The encoder is also said the contracting path and the decoder the expansive path. The contracting path is composed by basic blocks of two convolution layers, an activation function and a max pooling layer. Convolution layers consist in the application of a 3x3 filter (in this case, but it can also have different size) on the input volume performing some operation that applies a local perturbation depending on the value of surrounding pixels. We use the word volume to refer to any intermediate result between any couple

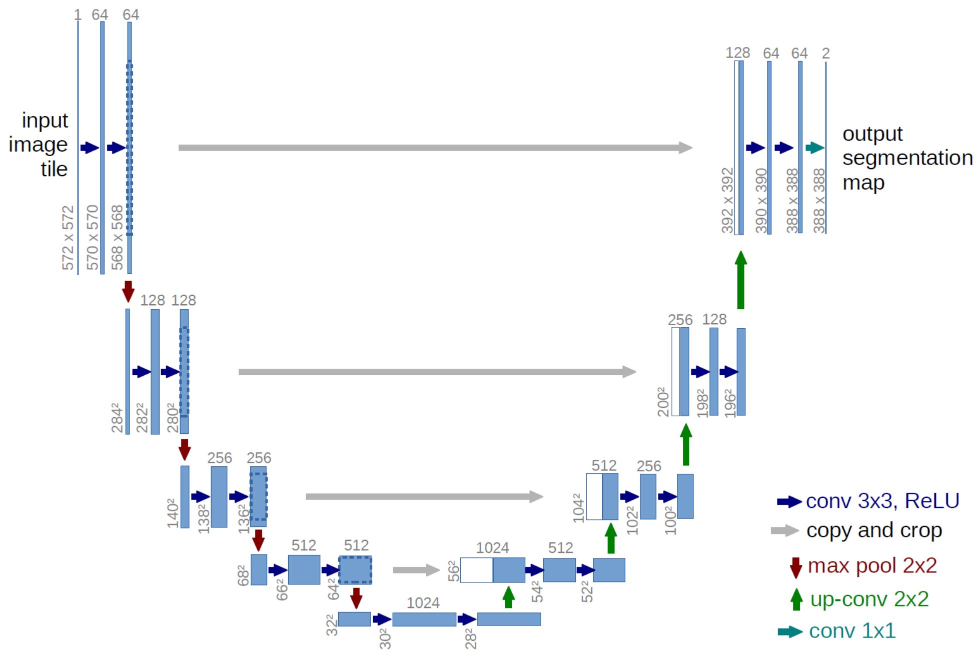


Figure 3.8: U-Net architecture, from [82]. It is composed by an encoder and a decoder part, and performs concatenation of layers on contracting path with layers on expanding path for higher resolution maps.

of layers. It is actually defined by three dimensions which depend on the parameters of the layer that produced it. Going on with the analysis, we have activation functions, being mathematical equations that determine if the neuron is activated. U-Net uses ReLu which is considered one of the most efficient activation functions when the vanishing problem would arise, otherwise making training ineffective. Max pooling is actually the one causing the contraction since 2x2 max pooling with stride 2 layers halve the height and width of the input volumes. The increasing depth, instead, is due to the increasing number of convolution filters at each layer and is also referenced as the number of features or channels in the network.

The expansive path is composed by basic blocks of an upsamplig layer, a concatenation with the feature maps from the decoder part, a couple of convolutional layers and an activation function. Convolutional layers and activation function are the same as described above. However, the upsampling layer is an 'up-convolution', that is a transposed convolution that halves the number of feature channels and increments the size of the feature volume. Concatenation of the output volume with layers from contracting path produces higher resolution feature maps. Layer by layer we get back to the original size and finally apply a 1x1 convolution producing the classification result for each pixel. The final layer and especially the choice of the activation function are strongly correlated to the application requirements, such as the number of output classes.

Being a neural network, U-Net needs to be trained to learn the specific problem it faces. Once it has been trained, the weights, which represent the parameters to be learnt by the network, can be stored and re-used. When a trained model is loaded, it can be directly applied to the problem by feeding the images to the network.

3.4 DBSCAN

DBSCAN [76] is a clustering algorithm based on the identification of high density areas, separated by low density areas. It is particularly effective because, unlike other clustering algorithms, it does not assume a convex shape for clusters. This is essential for our application, where clusters to be identified correspond to crops which have leaves spreading from the center of mass and are not convex at all.

The algorithm works by computing *core* samples which are the ones lying in the areas with higher density. The *core* samples are connected to other *core* samples and to *edge* samples which lie at the border of clusters. The difference between *core* and *edge* samples is the number of connected samples. The parameter ϵ customizing the algorithm is a measure of the desired density minimum limit for two samples to be in the same cluster, thus the maximum distance allowed. All samples not connected to other samples are noise points. The possibility to identify noise points is the second key feature that made us choose this algorithm, since it is applied on the result of segmentation and can identify some misclassified points. This idea is clarified during the software architecture explanation.

As we said, in order for a sample to be a *core* sample, it must have at least *min_samples* neighbours, which are the connected samples within ϵ distance. *min_samples* defines tolerance with respect to noise. The algorithm is implemented in the scikit-learn [77] module for python, being an essential tool to work with machine learning models.

4 DATASET AND TOOLS

This chapter deals about the description of the robot and the sensors used to collect the dataset we used in out work. The dataset [21] was collected by the robot Bonirob, by Bosch, in a sugar beet field near Bonn in Germany over a period of three months in spring 2016. It was recorded in the form of ROS bags, with all sensors readings registered as ROS messages. Its purpose is to collect a large quantity of data in order to enhance the study of agricultural SLAM and validate it over available data. It also provides an image segmentation ground truth classifying different plants and identifying the centroids, or stem emerging points, of the crops observed in the field. After the dataset description, we include the explanation of the Software tools we used in the thesis.

4.1 BONIROB

Bonirob is an agricultural robot which can be set up in order to complete several tasks. Its chassis is 1.8 m x 1.3 m x 0.8 m, and it is mounted over four wheels. The clear height of the chassis is 85 cm. The robot coordinate frame is located at the base of the chassis, as it is shown in Figure 4.1.

4.1.1 Sensors

The robot can be equipped with different sensors. In particular, in the considered setup the sensors used are depicted in Figure 4.2. In the following paragraphs, we describe the sensors which were used to collect the data we used in our work:

JAI AD-130GE camera: A prism-based camera with four channels image: RGB and near-infrared (NIR) channels, which can be appended together as the optical path is the same. Image resolution is 1296 pixel x 966 pixel. The camera is mounted at the bottom of the robot chassis, resulting to be at the height of 85 cm, looking directly downwards. The field of view is thus 24 cm x 31 cm. The main purpose of images collection is to exploit visual information for the implementation of crop perception systems and for phenotyping information extraction. In this study however, the images are used to identify the 3D points to be used as landmarks in the SLAM algorithm.

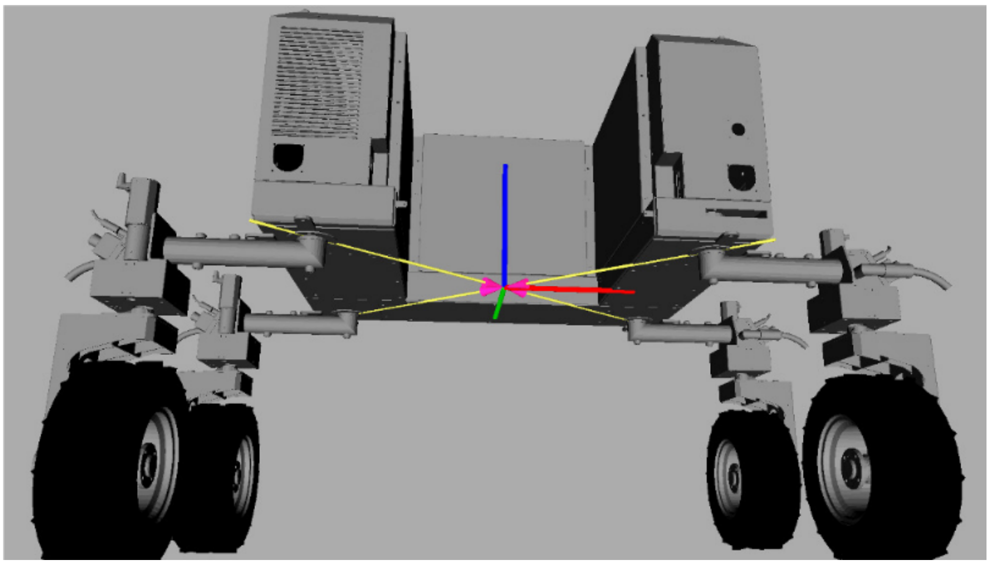


Figure 4.1: Bonirob coordinate frame, called base link : x -axis in red, the y -axis in green, and the z -axis in blue. (Image from [21])

Leica RTK GPS: A Real Time Kinematic GPS by Leica. It provides really precise estimation of the robot position, also thanks to the base station with known location it needs close to the operating area of the robot. The final estimate, corrected by the base station, has an accuracy of few centimeters. The signal is recorded at a frequency of 10 Hz, with respect to World Geodetic System 1984 (WGS84). WGS84 is a Cartesian coordinate system where the origin is the center of the Earth, z-axis goes through North Pole, x-axis is choosen in order to have the Greenwich meridian on the xz plane, and the y-axis is such that the right-hand rule is respected. This sensor has the disadvantages to be extremely expensive and requiring a base station to be installed in the field.

Ublox GPS: A consumer-grade Ublox EVK7-P GPS to estimate the robot position. The estimation principle is Precise Point Positioning. It is recorded at 4 Hz with respect to World Geodetic System 1984. Precise Point Positioning (PPP) relies on a network of reference stations to compute precise estimates of Global Navigation Satellite System satellites orbits and clock errors. However, it requires a smaller number of reference stations globally distributed as compared with differential approaches (for example, Real Time Kinematics, RTK), and one set of precise orbit and clock data (computed by a processing center) is valid everywhere. Thus, it is rather reliable, but much cheaper than RTK sensors.

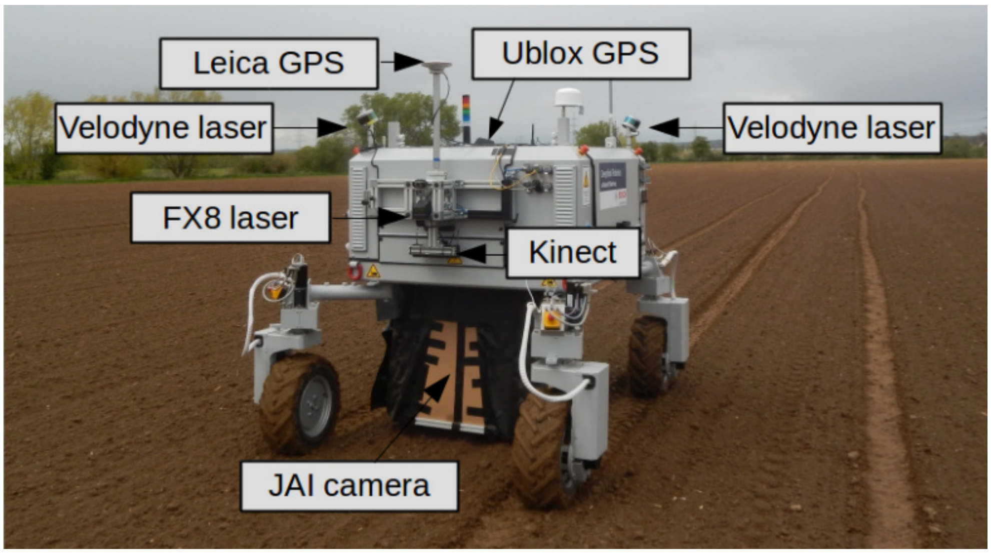


Figure 4.2: Bonirob sensors, from [21].

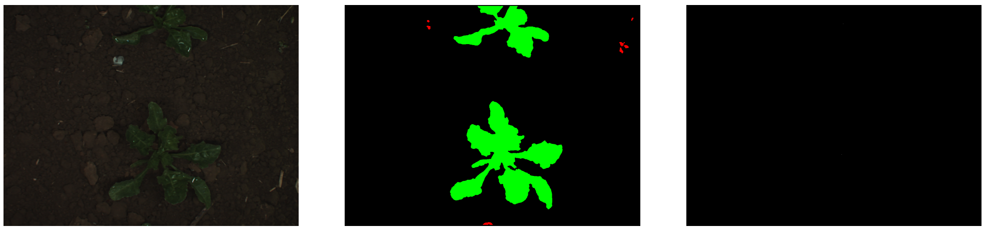
4.1.2 Sensors Calibration

In order to make use of the sensors measurements, we must know the relative transformations between the sensor position and the robot center: the base_link. Indeed, while dealing with SLAM, the robot pose estimation always refers to the center of the robot coordinate frame, the base_link, in fact. Therefore, if we have the GPS measurement for example, from the position of the GPS antenna, we must obtain the robot position. To extract this information, we need to know the sensors calibration. Eventually, after transforming all sensors measurements to the robot coordinate frame, we will be able to fuse all of them in the pose estimation.

Sensors calibration parameters are of two types: extrinsic and intrinsic parameters. Intrinsic parameters, in this context, are only characteristic of the camera sensor. They refer to the projection between the image plane pixel coordinates and the corresponding 3D point. Extrinsic parameters, instead, describe the relative transformation between the position of the sensor and the base_link. In particular, the transformation is from the robot coordinate frame to the frame of each sensor, from which we can compute also the inverse transformation.

4.2 DATASET DESCRIPTION

The dataset is divided into sequential ROS bags recorded during different days and at different growing stages. Not all bags contain all sensor measurements. In particular, in the first group of bags GPS is not included.



(a) RGB image. **(b)** Segmentation ground truth. **(c)** Centroids ground truth.

Figure 4.3: Image segmentation ground truth provided in the Bonirob dataset. (Image from [21])

In addition to ROS bags, the dataset contains manually labeled images which can be used for image segmentation. They provide a ground truth for segmenting the image in crops and weed against the background. Manual labels provide the classification into sugar beets, which is the crops class, and nine different species of weeds. Also, the same images were post-processed to extract the center of mass of each identified plant. Results are stored in a black and white image, where only the pixel corresponding to the SEP position is coloured. As manual labeling is an extremely expensive task, it was done only for some chunks of data, corresponding to few ROS bags. Figure 4.3 shows an example of the ground truth provided for an image in the dataset. On the left (Figure 4.3a) we have the RGB image, in the center (Figure 4.3b) we have the segmented image ground truth and on the right (Figure 4.3c) we have the ground truth with identified centroids for the two crops in the image. As a first implementation of the application, we directly employed the centroids ground truth images, to have a measure of the effectiveness of the algorithm when integrating natural landmarks in the optimization.

In the bag files, beyond the sensors measurements, odometry measurement is recorded. It includes the estimation, according to the motion model, of the position, orientation and linear and angular velocities of the robot. All measurements messages are associated to a timestamp. In this way, we can correctly order the measurements in the same bags. Bags are globally ordered by the date and time they were recorded.

Since we are interested in fusing the GPS measurements and the plants position into the SLAM estimation, we started from the bags that include both GPS and JAI sensors measurements, together with SEP ground truth. However, as soon as we implemented the centroids extraction component, we were able to apply the developed algorithm to bags that did not provide the segmentation ground truth.

4.2.1 Images for Centroids Identification

Even if the main purpose of the application is to develop a SLAM algorithm, we also implemented a component to identify SEPs from input images in the dataset. This goal was achieved by applying a neural network for image segmentation on the images and later process the results with a clustering algorithm and a the computation of the center of mass for each identified cluster. To train the neural network we have used the Kaggle [75] platform, which provides the possibility to use a gpu to train neural networks. As training dataset we have used the segmented images in the Bonirob dataset, while we estimated the error on centroids computation by comparing the obtained results with the labeled binary images with SEP positions.

4.3 ROS

ROS (Robot Operating System) [7] is a middleware for robotics. Robotics applications, indeed, often rely on a middleware layer to manage the complexity of connecting software and hardware. On the one hand, they have to interact both with high level libraries and other software packages. On the other hand, they collect sensors data and connect with low level hardware. The use of a middleware also enhances the portability and the reliability of applications. ROS is distributed and potentially scaling, it reuses code and works both with Python and C++. The main components of ROS architecture will be be described hereafter.

The executable units of ROS are the nodes. They communicate with each other and perform computations on resources they exchange. For each network of nodes, a master must be created (**roscore**) which is in charge of naming and registering other nodes. It manages the communication and allows nodes to locate each others.

Communication occurs through a paradigm called publish/subscribe, which is characteristic in middlewares. It implies a group of nodes publishing some data, while a group of subscribers receives those data. In this way, sending and receiving the message is completely decoupled, and both senders and subscribers do not need to know the identity of other entities. The only information needed is the name of the channel on which data are published. In ROS, messages are published on a channel which is called 'topic'. Each topic has a name and a message type. Multiple nodes can publish on a topic and each topic can be read by multiple nodes.

Messages are the information exchanged on topics, which in turn must be formatted according to the message type. Standard messages such as strings, integers and headers are already implemented, and there exist the possibility to define custom messages by composing standard types.

Services can be developed that implement client/server paradigm, where the client connects to the server in order to make use of a specific service, such as the sharing of a resource or a given computation. They have a service type, as with topics, however the call is synchronous, so the execution waits for the response to be computed and sent back. Thus, service requests are guaranteed to be attended, whereas messages are not guaranteed to be received. A parameter server is a tool that stores parameters for node execution. Nodes can access those parameters at runtime from a dictionary.

ROS bags are the containers to store data, such as ROS messages, services and parameters. After being stored, they can be accessed by different nodes at any time. The bag can read inside the code, or played by the command line. In the second case, a 'publisher' node is created to publish all messages in the bag. Recording a sequence of messages in a bag can be useful for multiple reasons. An example is to test algorithms with data, simulating messages to be exchanged at the time of execution of the algorithm.

Software code is organized in ROS packages. They contain the source code for nodes and services and eventually custom implemented messages. As we said, the execution of ROS requires a master node. It can be manually switched on by the command **roscore**, while manually executing all other nodes, or a launch file managing the execution of all nodes together can be created. A launch file includes all nodes to be created and registered, and when executed it automatically creates the master node. It also allows nodes to use parameters and specify options for the execution.

4.3.1 ROS Messages

In the following, we present the messages we used in this thesis.

The odometry measurement is represented by a **nav_msgs/Odometry.msg** [10] message. It is composed by the following elements, some of which are already existing messages per se, which are merged while keeping their internal structure:

- std_msgs/Header *header*: a set of meta information about the time at which the message was published and the frame (*frame_id*) in which the *pose* message is computed.

- string *child_frame_id*: the name of the coordinates frame in which the *twist* must be published.

- geometry_msgs/PoseWithCovariance *pose*: the set containing position, orientation and covariance of the odometry measurement. *Position* is in 3D vector form, *orientation* is in quaternion form, and *covariance* is a 36-dimensional vector of floating-point precision.

- geometry_msgs/TwistWithCovariance *twist*: it includes the velocity and the covariance of the robot. In particular, the velocity has two components: *angular* and *linear* velocity, in the form of two 3-dimensional vectors.

The GPS measurement in the ROS bag file is stored as a **sensor_msgs/NavSatFix.msg** [9] message. It is composed by the following elements:

$$\left.\begin{array}{l}\bullet\ \text{uint8 } \textit{COVARIANCE\_TYPE\_UNKNOWN=0}\\ \bullet\ \text{uint8 } \textit{COVARIANCE\_TYPE\_APPROXIMATED=1}\\ \bullet\ \text{uint8 } \textit{COVARIANCE\_TYPE\_DIAGONAL\_KNOWN=2}\\ \bullet\ \text{uint8 } \textit{COVARIANCE\_TYPE\_KNOWN=3}\end{array}\right\}\begin{array}{l}\text{Information}\\ \text{about the type}\\ \text{of covariance}\\ \text{measurement}\\ \text{provided with the}\\ \text{GPS estimation.}\end{array}$$

- std_msgs/Header *header*: the same as for the *odometry* message.

- sensor_msgs/NavSatStatus *status*: the information about the fix status for the Global Navigation Satellite System, defining the type of signal being used by the receiver to calculate its location. It indicates the quality or reliability of the resulting location.

$$\left.\begin{array}{l}\bullet\ \text{float64 } \textit{latitude}\\ \bullet\ \text{float64 } \textit{longitude}\\ \bullet\ \text{float64 } \textit{altitude}\end{array}\right\}\begin{array}{l}\text{The GPS coordinates in floating-point preci-}\\ \text{sion.}\end{array}$$

- float64[9] *position_covariance*: the array of covariance of the estimation, to be reseized as a 3x3 matrix.

- uint8 *position_covariance_type*: the type of covariance of the estimation.

Image data is stored as a **sensor_msgs/Image.msg** [8] message. It is composed by the following messages:

- std_msgs/Header *header*: the same as for the *odometry* message.

- uint32 *height*: the height of the image.

- uint32 *width*: the width of the image.

- string *encoding*: the channel meaning, ordering, size.

- uint8 *is_bigendian*: if the data is bigendian.

- uint32 *step*: the row length in bytes.

- uint8[ ] *data*: the data matrix of size *step*×rows.

A GPS measurement can also be stored as a **geographic_msgs/GeoPoint.msg** [11] message. It is more compact than the *NavSatFix* message and can be used for transforming the measurement into UTM frame.

- float64 *latitude*
- float64 *longitude*
- float64 *altitude*

} The GPS coordinates in floating-point precision.

For the purpose of broadcasting plants centroids in online optimization a custom message is used: **localization_jai_gps/ImageMap.msg**. It stores computed centroids for each image.

- std_msgs/Header *header*: the same as for the *odometry* message.
- int64 *image_number*: the number of the image.
- sensor_msgs/Image *image*: the image message.
- float64 *h*0: first component of the homogeneous coordinates of the centroid.
- float64 *h*1: second component of the homogeneous coordinates of the centroid.
- float64 *h*2: third component of the homogeneous coordinates of the centroid.
- float64 *h*3: fourth component of the homogeneous coordinates of the centroid.
- int64 *u*: row pixel coordinate of centroid in the image.
- int64 *v*: column pixel coordinate of centroid in the image.

4.3.2 ApproximateTime Policy Filter

Simple implementations of the publish/subscribe ROS paradigm involve a subscriber node using a callback which is activated for each incoming message and allows to receive one message at once. However, one might be interested in receiving more than one message at once. ROS provides some synchronization policies for incoming messages which aggregate different message types based on the timestamp, up to 9 different messages. The synchronization is implemented by a Synchronizer filter component [13], which can apply two different synchronization policies: ExactTime and ApproximateTime.

ApproximateTime policy matches messages by applying an adaptive algorithm [14] where each message is used at most once. The algorithm works by minimizing the maximum difference in time in the same set, while matched messages can have different timestamps. A callback using the Approximate-Time policy filter is activated when all expected topics arrive.

4.3.3 Mapviz

Mapviz [79] is a ROS package which allows to visualize 2D navigation data. A background image with relative high resolution can be set to be the underlying map, taken from a local o remote repository. Data are accepted by subscribing to topics with usual publish/subscribe mechanism. Topics are defined by the user and can be of different types. In particular we are interested in sensor_msgs/NavSatFix topics, which contain GPS and odometry estimation messages from our application.

4.4 EIGEN UMEYAMA FUNCTION

In order to compare two trajectories independently of the roto-translation error, we use the algorithm developed by Shinji Umeyama, described in [71]. The function ready-to-use is available in the Eigen library [72] for c++, taking as input two sets of points and a parameter defining the scaling between the two sets. As a result, it returns the homogeneous transformation matrix between the two sets of points:

$$\mathbf{T} = \begin{bmatrix} c\mathbf{R} & \mathbf{t} \\ \mathbf{o}^T & 1 \end{bmatrix} \tag{4.1}$$

where c is the scaling factor.

The algorithm applies Singular Value Decomposition (SVD) to find the best correspondence between a source set of points $\{s_i\}$ and a destination set of points $\{d_i\}$ for i=1..N, by assuming that

$$d_i = \mathbf{R}s_i + \mathbf{T} + \mathbf{V}_i \tag{4.2}$$

where $\mathbf{R}$ is the 3x3 orthonormal rotation matrix of the transformation, $\mathbf{T}$ is the 3D translation vector and $\mathbf{V}_i$ is a noise vector. We solve for the optimal transformation $[\widehat{\mathbf{R}}, \widehat{\mathbf{T}}]$ by minimizing the least squares error

$$\Sigma^2 = \sum_{i=1}^{N} ||d_i - \widehat{\mathbf{R}}s_i - \widehat{\mathbf{T}}||^2 \tag{4.3}$$

Therefore, if we imagine to have already found the optimal translation $\widehat{\mathbf{T}}$, the two point sets should eventually have the same centroid. If we write

$$\begin{aligned} \bar{d} &= \frac{1}{N}\sum_{i=1}^{N} d_i & d_{C_i} &= d_i - \bar{d} \\ \bar{s} &= \frac{1}{N}\sum_{i=1}^{N} s_i & s_{C_i} &= d_i - \bar{s} \end{aligned} \tag{4.4}$$

then we can rewrite eq. (4.3) as

$$\begin{aligned} \Sigma^2 &= \sum_{i=1}^{N} \|d_{C_i} - \widehat{\mathbf{R}} s_{C_i}\|^2 \\ &= \sum_{i=1}^{N} \left(d_{C_i}^T d_{C_i} + s_{C_i}^T s_{C_i} - 2 d_{C_i}^T \widehat{\mathbf{R}} s_{C_i} \right) \end{aligned} \tag{4.5}$$

which we minimize by maximizing last term and hence maximizing Trace($\widehat{\mathbf{R}}\mathbf{H}$) where

$$\mathbf{H} = \sum_{i=1}^{N} s_{C_i} d_{C_i}^T \tag{4.6}$$

Now we compute the SVD of $\mathbf{H}$ as $\mathbf{H} = \mathbf{U}\Lambda\mathbf{V}^T$ and get $\widehat{\mathbf{R}}$ as

$$\widehat{\mathbf{R}} = \mathbf{V}\mathbf{U}^T. \tag{4.7}$$

In order to compute the optimal translation to align the two point sets, it is sufficient to solve the following equation:

$$\widehat{\mathbf{T}} = \bar{d} - \widehat{\mathbf{R}}\bar{s} \tag{4.8}$$

if $\det(\widehat{\mathbf{R}}) = +1$. Though, if we have planar point sets or large amount of noise and therefore $\det(\widehat{\mathbf{R}}) = -1$, the transformation might be a reflection rather then a rotation and a different formula applies. In this special case, rotation is found by

$$\widehat{\mathbf{R}} = \mathbf{U} \begin{bmatrix} 1 & & \\ & 1 & \\ & & \det\left(\mathbf{U}\mathbf{V}^T\right) \end{bmatrix} \mathbf{V}^T. \tag{4.9}$$

4.5 TENSORFLOW

Tensorflow [81] is an open source library for machine learning development. In particular, we used the python library for the image segmentation component of this work. It provides all the needed tools to represent input data, as tensors, and to create and train the neural network architecture. Tensors are a generalization of matrices and are represented as n-dimensional arrays.

Tensorflow also wraps the Keras library, a deep learning library. It is Keras, indeed, that allows to handle most of the steps for generating and training a neural network. The first object we are interested into is the ImageDataGenerator. It allows to load an image dataset and to perform pre-pocessing steps, such as separation into training and validation sets, and some rescaling of the image for example. Tensorflow later transform this object into an iterable Dataset object which can be directly fed to the neural network.

The network architecture is defined as a Sequential Model, by adding Keras layers on top of each others. Layers include convolutional, max-pooling, up-sampling, concatenation and softmax layers, for example. Only few parameters must be defined, such as the number of filters or the activation functions. The model is then fit with dataset and various options can be defined such as early stopping, loss function, and checkpoint creation at each epoch.

Model and trained weights can be saved and reloaded when needed. It is particularly useful to avoid wasting training time. A trained model can also be retrained starting from saved weights.

Training a neural network requires a lot of memory resources. However, it can be parallelized, thanks to the intrinsic nature of input data, by using a GPU. In this way, a lot of time can be saved. Trained models are saved and used as they are, since we do not want to waste more resources for re-training.

5 SOFTWARE ARCHITECTURE

In this chapter we describe the algorithm we developed for addressing the SLAM problem. First, we briefly mention the project from which we started, then we provide a detailed description of each component of our system.

5.1 AN EFFECTIVE MULTI-CUE POSITIONING SYSTEM FOR AGRICULTURAL ROBOTICS

The current work started from an existing project developed by Università La Sapienza performing Simultaneous Localization And Mapping, described in [16]. We briefly describe their work as it is essential to understand how we developed new features on top of it.

They addressed the problem of self-localization in an agricultural environment by designing a 3D global pose estimation system for Unmanned Ground Vehicles (UGV), whose specific target was to face the complexity of a visually repetitive and homogeneous agricultural scene. They stated that conventional landmark based systems are prone to failure, since no globally distinguishable feature can be recognized. In order to gain accuracy in estimation, the authors proposed a method integrating a large number of sensor cues and introduced two domain constraints to better handle the estimation along the z-axis: an altitude prior (Digital Elevation Model - DEM) and a smoothness constraint among adjacent nodes. Domain constraints were valid since the ground plane could be approximated by piece-wise smooth surfaces and the robot traversed the field along the crop rows.

The 3D global pose estimation problem was solved by finding the configuration of the nodes for which the likelihood of the actual measurement was maximized. Since the noise was assumed to be Gaussian, the problem was translated into an iterative least square approach. The pose optimization problem was represented by a graph where nodes were random variables and edges were constraints among them. First of all, the nodes were robot poses estimated at discrete time. They were collected in a vector $\mathbf{X} = \{\mathbf{x}_0, ..., \mathbf{x}_N\}$, where each pose $\mathbf{x}_i = (T_i, R_i)$ is respectively composed by a 3-dimensional translation vector and a 3-dimensional orientation vector. The pose $\mathbf{x}_0$ represented the global reference, and all other poses are defined relative to it. Edges were associated to sensor measurements, which could be of two types: relative motion measure $\mathbf{z}_{ij}$ between node $\mathbf{x}_i$ and node $\mathbf{x}_j$, and global pose measurement $\mathbf{z}_i$ associated to node $\mathbf{x}_i$. The motion model was the Ackermann motion model, extended to non-planar motion case. Each measurement had its information matrix, acting as confidence weight. In order to compute the error in the estimation, the predicted measurement $\hat{\mathbf{z}}_{ij}$, or the predicted global measurement $\hat{\mathbf{z}}_i$, was computed and then compared with the actual one. Leading to:

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}, \quad \mathbf{e}_i = \mathbf{z}_i - \hat{\mathbf{z}}_i. \tag{5.1}$$

So, each edge was characterized by the error and the information matrix of the corresponding constraint. Those constraints representing global pose information were unary constraints, and the edge was a prior edge with absolute information. Hereafter, the list of sensor measurements is proposed without going into details:

1. Relative pose constraints: wheel odometry (wo), visual odometry (vo), elevation constraints among adjacent nodes (Markov Random Field), Ackermann motion model (AMM), LIDAR point-clouds local registration (LID);

2. Global pose constraints: GPS readings (GPS), Digital Elevation Model (DEM), IMU readings (IMU).

It is worth to notice that there is a major difference between relative motion constraints and global measurements. The former are affected by cumulative drift, while the latter are noisy but drift-free, so they could be merged in the estimation as prior information. This is the case of GPS and IMU readings, and of DEM, which is a regularly spaced grid, and can be used as an altitude prior.

The pose graph optimization worked by minimizing the sum of the single objective function terms, which depended on the error of the constraints, computed as in (5.1), with respect to the expected measurement $\hat{\mathbf{z}}_{ij}$. However, it must be highlighted that not all constraints belonged to $SE(3)$, as the poses did, and thus they could not compute the expected measurement for the whole state. In fact, most of the sensors could only observe a portion of the state, such as either the translation or the rotation exclusively. In particular, VO and LID could observe the full 6D motion, WO computed the planar motion as a roto-translation, and MRF and DEM only concerned the z-axis estimation. GPS measurements were converted into translation vectors, IMU provided roll and pith angles, and AMM provided the roto-translation around the instantaneous center of rotation and the rotation along the x and y-axes. Each error was dynamically weighted in the optimization algorithm by the information matrix. Specifically, the weight was computed as the inverse of the covariance matrix, scaled by the travelled distance or an empirical parameter.

An online sliding window approach was also implemented. It is a technique consisting in optimizing at each step only a sub-graph including the most recent nodes. Older nodes were maintained fixed, unless they were connected to the most recent ones and hence re-optimized. Even if it is defined an online optimization, data were actually preprocessed all together prior to the graph creation. This makes the approach unfeasible for robot real-time optimization in the field, where data are available only as they are collected and no global view is achievable, thus it is only suitable for simulation. The global optimization was then performed offline. The algorithm used the Levenberg-Marquardt version of Gauss-Newton least squares implemented in g2o framework [15].

5.2 GENERAL VIEW

In this chapter we deal about the software architecture. From an high level point of view, the structure of the program can be divided into two main components, as it is shown in Figure 5.1. The **Landmarks Identification** component is the development of a model for crops SEP identification, to be used as landmarks in the **Localization and Mapping** component. It includes a neural network training step to set the convolutional neural net weights, and the implementation of a ROS node performing image segmentation and clustering on images, using the previously trained model.

The Localization and Mapping component has two use cases, depending on the nature of the task we are facing. In general, robot localization and mapping can be applied offline, after collecting all data in order to refine the robot trajectory estimate and the environment map, or online for real-time localization. The difference lies in the available quantity of data and in the required processing time to have real-time results. Indeed, online optimization has some peculiar characteristics. First, it uses data at the same time they are collected, thus having a limited vision to the current instant of time. Second, it requires a higher computational effort sinxe the optimization time is limited by the real-time constraint.

In this thesis, offline and online optimization are treated as two use cases of the same component. They have similar structure, although the modules (Figure 5.2) perform slightly different tasks. As reported in the figure, modules include: a data extraction module simulating the real collection of data from the field, a centroid extraction module identifying landmarks in input images, a graph creation module creating the g2o graph, a graph optimization module performing the optimization and finally an error computation and a visualization module. Graph creation and graph optimization modules are interleaved for both use cases, however this happens in a very different way that is explained below. Moreover, the offline optimization can preprocess all data before starting the algorithm.

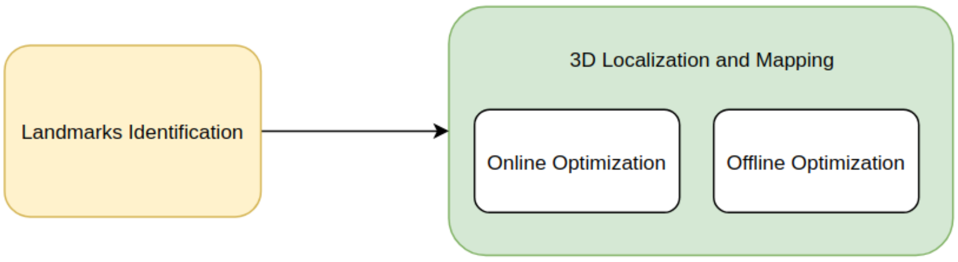


Figure 5.1: The architecture is divided into two main components: in yellow the Landmarks Identification component and in green the 3D Localization and Mapping component with two use cases.

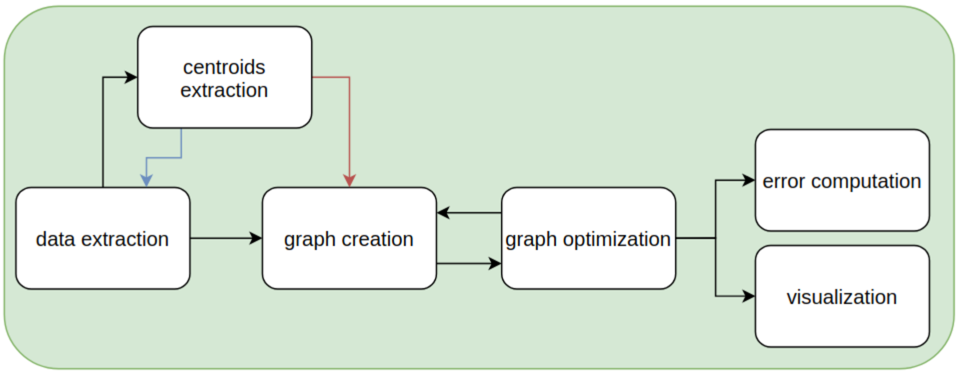


Figure 5.2: Main modules of the Localization and Mapping component. Blue features belong only to the **offline optimization** use case, while red features belong only to the **online optimization** use case. Black features are common to both components.

5.3 LANDMARKS IDENTIFICATION

In the following section we describe the landmarks identification process, which is implemented as the first independent component in Figure 5.1. It is a machine learning process which requires a training phase and the actual classification and post-processing phase of centroids identification. The purpose of this component is to extract the location of the center of mass of the crops included in each input picture, which corresponds to the crops SEP location. Crops SEP are later used as landmarks in the Localization and Mapping component to improve the robot localization accuracy and produce a sparse landmark map, which is the collection of individual feature points position relative to the robot poses. As we can see from Figure 5.3, the component is composed by a convolutional neural network, a clustering algorithm and the actual centroid computation modules. Hereafter we describe the single modules which compose the algorithm.

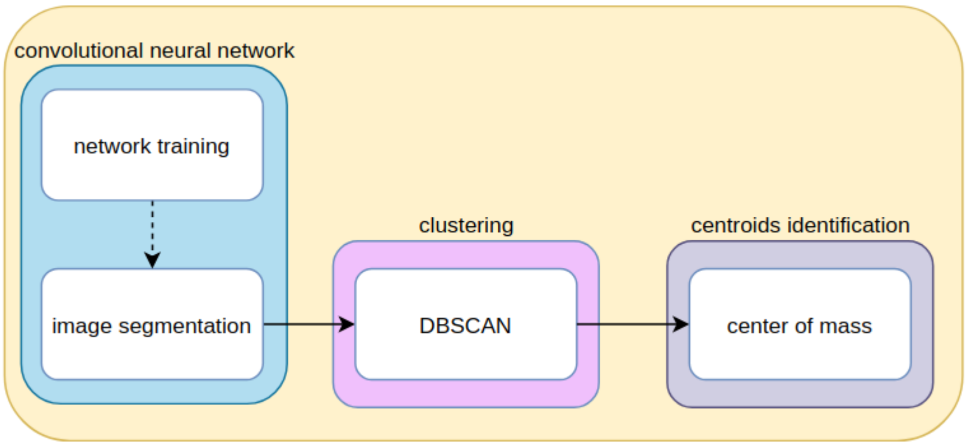


Figure 5.3: Software architecture of Landmarks Identification component. It is divided into three modules: a convolutional neural network, a clustering algorithm and the centroid computation algorithm.

5.3.1 Convolutional Neural Network

The convolutional neural network is the core part of the Landmarks Identification component. It was developed in Python using the Tensorflow package and it started from the U-Net [82] basic structure, but we ended up with an adjusted number of convolution filters per layer. This adjustment was performed by leveraging the validation error with the number of epochs needed for training. The results are explained in the next chapter. The final architecture is the one depicted in Figure 5.4. In order to use the identified pixel coordinates on the corresponding input image, it was essential to have the output image of the same size of the starting one. To achieve higher resolution in the output image, we developed an upsampling path concatenated to higher resolution feature maps from the downsampling layers. In this way, some lost information is recovered.

In order to leverage the background information of the specific domain of the application, we enriched the input volume with additional channels. The idea was to include the features described in [64] as background knowledge for real-time semantic segmentation of crops and weed in precision agriculture. In the mentioned paper, a table of useful features is reported, corresponding to the first column of Table 5.1, while the second column shows which of these features we included in our work.

We ended up with 9 input features, including the RGB channels. The excluded features were left out based on the evaluation of their effectiveness on the result of the image segmentation. Each input layer, indeed, adds a complexity to the neural network, thus slowing the computation, so a trade-off must be reached in order to include only useful features which may help

Input feature List	
$\mathcal{I}_{R}$	yes
$\mathcal{I}_{G}$	yes
$\mathcal{I}_{B}$	yes
$\mathcal{I}_{ExG}$	yes
$\mathcal{I}_{ExR}$	yes
$\mathcal{I}_{CIVE}$	yes
$\mathcal{I}_{NDVI}$	yes
$\mathcal{I}_{HUE}$ (from HSV colorspace)	no
$\mathcal{I}_{SAT}$ (from HSV colorspace)	no
$\mathcal{I}_{VAL}$ (from HSV colorspace)	no
$\nabla_x \mathcal{I}_{ExG}$ (Sobel in x direction on $\mathcal{I}_{ExG}$)	yes
$\nabla_y \mathcal{I}_{ExG}$ (Sobel in y direction on $\mathcal{I}_{ExG}$)	yes
$\nabla^2 \mathcal{I}_{ExG}$ (Laplacian on $\mathcal{I}_{ExG}$)	no
$\mathcal{I}_{EDGES}$ (Canny Edge Detector on $\mathcal{I}_{ExG}$)	yes

Table 5.1: List of semantic features extracting background knowledge to improve semantic segmentation of crops and weed in precision agriculture contexts. The first column is from [64], and the second column specifies which of these features we have included.

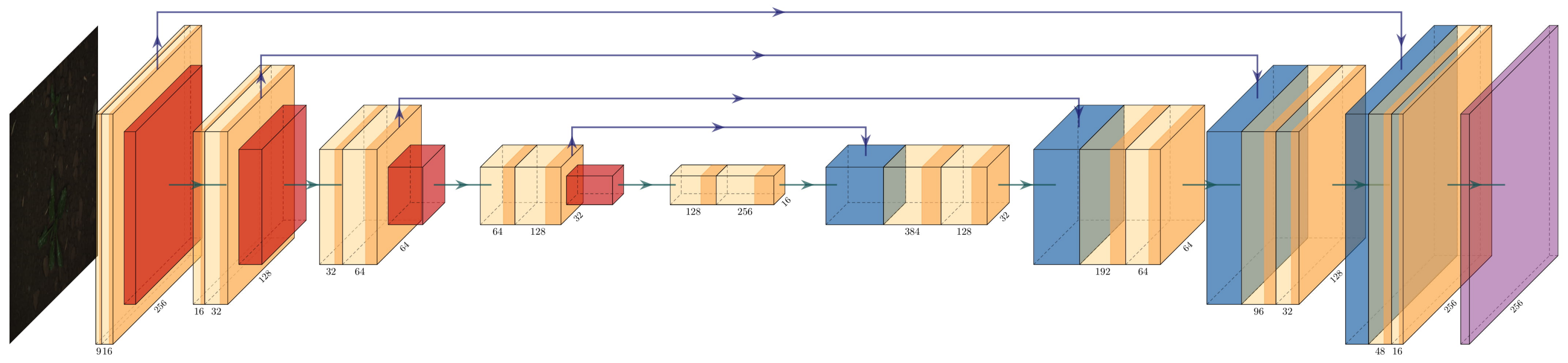


Figure 5.4: Convolutional neural network used for image segmentation. Convolution layers are colored in yellow, max-pooling layers in orange and upsampling layers in blue. The input volume has 9-channels of shape 256x256, and output is a a 3-channels volume with the same size. Last layer in violet indeed is a softmax layer producing the probability of each pixel to belong to the 3 classes: crop, weed or background. Concatenation between layers on contracting and expanding paths helps producing higher resolution maps, since with segmentation is preferable to have an output of the same size of the input, in order to map the probability on the original picture.

the pixel classification. To give a general intuition, the first seven features give information on the color of the pixel and in particular on the greenness level with respect to the other colors, since these features are considered as an hint of where vegetation might be found. The Sobel [84] indexes, instead, point out the margins of the objects in the image, highlighting contours by means of a 3x3 kernel convolution which approximates derivatives and computes the horizontal and vertical changes respectively.

Assuming r, g, b are the RGB channels: r for red, g for green and b for blue, we provide the formulas for the $\mathcal{I}_{\text{ExG}}$, $\mathcal{I}_{\text{ExR}}$, $\mathcal{I}_{\text{CIVE}}$ and $\mathcal{I}_{\text{NDVI}}$ indices:

$$
\begin{aligned}
\mathcal{I}_{\text{ExG}} &= 2g - r - b \\
\mathcal{I}_{\text{ExR}} &= 1.4r - g \\
\mathcal{I}_{\text{CIVE}} &= 0.881g - 0.441r - 0.385b - 18.78745 \\
\mathcal{I}_{\text{NDVI}} &= \frac{g-r}{g+r}
\end{aligned}
\tag{5.2}
$$

Before submitting the input volume to the network, we resized the image to 256x256 and aggregated multiple images with a batch size of 4.

We used a weighted crossentropy (Equation 5.5) loss function with much higher weight given to crops and weed classes than to background class, in order to partially solve the class imbalance problem. We chose weights by trial and error, observing the results obtained by the neural network. The assigned weights are:

$$\mathbf{w} = [0.1, 5, 10] \tag{5.3}$$

where classes are:

$$\textbf{classes} = [\text{background}, \text{weed}, \text{crop}] \tag{5.4}$$

The resulting weighted crossentropy is 5.5, where y_c is the true class label and p_c is the predicted class label for each sample. w_c is the weight, used as scaling factor to weight more the samples belonging to vegetation classes with respect to the background.

$$\mathcal{L} = -\sum_{c=1}^{M} y_c \cdot w_c \log(p_c) \tag{5.5}$$

The output of the convolutional neural network is a three-layers volume containing the probability for each pixel to belong to one of the three classes: crops, weed, background. Without the class imbalance problem we would have assigned to each pixel the class with the highest probability. However, in this case we apply a method to give priority to the two vegetation classes. In particular, all pixels with a crop probability higher than a given threshold are assigned to the crop class with the highest priority, then, all those with

Thresholds on class probabilities			
	background	weed	crop
threshold	-	0.3	0.4

Table 5.2: Thresholds used to assign classes to each pixel. Results for each class are probabilities, so they scale in [0, 1]. Classification is applied in order: weed identified is first, crop can overwrite weed result, background is everything left.

weed probability higher than a second threshold are assigned to weed class. All remaining pixels belong to the background class. Again, because of the class imbalance issue, we chose thresholds (see Table 5.2) lower than 0.5. In this way, we lowered the misclassification error.

Note that in the current application we only care of crops identification, thus pixels assigned to weed class only have the meaning to partially lower the noise of background imbalance. However, since many pixels with a high probability to belong to the crop class, also have an high probability to belong to the weed class, a priority must be defined in order to correctly assign the classification to the crop class. It is crucial to highlight that our main goal is not to perform image segmentation, but to identify crop centroids. So, if the single crop is not perfectly segmented is not a problem, as long as the centroid location still lies in the correct region.

In order to eliminate some noise before the next step of the algorithm, we applied some smoothing on the resulting segmented images. After analyzing the methods in [78], we found that the most appropriate method in this case is the 2D Convolution (Image Filtering) which applies to the image a smoothing filter of 'ones' divided by the size of the filter, basically averaging the result. The size of the filter decides how many neighbour pixels have influence on each pixel. For our application, we found a size of 10x10 to be good enough. After smoothing we thresholded the obtained values to refine classes. In this way, we found class labels to be slightly more robust to noise.

5.3.2 Clustering

As we introduced in Chapter 3, we have used DBSCAN [76] as clustering method to group pixels belonging to the same crop. The simple connected component approach, which is commonly used in similar situations, could not be enough because pixels belonging to the same crop plant were not actually connected in most cases, since we applied only a rough segmentation model on images. We considered that DBSCAN was best model to apply because of the outliers rejection feature and the possibility to identify clusters even if they have non-convex shape. Parameters were tuned by trial and error:

$$\epsilon = 1,\ min\_samples = 10 \tag{5.6}$$

where ϵ is the maximum distance for two samples to be in the same cluster and $min\_samples$ is the minimum number of samples to form a cluster.

The training dataset we worked on includes crop plants from different growth stages, as we wanted to make the algorithm as general as possible. However, with different growth stages, some complexities arise. Indeed, results show that with medium grown plants the algorithm works pretty well, since clusters are big enough to be identified and they are not so big to overlap. However, in early growth stages, the crop plants are so small that clusters would be composed of an extremely low number of pixels and the clustering algorithm would be able to identify the crop only by setting a very small $min\_samples$ value. On the other hand, in late growth stages crop plants overlap and leaves belonging to one plant might be clustered with another one, or even worse two adjacent crop plants could be clustered together.

In order to solve the clustering problem for all growth stages, we had to make the following considerations. For the early growth stage, we tried to lower $min\_samples$ (minimum cluster size), in order to let the algorithm identify tiny crops. However, for later stages, this solution would not work because, as plants grow, pictures include more weeds as well, and pixels belonging to weed that were incorrectly classified in the crop class ended up in independent clusters. In fact, weeds usually have a narrow shape, like simple grass, and after segmentation and smoothing, only few points would survive, and the clustering algorithm would ignore them if the $min\_samples$ parameters was set high enough. However, with a smaller $min\_samples$ value, points representing noise would be classified in some clusters. Therefore, a trade-off was performed to allow the identification of tiny crop plants without including weed.

For the late growth stage, we modified the algorithm by introducing a $max\_samples$ empirical parameter, to separate multiple plants clustered together. If, after performing a first iteration of DBSCAN, one of the resulting clusters was larger than $max\_samples$, a new iteration on that cluster alone was performed, by lowering the minimum distance parameter ϵ to 0.5. This $max\_samples$ threshold was set to 3000. The results, as it is shown in the next chapter, did not perfectly cover all growth stages, but are particularly suitable for the medium growth stage, which is the case for the dataset we used for localization and mapping. However, by tuning the parameters, the method could be generalized to different growth stages.

5.3.3 Centroids Computation

At this point, only one more step is needed: the crop centroid computation. Having the cluster corresponding to the single crop, we computed its center of mass and we translated it to the original image size by means of a simple proportion. Since we considered all pixels to have the same weight, the crop SEP location was computed as:

$$\begin{aligned} c_x &= \frac{1}{M}\sum_{i=0}^{M} x_i \\ c_y &= \frac{1}{M}\sum_{i=0}^{M} y_i \end{aligned}, \tag{5.7}$$

where (c_x, c_y) are the centroid pixel coordinates, M is the size of the cluster and (x_i, y_i) are the coordinates of each pixel in the cluster.

Results are shown in the next chapter.

5.4 DATA PREPARATION

We include in the data preparation section a couple of tasks that must be handled before data can be used by the Localization and Mapping component. They are needed in order to prepare the dataset to be used in the next component.

5.4.1 Topics Filtering

As we have seen in Chapter 4, data are stored in ROS bags, characterized by a long list of topics related to different sensors measurements. We are not interested in the whole list of topics as we focused on the information provided by the wheel odometry estimation, the two GPS estimations, and the RGB images taken by the JAI camera. These data are respectively provided by the following topics:

- /odometry/odometry: wheel odometry measurement used as initial guess for trajectory estimation,
- /gps/leica/fix: used as ground truth for trajectory,
- /gps/ublox/fix: consumer-grade GPS used in the estimation process,
- /camera/jai/rgb/image: camera images from where plant centroids are extracted.

Therefore, the first step of data preparation is to filter out the not interesting topics.

5.4.2 Bags Merging

Data are subdivided into several ROS bags for a matter of portability. The algorithm can be applied to any dataset size, therefore we merged sequential bags to increase the number of measurements at hand. This step is easily done thanks to the pose graph library provided in [73] which is able to fuse two ROS bags ordering the ROS messages based on their timestamp.

5.5 CENTROIDS EXTRACTION

The centroids extraction module is a python ROS node applying the segmentation model and clustering algorithm to extract centroids location from images. Thanks to the ROS publish/subscribe pattern, it subscribes to images topic and publishes computed centroids. A 3D point is obtained by multiplying the predicted pixel coordinates in homgeneous coordinates by the inverse of the projection matrix. The result is a point at infinity representing the direction along which the point lies. To identify the precise point in camera frame, we need to use prior knowledge of the ground height with respect to the camera center, which is given by the dataset description.

Before being included in the localization and mapping estimation, the 3D point needs to be transformed to the robot frame. However we take care of this transformation in next modules. Each published centroid consists of a 3D homogeneous point in the form of a 4D column vector and the corresponding pixel coordinates in the image, as a 2D vector. The returned object also contains the sequence number of the associated image.

5.5.1 Offline SEPs Extraction

The module is used in two different ways for the two optimization use cases. Offline optimization requires the availability of the whole dataset when creating the graph. Indeed, all data are considered at once. For this reason, we decided to compute centroids as a preprocessing step, prior to the localization and mapping application launch. The node /rosbagPlayRawData reads images from the ROS bag and publishes them to the /jai_rgb topic. /segmentImages subscribes to the topic and after applying the segmentation and clustering model, publishes the extracted centroids on the /jai_centroids topic. Results are collected by the node /readImages in a file in order to be directly employed in the graph creation. The ROS graph representing the communication among nodes as produced by rqt_graph is shown in Figure 5.5.

5.5.2 Online SEPs Extraction

Online optimization, on the other hand, works with data as they are collected. The node processes images and generates useful data for the optimization. The resulting architecture is shown in Figure 5.6, where the /segmentIm-

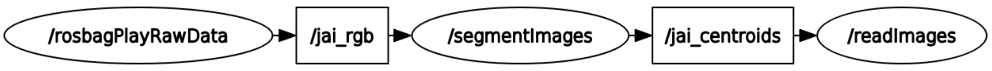


Figure 5.5: ROS nodes handling the centroids extraction. Nodes are represented as circles, while topics are inserted inside rectangles connecting a pair of nodes.

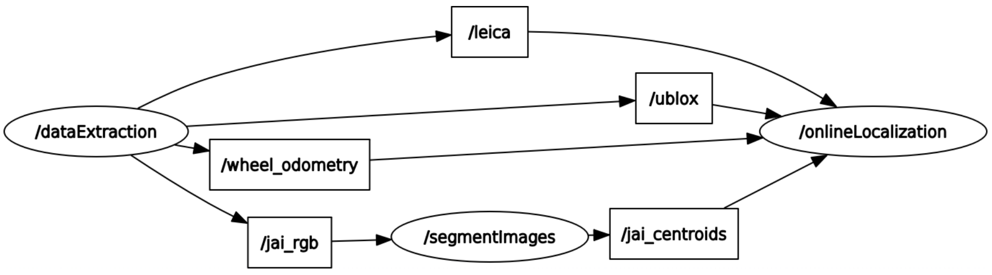


Figure 5.6: ROS nodes involved in the online optimization. Topics are the two GPS sensors measurements, the robot wheel odometry estimation and the RGB images from JAI camera transformed in centroids.

ages node applies the segmentation and clustering model and publishes the extracted centroids. The /dataExtraction node streams data from the ROS bag in real-time mode, and the /onlineLocalization node is responsible for collecting data in specific data structures to create and optimize the trajectory and the map.

5.6 DATA EXTRACTION

The data extraction module is in charge of extracting data from the ROS bags and applying some preprocessing information before they can be used in the g2o framework. The difference between the two optimization use cases lies in the number of measurements analyzed simultaneously. As we mentioned earlier, offline optimization takes the entire dataset, so it reads the entire ROS bags at once and applies the steps described below. Online optimization instead works on one measurement at a time, thus being slightly more imprecise.

In order to create the g2o graph, we need correspondence between pair of sensor measurements. The correspondence is given by the timestamp at which the measurement was collected. In the offline optimization case, we employ a data structure called TSS graph, created in the data extraction module. For online optimization we use synchronous ApproximateTime policy filters to achieve a direct synchronization for data aggregation.

5.6.1 TSS graph

Offline optimization uses an existing library [74] which creates a data structure where all sensor measurements are associated to a node identified by the sequence number of a wheel odometry estimation. Each measurement indeed is matched to the closest odometry estimation in time and is assumed to be taken at the odometry timestamp. The produced structure is defined temporal graph because of the focusing on measurement timestamps.

5.6.2 Transformation of GPS to Odometry

Before using data to estimate robot trajectory, a common reference must be established. The raw localization estimation data are available in two different formats. GPS signals are recorded in the form of a sensor_msgs/NavSatFix message, and wheel odometry is in the form of a nav_msgs/Odometry message. The first one is an absolute measurement with latitude and longitude components, while the second one is relative to the first robot pose in the dataset. We chose the absolute frame, since we wanted to visualize the results in the real world at the end of the trajectory estimation.

In order to be used in the localization and mapping framework, GPS measurements must be transformed to the odometry format. This is done by transforming latitude and longitude coordinates into UTM frame, which is compatible with the odometry measurement. Then we can transform all data to the absolute frame. The first GPS position was set as a global reference point to compute the frame transformation for all data. However, in order to make computations easier, after computing change of frame and saving the reference point, we go back to a relative coordinates frame with respect to the first GPS position.

The work flow is as follows:

1. the GPS is transformed into UTM frame, so coordinates can be used on the x, y and z axes,
2. the wheel odometry is transformed to UTM frame,
3. the UTM frame wheel odometry and GPS are transformed to GPS relative frame

The transformation to GPS relative frame was carried out to have lower scale coordinates, which are more comfortable for numerical stability in trajectory estimation. Resulting data are in odometry format, in a relative frame and ready to be digested by the graph creation module.

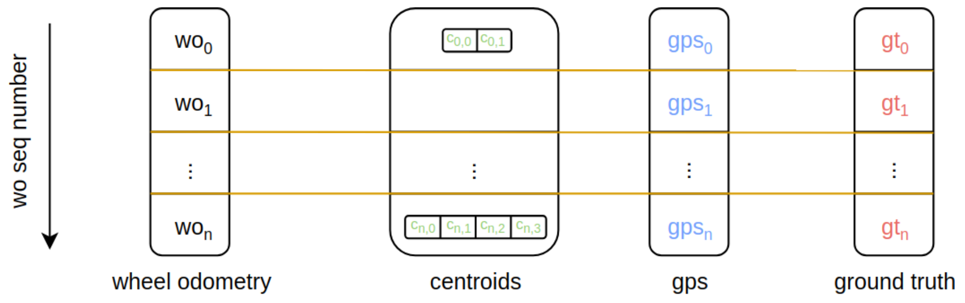


Figure 5.7: Data structure collecting sensor measurements in different arrays. Centroid array contains lists of centroids ($c_{i,j}$) aggregated for the same image number i which is also the wheel odometry sequence number. wo_i corresponds to actual wheel odometry measurement with sequence number $i = seqN$ which is represented as an element in the array with sequence number bigger than zero. All sensor measurements are paired to a wheel odometry measurement based on the timestamp. If we want to retrieve all measurements corresponding to a given wheel odometry measurement, it is sufficient to get elements from measurements arrays with index $i = seqN$ if they have a strictly positive sequence number, otherwise measurement does not exist. For example, in the image above, the wheel odometry estimate wo_0 with sequence number 0 can be associated with two crops centroids $c_{0,0}, c_{0,1}$, with a gps measurment gps_0 and the corresponding ground truth estimate gt_0.

5.6.3 3D Landmark Points Computation

The third step in the data extaction module is to extract centroids from data. In the offline optimization, 3D landmark points are read from a file and stored in a multi-dimensional array, where the list of centroids found in each image is stored at the index corresponding to the image sequence number. For the online optimization, images are streamed to the segmentation node, which, after extracting the corresponding list of centroids, returns the result to the optimizer module.

5.6.4 Data Structures

The extracted data are stored into arrays. For each sensor we have a different array, but they are all indexed the same way to have an easy correspondence for measurements belonging to the same node. The employed index is the ROS sequence number from the odometry message header, which comes from the TSS graph in offline case, and from message filters in online case. Figure 5.7 shows an example of sensor measurements collected in corresponding arrays.

5.7 GRAPH CREATION

In this section, we describe the general method we used to solve the robot localization and mapping problem. It is solved via a graph-based SLAM approach using the g2o framework. Therefore, the goal is to find the configuration of the nodes which maximizes the actual measurements likelihood. The graph creation module handles the development of the g2o graph. Offline and online optimization handle this process in a slight different way, so we describe the key points of the algorithm separately. However, in this section we want to give some common background useful for both tasks. As we described before, a g2o graph is composed by nodes corresponding to optimizable variables and edges representing constraints between pair of nodes or for single node measurements. In our algorithm, we used both binary and unary constraints.

5.7.1 Adopted g2o Edges and Vertices

The following is the list of g2o vertices used in the context of this thesis.

1. VertexSE3: public BaseVertex<6, Isometry3> $\rightarrow$ 3D translation and 3D rotation of a robot pose,
2. VertexLine3D: public BaseVertex<4, Line3D> $\rightarrow$ 4-dimensional vector representing the minimal parametrization increment in orthonormal form, while a 6-dimensional vector is used for representing the same 3D line in Plücker coordinates.

And hereafter a list of g2o edges.

1. EdgeSE3: public BaseBinaryEdge<6, Isometry3, VertexSE3, VertexSE3> $\rightarrow$ translation and rotation between two odometry poses,
2. EdgeSE3Prior: public BaseUnaryEdge<6, Isometry3, VertexSE3> $\rightarrow$ translation and rotation from odometry pose to GPS measure (prior estimate, not corresponding to any vertex).

In addition to these edges, custom ones have been implemented. In this section, all adopted vertices and edges are described. The ones that do not belong to the previous list were implemented from scratch.

5.7.2 Nodes

Robot poses and crop rows are the nodes of the graph and they are collected in a vector $\mathbf{X} = \{x_{robot,0}, ..., x_{robot,N}, x_{line,0}, ..., x_{line,M}\}$. Robot poses are implemented as g2o::VertexSE3, their estimate, which is composed by a translation and an orientation component, is initialized with the wheel odometry estimation. Crop rows are implemented as g2o::VertexLine3D, initialized with the 3D Plücker line estimation and internally represented with the orthonormal representation. Equation (5.8) shows the mathematical representation of the robot pose $x_{robot,t}$ at time t and of the m^{th} line $x_{line,m}$.

$$\begin{aligned} x_{robot,t} &= (t_x, t_y, t_z, q_x, q_y, q_z) \\ x_{line,m} &= (n_x, n_y, n_z, v_x, v_y, v_z) \end{aligned} \tag{5.8}$$

where $\mathbf{t} = (t_x, t_y, t_z)$ is the translation vector and (q_x, q_y, q_z) are the first three components of a unit quaternion, which imply $q_w = \sqrt{1 - (q_x^2 + q_y^2 + q_z^2)}$. The line estimate is composed by a normal vector $\mathbf{n} = (n_x, n_y, n_z)$ and a direction vector $\mathbf{v} = (v_x, v_y, v_z)$. The robot pose is often represented as a transformation matrix $X_{robot,t}$ derived from $x_{robot,t}$, and the Plücker line $x_{line,m}$ as a Plücker matrix $X_{line,m}$, where the respective relation is explained in Chapter 3.

5.7.3 Landmarks Parametrization

Landmarks are described by custom g2o parameters, which represent fixed constraints in the graph. Indeed, we wanted to model fixed constraints relative to robot poses, since we did not provide an identification method for the same landmark, if observed twice. So landmarks estimation relative to the pose remains fixed and no loop closure optimization is performed. Landmarks are represented by CropPositionParameters in the g2o graph.

The mathematical representation of CropPositionParameters is:

$$param_{crop} = (t_x, t_y, t_z) \tag{5.9}$$

where the vector $\mathbf{t} = (t_x, t_y, t_z)$ is the position in the camera coordinates frame. This is particularly useful to compute the crop position in world frame by $X_{robot,t} \cdot \left((extrinsic\_matrix)^{-1} \cdot param_{crop}\right)$. Where the extrinsic matrix is shown in Equation (3.11).

5.7.4 Edges

g2o edges model constraints between robot poses, assumptions on single poses, and constraints on observed landmarks. In the description below, $\mathbf{z}_{edge}$, $\hat{\mathbf{z}}_{edge}$, Ω_{edge}, $\mathbf{e}_{edge}$ represent the actual and the predicted measurement, the information matrix and the error of the edge, respectively. When $\hat{\mathbf{z}}_{edge}$ is not specified, it means it is computed as the actual measurement

$\mathbf{z}_{edge}$. The difference is that $\mathbf{z}_{edge}$ is always computed during the edge initialization phase, with the actual nodes value, while the predicted measurement is computed during optimization, therefore the actual node values are optimized. All vectors are column vectors, even if not specified for brevity.

EdgeSE3Prior

We inserted Ublox GPS measurements in the graph as prior position estimations for robot poses. Indeed, we considered GPS to be globally more reliable than wheel odometry estimation. We created a unary constraint to only optimize the robot pose, while the GPS measurement is kept fixed. Measurement is composed by the GPS position and a unitary rotation, since GPS does not provide attitude information. Information matrix was built to give zero weight to the rotation component. The constraint on $x_{robot,t}$ can be represented as:

$$\begin{aligned}
\mathbf{z}_{GPS} &= (\mathbf{t}_{GPS}, \mathcal{I}_3)\,,\ \hat{\mathbf{z}}_{GPS} = x_{robot,t} \\
\Omega_{GPS} &= \begin{bmatrix} \lambda_x & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_z & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
\mathbf{e}_{GPS} &= (T_{\mathbf{z}_{GPS}})^{-1} \cdot T_{\hat{\mathbf{z}}_{GPS}}
\end{aligned} \tag{5.10}$$

where λ_x, λ_y, λ_z scale the information based on the reliability we want to give to the GPS measurement and can be set from the graphical interface. In our experiments, we kept λ_z lower because GPS information on z-axis in not as accurate as on xy-plane. $T_{\mathbf{z}_{GPS}}$ is the transformation matrix built from edge measurement, and $T_{\hat{\mathbf{z}}_{GPS}} = X_{robot,t}$.

In addition to the GPS constraint, we introduced a new domain assumption, represented by the just described edge. In the application, we had the robot travelling along crop rows. Therefore, we were able to use the estimate of the crop row direction and the total distance travelled from the beginning of the crop row to infer the supposed robot position, as if it travelled along a straight line. Basically, we projected the total distance along the line to have a better estimate of the current position. The distance travelled was given by the wheel odometry estimation and the line direction by the current crop row estimate. The information matrix $\Omega_{alongLine}$ is the same as in GPS prior edge, but can use a different scaling factor. The same holds for error, while measurement is computed as follows:

$$\begin{aligned}
\mathbf{d}_{\text{rectified}} &= \|\mathbf{t}_{\text{robot},t} - \mathbf{t}_{\text{first\_vertex}}\| \cdot \mathbf{v}_{\text{line}} \\
\mathbf{t}_{\text{alongLine}} &= \mathbf{t}_{\text{first\_vertex}} + \mathbf{d}_{\text{rectified}} \\
\hat{\mathbf{z}}_{\text{alongLine}} &= \left(\mathbf{t}_{\text{alongLine}}, \mathcal{I}_3\right)
\end{aligned} \tag{5.11}$$

where $\mathbf{d}_{\text{rectified}}$ is the distance projected along the line, and $\mathbf{t}_{\text{robot},t}$ and $\mathbf{t}_{\text{first\_vertex}}$ are the translation estimation from wheel odometry of the current robot pose $x_{\text{robot},t}$ and of the first pose along the current line, respectively. Thus, the measurement includes the rectified translation $t_{\text{alongLine}}$ along the line relative to the first pose on the same line, and an identity rotation matrix $\mathcal{I}_3$ which is not weighted in the information matrix since we do not want to bias the optimization.

Because of its incremental nature, the constraint only worked in online optimization, as the line estimate needed to be refined while more crop plants were observed. In the offline optimization, we created a different edge, which we discuss later, where the line estimate is optimizable as well: EdgeTranslationAlongLine.

EdgeSE3

Robot poses can be constrained by relative wheel odometry estimations. Transformation matrix between $x_{\text{robot},t}$ and $x_{\text{robot},t-1}$ was computed and set as measurement for the binary edge.

$$\begin{aligned}
\mathbf{z}_{WO} &= (X_{\text{robot},t-1})^{-1} \cdot X_{\text{robot},t} \\
\Omega_{WO} &= \begin{bmatrix} \lambda_x & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_z & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_{rx} & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_{ry} & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_{rz} \end{bmatrix} \\
\mathbf{e}_{WO} &= (T_{\mathbf{z}_{WO}})^{-1} \cdot T_{\hat{\mathbf{z}}_{WO}}
\end{aligned} \tag{5.12}$$

EdgeSE3 has a complete information matrix, where rotation part is scaled by non-zero parameters λ_{rx}, λ_{ry}, λ_{rz}.

We used an EdgeSE3 constraint also for modeling the relative robot movement with the Ackermann motion model. As actual measurement we used $\mathbf{z}_{\text{Ackermann}} = T_{\text{Ackermann}}$ which is the transformation computed by Ackermann model and hence neglecting noise due to the uneven ground. Error is computed as for wheel odometry constraint, while information matrix gives reliability only to the xy-plane:

$$\Omega_{\mathrm{Ackermann}} = \begin{bmatrix} \lambda_x & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_{rz} \end{bmatrix} \tag{5.13}$$

Indeed, non-zero rows correspond to translation on x and y axes and to the yaw rotation.

EdgePoseSE3Line3D and EdgePoseSE3PriorLine3D

Crops are constrained to lie on a line, because of the domain assumption that they were sown in a row. We created two types of line edges. For one edge we have both crop plants position and line estimate optimizable. For the other edge we can keep the line estimate fixed and optimize crops position to lie on the line. In both cases the error to minimize is the distance from the line, which should tend to zero.

In EdgePoseSE3Line3D, both line and crops are optimizable. The edge must be connected to the line estimate $x_{line,m}$ and the current robot pose $x_{robot,t}$. It takes two parameters: the first parameter is the landmark position in the camera frame $param_{crop}$ and the second is the extrinsic matrix, used in order to transform the landmark from camera frame to robot frame. Robot pose is needed to transform the landmark into world frame, in order to be used for line interpolation.

$$\mathbf{z}_{cropLine} = \mathbf{0}, \ \hat{\mathbf{z}}_{cropLine} = \mathbf{d}_{crop-line}$$
$$\Omega_{cropLine} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \tag{5.14}$$
$$\mathbf{e}_{cropLine} = \hat{\mathbf{z}}_{cropLine}$$

Distance $\mathbf{d}_{crop-line}$ was computed by transforming the crop position to world frame and computing the distance by using projective geometry. In the g2o framework, the error vector must have the same size as the measurement, while projective geometry only provides a formula to compute the norm of the distance. To compute a vector distance, we projected the landmark point on to the line (foot of the perpendicular) and computed the 3D distance. To project the landmark position on the line, we computed the line passing through the crop position and perpendicular to the crop row, thus obtaining the foot of the perpendicular, and took the vector difference between the foot and the crop plant 3D points as the desired distance. Algorithm 1 sums up required computations.

Algorithm 1 Error computation for crop line distance in edge EdgePoseSE3Line3D.

```
$\mathtt{extrinsic\_matrix}, \mathtt{param}_{\mathtt{crop}} \Leftarrow \mathtt{edgeParameters}$
$\mathtt{crop}_{\mathtt{robot}} = (\mathtt{extrinsic\_matrix})^{-1} \cdot \mathtt{param}_{\mathtt{crop}}$
$\mathtt{crop}_{\mathtt{world}} = X_{\mathtt{robot},t} \cdot \mathtt{crop}_{\mathtt{robot}}$    ▷ Crop position in world frame

$\mathtt{plucker\_matrix} = \mathtt{toPluckerMatrix}(x_{\mathtt{line},m})$    ▷ Equation (3.22)
$\mathtt{plucker\_dual} = \mathtt{pluckerDualMatrix}(\mathtt{plucker\_matrix})$    ▷ Equation (3.20).

$d = \|(\mathtt{plucker\_dual} \cdot \mathtt{crop}_{\mathtt{world}})_{1:3}\|$    ▷ Equation (3.32).

$\mathtt{plane1} = \mathtt{planeThroughPointAndLine}(\mathtt{plucker\_matrix}, \mathtt{crop}_{\mathtt{world}})$
        ▷ Plücker dual is the intersection of two planes: plane1 and plane2.
$\mathtt{plane2} = \mathtt{perpendicularPlaneThroughLine}(\mathtt{plucker\_matrix}, \mathtt{plane1})$
                                                            ▷ Equation (3.20).

$\mathbf{d}_{\mathtt{crop-line}} = d \cdot \mathtt{plane2}_{1:3}$
        ▷ Plane in projective geometry are represented as a 4D vector where
    the first 3 components represent the normal vector to the plane, Equation
    (3.13). We scale it by the norm of the distance to have the vector distance.
```

EdgePoseSE3PriorLine3D uses the same computation but keeps line fixed when optimizing. To do so, the line estimate is passed as actual measurement: $\mathbf{z}_{\mathtt{cropLinePrior}} = x_{\mathtt{line},m}$.

EdgePosePlanePrior

Robot poses can be constrained to lie on the same plane through the unary constraint EdgePosePlanePrior, because of the domain assumption that an agricultural field is roughly flat. It takes the estimate of the so called robot plane as measurement and minimizes the distance of the pose by projective geometry. We call it robot plane because the center of the robot, which is used as reference point, is at an height of about 70 cm above the ground plane.

We use it as reference plane, indeed $z = 0$ on the robot plane. We have the following parameters:

$$\begin{aligned}
\mathbf{z}_{\mathrm{robotPlane}} &= (0, 0, 1, 0),\ \hat{\mathbf{z}}_{\mathrm{robotPlane}} = (\mathbf{t}_{\mathrm{robot},t}, 1) \\
\Omega_{\mathrm{robotPlane}} &= \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & \lambda \end{bmatrix} \\
\mathbf{e}_{\mathrm{robotPlane}} &= \hat{\mathbf{z}}_{\mathrm{robotPlane}} \cdot \mathbf{z}^{T}_{\mathrm{robotPlane}}
\end{aligned} \tag{5.15}$$

where error corresponds to the distance between robot position and robot plane in projective geometry.

EdgePriorLine3D

If we have an initial guess on crop row position and orientation, we can use the unary edge EdgePriorLine3D on $x_{\mathrm{line},m}$. Edge parameters are the following:

$$\begin{aligned}
\mathbf{z}_{\mathrm{priorLine3D}} &= (\mathbf{n}_{\mathrm{prior}}, \mathbf{v}_{\mathrm{prior}}), \\
\hat{\mathbf{z}}_{\mathrm{priorLine3D}} &= (\mathbf{n}, \mathbf{v}) \\
\Omega_{\mathrm{priorLine3D}} &= \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \\
\mathbf{e}_{\mathrm{priorLine3D}} &= \mathbf{z}_{\mathrm{priorLine3D}} - \hat{\mathbf{z}}_{\mathrm{priorLine3D}}
\end{aligned} \tag{5.16}$$

where $(\mathbf{n}_{\mathrm{prior}}, \mathbf{v}_{\mathrm{prior}})$ is the prior line used to correct the estimate of $x_{\mathrm{line},m} = (\mathbf{n}, \mathbf{v})$.

EdgePointPlanePrior

Also the landmark position can be constrained to lie on a plane. EdgePoint-PlanePrior minimizes the distance of the crop SEP from the ground plane, which in our world frame was about 70 cm (ground_height) below the reference $z = 0$ plane. The edge is characterized as follows:

$$\begin{aligned}
\mathbf{z}_{\mathtt{pointPlane}} &= (0, 0, 1, -\mathtt{ground\_height}), \\
\hat{\mathbf{z}}_{\mathtt{pointPlane}} &= \mathtt{crop}_{\mathtt{world}} \\
\Omega_{\mathtt{pointPlane}} &= \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & \lambda \end{bmatrix} \\
\mathbf{e}_{\mathtt{pointPlane}} &= \hat{\mathbf{z}}_{\mathtt{pointPlane}} \cdot \mathbf{z}^{T}_{\mathtt{pointPlane}}
\end{aligned} \tag{5.17}$$

where $\mathtt{crop}_{\mathtt{world}}$ represents the crop SEP in world frame and is computed as described before starting from $\mathtt{param}_{\mathtt{crop}}$ in camera frame.

EdgeLinePlanePrior

Since crops are assumed to lie on the ground, we can add a new constraint on the crop row itself. Again, by using projective geometry (Equation (3.29)) we minimize the distance of the line from the ground plane.

$$\begin{aligned}
\mathbf{z}_{\mathtt{linePlane}} &= (0, 0, 1, -\mathtt{ground\_height}), \\
\hat{\mathbf{z}}_{\mathtt{linePlane}} &= \mathtt{toPluckerMatrix}(\mathtt{x}_{\mathtt{line,m}}) \\
\Omega_{\mathtt{linePlane}} &= \begin{bmatrix} \lambda & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & \lambda \end{bmatrix} \\
\mathbf{e}_{\mathtt{linePlane}} &= \hat{\mathbf{z}}_{\mathtt{linePlane}} \cdot \mathbf{z}_{\mathtt{linePlane}}
\end{aligned} \tag{5.18}$$

where $\mathtt{toPluckerMatrix}$ is a function implemented to transform the line from vector to matrix form, in Plücker coordinates.

EdgePriorLineDirection

If we have a prior guess on the line direction $\mathbf{v}_{\mathtt{prior}}$ and we want to make the crop row direction $\mathbf{v}$ parallel to the desired direction vector, we can use EdgePriorLineDirection:

$$\begin{aligned}
\mathbf{z}_{\mathtt{priorLineDir}} &= \mathbf{v}_{\mathtt{prior}}, \ \hat{\mathbf{z}}_{\mathtt{priorLineDir}} = \mathbf{v} \\
\Omega_{\mathtt{priorLineDir}} &= \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \\
\mathbf{e}_{\mathtt{priorLineDir}} &= \hat{\mathbf{z}}_{\mathtt{priorLineDir}} \times \mathbf{z}_{\mathtt{priorLineDir}}
\end{aligned} \tag{5.19}$$

where we are enforcing the property that parallel vectors have a zero cross product, Equation (3.33).

EdgeLineLine3D

In order to constraint two free crop rows $x_{\mathrm{line},i}$ and $x_{\mathrm{line},j}$ to be parallel we can use EdgeLineLine3D. The difference with respect to EdgePriorLineDirection is that both line directions are optimizable. It is characterized by the following parameters:

$$\begin{aligned}
&\mathbf{z}_{\mathrm{lineDir}} = \mathbf{0},\ \hat{\mathbf{z}}_{\mathrm{lineDir}} = \mathbf{v}_i \times \mathbf{v}_j \\
&\Omega_{\mathrm{lineDir}} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \\
&\mathbf{e}_{\mathrm{lineDir}} = \hat{\mathbf{z}}_{\mathrm{lineDir}}
\end{aligned} \tag{5.20}$$

EdgeTranslationAlongLine

Similarly to the prior edge projecting the wheel odometry translation distance along a line, we can have a prior guess on the robot position by means of the line estimation. In online optimization, it works because at each sliding window, we have a more precise estimation of the current line, and we can use the direction vector to rectify the travelled distance. However, in the offline optimization case we have to concurrently optimize the line estimate and the robot position, without relying on previous optimizations. We pass as offset parameter the translation of the first robot pose along the line $\mathbf{t}_{\mathrm{first\_vertex}}$ and use it as a reference point to compute the distance travelled along the crop row.

$$\begin{aligned}
&\mathbf{z}_{\mathrm{alongLine}} = \mathbf{0},\ \hat{\mathbf{z}}_{\mathrm{alongLine}} = \mathbf{t}_{\mathrm{alongLine}} \\
&\Omega_{\mathrm{alongLine}} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} \\
&\mathbf{d}_{\mathrm{rectified}} = \|\mathbf{t}_{\mathrm{robot},t} - \mathbf{t}_{\mathrm{first\_vertex}}\| \cdot \mathbf{v}_{\mathrm{line}} \\
&\mathbf{t}_{\mathrm{alongLine}} = \mathbf{t}_{\mathrm{first\_vertex}} + \mathbf{d}_{\mathrm{rectified}} \\
&\mathbf{e}_{\mathrm{alongLine}} = \hat{\mathbf{z}}_{\mathrm{alongLine}} - \mathbf{t}_{\mathrm{robot},t}
\end{aligned} \tag{5.21}$$

where we use as measurement the supposed translation component $\mathbf{t}_{\mathrm{alongLine}}$ computed by projecting along the line the distance travelled from the first robot pose $\mathbf{t}_{\mathrm{first\_vertex}}$ on the line. $\mathbf{d}_{\mathrm{rectified}}$ is the rectified distance between the two vertices and $\mathbf{v}_{\mathrm{line}}$ is the direction vector of the line along which the robot is travelling. The measurement, differently from the corresponding prior edge for the online case, only includes the translation component, so both measurement and error are 3D vectors.

Jacobians

In the g2o framework, custom implemented edges need the Jacobian of the error function with respect to the linked node variables to be estimated. One can either provide the analytic computations for the matrix or use the numerical matrix which is automatically computed if no analytic Jacobian exists. For all custom edges, we computed the Jacobian, however, we left to the user the possibility to enable its use in the optimization. While it speeds up the computations, it also affects the growth of the damping factor in the Levenberg-Marquardt optimization algorithm and needs further studying.

5.8 OPTIMIZATION

In this section, we describe the two main use cases which perform offline and online optimization, respectively.

5.8.1 Offline optimization

Offline optimization takes as input the whole dataset, it creates the entire g2o graph and optimizes it.

Data extracted are passed to the graph creator module which iterates over the arrays to create vertices and edges for the g2o graph. The configuration file and the graphical interface allow the user to enable constraints to be inserted in the graph. In the following description we consider all constraints enabled.

First, we iterate over the wheel odometry and GPS estimations. Measurements corresponding to the same robot pose have the same index, which is related to the timestamp, thus the two data arrays can be iterated together. Vertices representing robot poses are created using the header sequence number as id number. For each new vertex with $\mathtt{id} = t$, a set of constraints relative to the previous one (previous vertex has $\mathtt{id} = t - 1$) are created:

- Wheel odometry (WO): Transformation $T = (X_{robot,t-1})^{-1} \cdot X_{robot,t}$ is computed and set as measurement of an EdgeSE3 between the two vertices.
- Ackermann (Ack): Motion model is used to compute the theoretical transformation neglecting any influencing external variable such as drift. T_{Ack} is set as measurement for an EdgeSE3 between the two vertices.

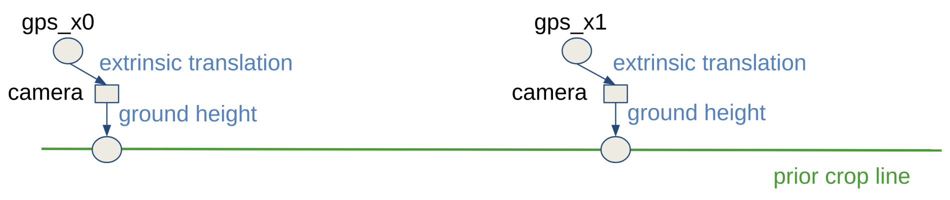


Figure 5.8: Prior crop row built from GPS estimation.

Then, some prior edges on the robot pose are created.

- GPS: If GPS is enabled, an EdgeSE3Prior is added to the graph with the GPS measurement, otherwise, for the first n poses we fix the initial estimation by using an EdgeSE3Prior. This is needed because, if no absolute constraints are enabled, the SLAM algorithm has no reference point for the trajectory estimation.
- Planarity (PlanePose): A prior edge with the robot plane estimation is set to constraint the pose to lie on the desired plane.

While iterating on the two arrays, the yaw variation between successive robot poses is observed. When it overcomes a threshold, we consider the robot to have inverted the path and started following a different crop row. The identifier of each vertex on the new line is stored to build the lines in next step.

The last step in this loop is to interpolate the GPS measurements along the same line to store an initial guess on the possible crop row direction. Two measurements at a fixed distance are collected and used to create a Plücker line. The Plücker line is transposed to the ground, using the knowledge about the camera mounting point with respect to the robot center and the distance to the ground. This knowledge is encoded in the extrinsic matrix between robot and camera frame. The resulting line is that observed if crops were exactly placed at the center of the camera frame, which is a good guess, as the robot actually follows the crop row below itself, as in Figure 5.8.

Once the first loop ends, a new loop starts to create all edges relative to the observed landmarks. For each vertex already inserted in the graph, we use the observed crops SEP to add new constraints. When a new crop row starts, first we initialize the Plücker line and then add all connected landmarks to the graph. The line is initialized by taking two SEPs at a fixed distance and computing the line between them by projective geometry, as in Equation (3.16). A line vertex is created with this estimate.

The collected landmarks are in camera frame. Even if they are in the correct frame to be inserted in the g2o graph, they need to be transformed to world frame for visualization purposes. The visualization module is explained in details in next section, however, the computation steps are made during the graph creation task. Namely, camera frame landmark is transformed to robot frame by means of extrinsic matrix in Equation (3.11), and then it is multiplied by the robot transformation matrix to obtain the position in world frame.

Landmark points on a line are used to create constraints of two kinds, which are mutually exclusive. If we have a prior estimate of the line and we do not want to optimize it, we use an EdgePoseSE3PriorLine3D constraint, where the line is fixed, using the prior estimate initialization. On the other hand, if we need to optimize the line as well, we use the edge EdgePoseSE3Line3D. Both edges are connected to the landmarks in camera frame (g2o parameter), the robot vertex and the line estimate. The last one can be represented either as the edge measurement in the first case, or as a vertex in the second case. The error minimizes the distance between the landmark position and the line.

Landmarks can be constrained to lie on the ground plane with the prior edge EdgePointPlanePrior, as the line by means of the EdgeLinePlanePrior. Lines can also be constrained to be parallel by means of EdgePriorLineDirection. Finally, the edge EdgeTranslationAlongLine projecting the travelled distance along the line can be added.

Once the graph structure is complete, it is written into a file, which is read from the optimizer module. The g2o optimizer uses Levenberg Marquardt algorithm to minimize the cumulative error of all constraints and find the best configuration for robot poses and lines, which in our application are the variables to optimize. Edges are robustified by Huber kernel, as in Equation (3.48).

Even if in the offline case we had the whole dataset available at once, we found more convenient to optimize the robot poses along one line at a time. Indeed, if we start by optimizing the first line, then we can use the optimized value to correct some of the drift accumulation error in the poses initialization of next line. We can also employ the optimized line in the next optimization to have a better estimate of both of them. Results show a little improvement in the second robot trajectory estimate along the second line.

When the optimization is complete, optimized nodes are read from the file where output is stored and they are passed to the visualization module. Error is computed as described in the Error Computation section.

5.8.2 Online optimization

In the online optimization we used multiple threads to manage the concurrent data collection, graph creation, optimization and visualization tasks. Thanks to a set of callbacks, the application stores data arriving in the form of ROS messages into arrays indexed by the WO sequence number, which provides an easy correspondence between different sensor measurements. Callbacks are activated by a ROS message filter applying the ApproximateTime policy,

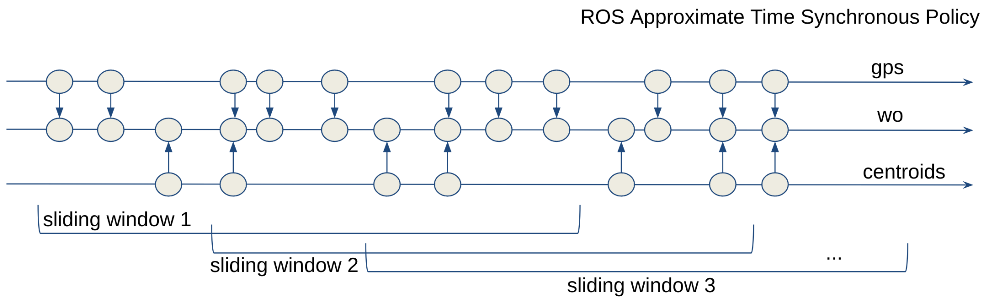


Figure 5.9: A series of ROS message filters synchronize data published from the data extraction module by timestamp. Data are stored into a list of arrays where correspondence among different sensors is given by the WO sequence number, that is used as index. The sliding window size is computed on the WO array.

as described in Chapter 4. Once the data arrays reach the size of the sliding window at first, or a new number of data measurements equal to the step size is collected, the sliding window optimization is triggered, as it is shown in Figure 5.9. The graph is created and optimized and the results are visualized, while still collecting new data. In this way the data are optimized multiple times as they enter in several sliding windows and the trajectory estimation is gradually refined.

The graph creation task is similar to the offline one, however we needed to store the previously created vertices, edges and parameters in global data structures to be reused in next iterations. Thus, before creating the graph, already existing nodes and edges are retrieved with their optimized value if they enter the current window. Then, new nodes and edges are created. Also, we needed some global structures to store the information about the current line, which have to be updated in all windows, since the line could have the first endpoint in one window and the second endpoint in several windows later. In the meanwhile, all landmarks connected to the line are stored, but the edges are only added to the graph when the line is finally initialized.

Another difference relative to the offline case is that all computations are made inside the same unique loop. Nodes are the same, while edges are slightly different. The edge that is computing the translation along the line can now confidently use the previous optimized line estimate (which is continuously refined), thus it is only a prior estimate on the robot vertex, as in Equation (5.11), without the need to optimize the line estimate as well. Also the constraints between line and landmarks change. Indeed, for each optimization only the crops observed in the current windows would be included, but this would make the estimate vary too much. Therefore, every time the line is optimized after initialization, we include the first connected landmarks edges, with a prior on their already optimized estimate. In this way, they can bias the new optimization to avoid excessive fluctuations.

In addition to this, the previous optimized line estimate can be used to instance an EdgePriorLine3D, to add a further bias on new optimization. It is necessary to say that all prior estimations are eventually updated after each optimization, as they are only considered as guesses and cannot be too different from the current values. The only estimates which are never updated are the GPS measurements.

After the graph is created and saved into a file, optimization is performed on the current window and global data structures are updated, while visualization module takes care of the visualization. In addition to the sliding window optimization, we have the concurrent re-optimization of the previous trajectory estimation. The sliding window indeed must be kept small to work online, but a larger window works better, and if we start from already optimized values, only few iterations are needed. Therefore, we apply three different re-optimizations:

- Re-initialization of the line: Crop row needs quite a few samples to be effectively estimated, but since the online algorithm cannot wait too long before starting the optimization, we first initialize the line with a small number of centroids. Then, when we have more available observations, we re-compute the Plücker line, set the current optimized value as a prior constraint on the new line (EdgePriorLine3D), and re-optimized the graph corresponding to the portion of the line up to this point.
- Periodic re-optimization: Every k sliding window optimizations we consider a larger window and re-optimize the g2o graph. This gives us a better estimate of the robot trajectory and of the landmarks map.
- Complete line re-optimization: When a new line is started, the previous one is completely re-optimized to have a final estimation. Since data are no more optimized after this optimization, error can now be computed and final trajectory and mapping can be stored in a ROS bag.

The key idea behind these re-optimizations is that more available data mean greater accuracy of the result. The reason to keep the sliding window as small as possible, however, is to have an algorithm working in real time.

5.9 GRAPHICAL INTERFACE AND VISUALIZATION

The application provides a graphical interface that allows the user to both interact with the graph configurations and visualize data. Graph configurations are set in a yaml configuration file, however it is actually much more easy for the user to do it from the graphical interface. Almost every setting can be modified by the interface, such as the enabling and disabling of constraints, the scaling factor of the information matrix, the number of optimization iterations, the robust least square optimization, and so on.

Regarding the data visualization, there are two different visualization methods. We think that picturing results is essential in order to have a first qualitative measure of their accuracy. The reason to have two methods is that one is implemented in the graphical interface, by means of Qt OpenGL [83], and the other is using ROS Mapviz to show mapping results in the real world.

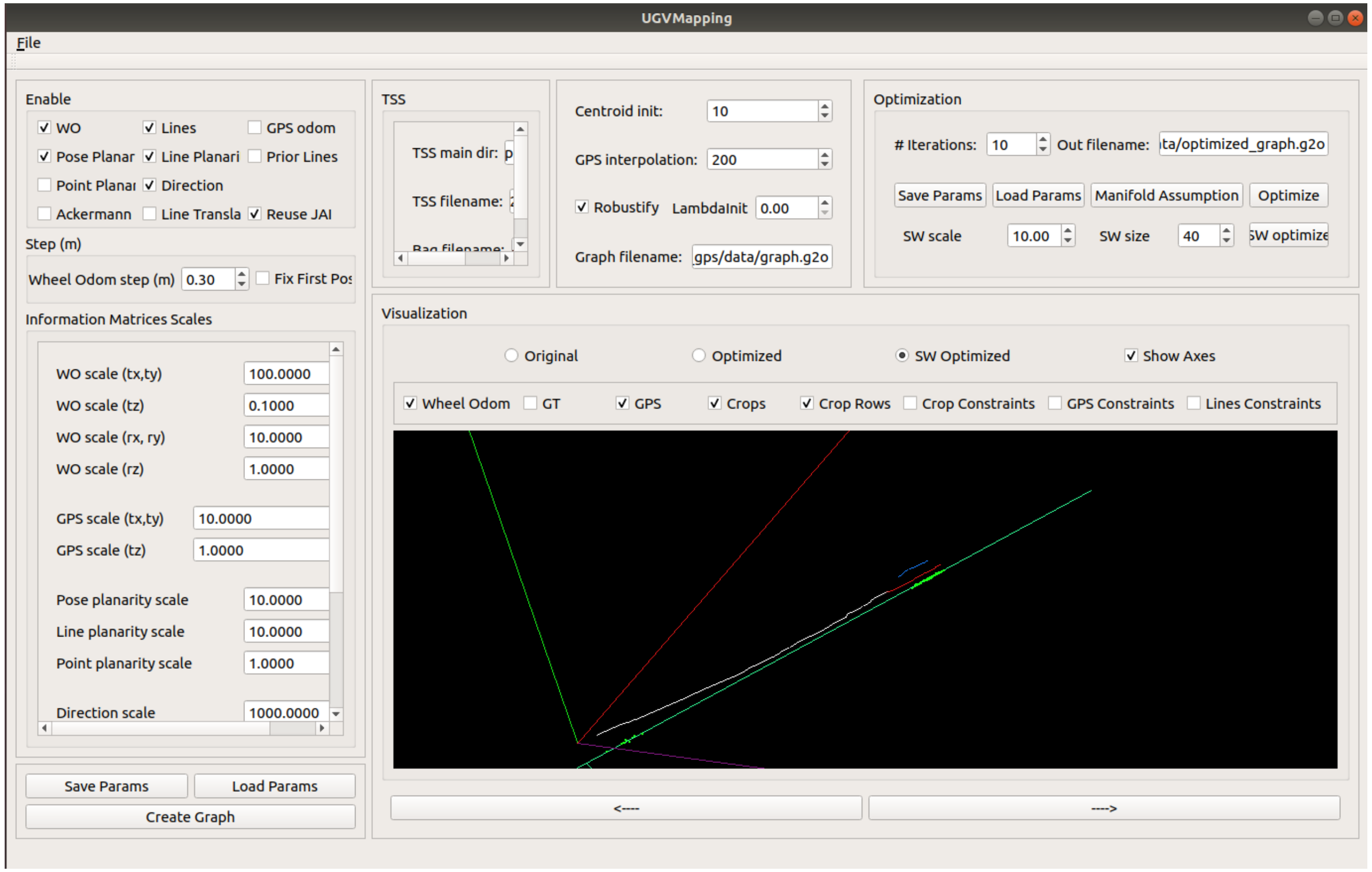
5.9.1 Integrated Visualization

The integrated visualization can be quite customized. It represents data in a 3D space from a given viewpoint, which can be rotated and translated. Three classes of data can be pictured: original, offline optimized and online optimized data, as shown in Figure 5.10. The coordinates frame axes are centered on the first GPS measurement, which is taken as reference point in the algorithm as well. Visualization options can be enabled to show:

- Wheel Odometry data (WO),
- GPS measurements (Ublox),
- Ground truth trajectory (Leica),
- Landmarks position,
- Crop rows as lines (also the normal vector is shown for easier debugging),
- GPS constraints between robot poses and GPS measurements,
- Line constraints between landmarks and crop row,
- Robot-Landmarks constraints (this is a fixed link with respect to the robot pose).

5.10 MAPPING

To have a complete Localization and Mapping algorithm, not only the robot localization, but also the environment mapping must be computed. As we have already highlighted, agriculture does not provide any other reference point rather than individual crop plants or crop rows because of the uniform visual appearance of the environment. In this case, we chose a landmark map, made of crops SEP, because from them we can infer time-invariant landmarks which can be reused over time. Crops positions are stored in a ROS bag, and can be visualized in the real world by using the Mapviz ROS package.



Enable
✓ WO ✓ Lines ☐ GPS odom
✓ Pose Planar ✓ Line Planari ☐ Prior Lines
☐ Point Planar ✓ Direction
☐ Ackermann ☐ Line Transla ✓ Reuse JAI

Step (m)
Wheel Odom step (m) 0.30 ☐ Fix First Pos

Information Matrices Scales

WO scale (tx,ty)	100.0000
WO scale (tz)	0.1000
WO scale (rx, ry)	10.0000
WO scale (rz)	1.0000
GPS scale (tx,ty)	10.0000
GPS scale (tz)	1.0000
Pose planarity scale	10.0000
Line planarity scale	10.0000
Point planarity scale	1.0000
Direction scale	1000.0000

TSS
TSS main dir: p
TSS filename:
Bag filename:

Centroid init: 10
GPS interpolation: 200
✓ Robustify LambdaInit 0.00
Graph filename: gps/data/graph.g2o

Optimization
Iterations: 10 Out filename: ta/optimized_graph.g2o
SW scale 10.00 SW size 40

Visualization
○ Original ○ Optimized ● SW Optimized ✓ Show Axes
✓ Wheel Odom ☐ GT ✓ GPS ✓ Crops ✓ Crop Rows ☐ Crop Constraints ☐ GPS Constraints ☐ Lines Constraints

Figure 5.10: Graphical Interface of the application. On the left side we have the list of constraints to be enabled or disabled and the list of scaling factors for the information matrices of all constraints. Then, on the upper part of the interface we have the file names and paths for input and output files used by the application, the number of centroids to initialize a crop row, the number of GPS measurements to initialize the orientation of the GPS relative frame and some parameters taken from the graph optimization framework. In particular, we can set the number of iterations the optimization performs, the size and the step of the sliding window for online optimization and enabling the edge robust least squares option. Below, we have the visualization of the robot trajectory and eventually of sensor measurements and constraints. In the example above, we can see the online optimization of the robot trajectory in white (current sliding window optimization is in red), the GPS trajectory corresponding to the current sliding window in blue, the crop row in green and the landmarks of the current sliding window overlapped to the line. The axes of the world coordinates frame are in red, violet and green.

If we play the ROS bag publishing optimized trajectory on the /wheel_odometry topic and landmarks map on the /crops topic, Mapviz subscribes to each topic and allows the user to visualize data at the exact streamed coordinates. Both topics must be recorded in the sensor_msgs/NavSatFix format in order to have a precise location on Earth. Mapviz allows also to set a very detailed background picture tile. We chose a detailed satellite image including the agricultural field where the dataset was recorded. Of course, the image cannot be zoomed up to see every single crop plant, but it is possible to see the crop rows and have a qualitative measure of the achieved accuracy.

5.11 ERROR COMPUTATION

In order to quantitatively asses the achieved results, we have developed an error computation module. It compares the optimized robot trajectory with the ground truth provided by the Leica GPS measurement. Basically, we have three different computation methods, evaluating the root mean squared error (RMSE).

5.11.1 Basic Error Computation

The basic error computation method was inherited by the initial project [16]. Since we have one data array per sensor type, and they can be matched based on the sequence number, which is conceptually related to the time when the measurement was produced, we can match each robot pose in the optimized trajectory to a ground truth position. Theoretically, it is not entirely correct if the two measurements have not the same exact timestamp, but can be considered a good approximation. Therefore, this method consists in iterating over the optimized robot poses, and for each one of them taking the square distance from the ground truth position. Square distances are summed and averaged by the number of robot poses, leading to the RMSE.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{seq=1}^{N} \|x_{gt,seq} - x_{robot,seq}\|^2}, \tag{5.22}$$

where $x_{gt,seq}$ is the ground truth position and $x_{robot,seq}$ is the robot position at sequence number seq.

5.11.2 Error Computation from Interpolation

In order to achieve a more accurate result, it is possible to use the error computed between the robot pose on the optimized trajectory at the exact timestamp of the corresponding ground truth position. To do so, we first find the two closest robot poses to the reference timestamp, then linearly interpolate the pose between the two. RMSE can now be computed matching poses at the exact timestamps.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N} \|x_{gt,t} - x_{robot,t}\|^2}. \tag{5.23}$$

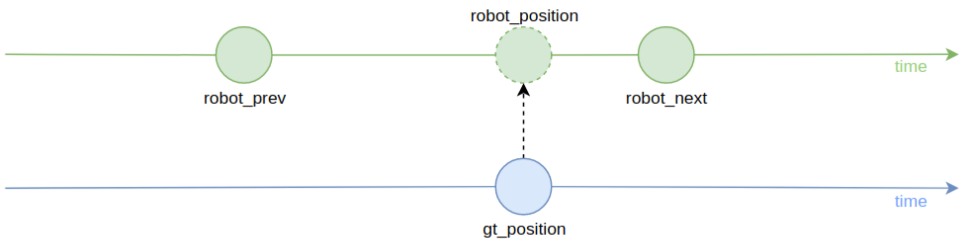


Figure 5.11: How to interpolate robot trajectory to compute a more accurate error.

Interpolation can be visualized in Figure 5.11.

5.11.3 Error Computation from Roto-Translation

Finally, we have developed another error computation method, which is applied when no GPS is used in the trajectory estimation. In this case, no absolute information is used and cannot be compared to the accuracy of the system based on the absolute comparison with ground truth only. Indeed, the goal of this thesis was to investigate a SLAM algorithm which could be applied without necessarily having GPS measurements. This is done by neglecting the error due to the roto-translation needed to align the optimized robot trajectory to the ground truth. We use the Umeyama function [71] which finds the best roto-translation transformation to align two sets of points. We use it to compute the transformation between the sets $\{x_{robot,1}, x_{robot,\frac{N}{2}}, x_{robot,N}\}$ and $\{x_{gt,1}, x_{gt,\frac{N}{2}}, x_{gt,N}\}$ and apply it to every robot pose in the optimized trajectory. Resulting trajectory can be compared to ground truth with one of the previously described methods: basic error computation or error computation from interpolation.

6 EXPERIMENTAL RESULTS AND EVALUATION

In this chapter, we present the results obtained by our application both for the Centroids Identification and for the SLAM components.

6.1 CENTROIDS IDENTIFICATION

The Centroids Identification component was trained in two phases. First we trained the neural network for image segmentation in order to have a system able to identify sugar beets (crop plants) in the images. The network output consisted of three classes: crops, weed and background. We trained the network on a training set of 903 images, and a validation set of 226 images. The final validation loss, computed with the weighted cross entropy function proposed in Chapter 5, was:

$$weighted\_crossentropy = 0.0340 \tag{6.1}$$

After the neural network was considered successfully trained, we applied some filtering on the output to remove noise and get a better classification of the pixels belonging to the crops class. We fed the ouptut to the clustering algorithm, however, we could not directly validate the clustering algorithm because we did not have a ground truth for the object detection of individual crop plants. Therefore, we evaluated the results in terms of the number of missed centroids and the average distance from the ground truth centroid. The clustering algorithm was able to identify most of the clusters representing crops in the image.

At this point, it is worth to stress that the centroids ground truth did not distinguish between centroids of crop plants and weeds, while the specific purpose of our component was to identify only crops SEP to be used as landmarks in the field. We propose an evaluation against the ground truth, however we must keep in mind that the results cannot be evaluated this way only. Indeed, we used both the objective evaluation and the experimental observation of the results. We observed that our algorithm was quite good at identifying crops, and not at identifying weeds that were often classified as noise and discarded. This behaviour was due to the narrower shape of the weeds relative to the crop plants, which makes the segmented components much smaller and affected by background noise in segmentation results, and resulted in the clustering algorithm processing weeds pixels as noise. Therefore it can be stated with reasonable confidence that detected centroids belonged to crop plants, while most of the missed centroids belonged to the

Error in centroids detection				
Max distance [pixels]	Detected	Precision	Recall	Avg error [pixels]
50	316	0.486	0.181	28.131
100	461	0.690	0.264	42.422
200	734	0.889	0.420	83.002

Table 6.1: Results from centroids detection algorithm. Average error is the average of the euclidean distance for all detected centroids from the ground truth ones. It takes into account the detected centroids, which are the ones whose distance from ground truth is smaller than the maximum distance. Detected centroids are an absolute number, while precision and recall are in the range between 0 and 1.

weed plants, therefore representing a negligible error. However, the previous consideration is not valid in general, as it depends on the relative shapes of crop plants and weeds. We also note that the high number of missed centroids is due to the prevalence of weeds in some of the analyzed images. With these premises, the experimental evaluation was performed using a visualization algorithm pointing out detected centroids and ground truth ones, while summarizing quantitative results in terms of the number of missed centroids. In the following, the quantitative and qualitative evaluation is presented.

Table 6.1 summarizes the quantitative evaluation of the centroids detection component. We tested the algorithm on 275 images in three different settings of maximum distance. The maximum distance is the limit of the distance between the hypothesis centroid and its ground truth point to be considered a correct detection. The difference between the three settings was that a maximum distance of $50 pixels$ was allowed in the first case, $100 pixels$ in the second case and $200 pixels$ in the third case. Considering that the average number of crop plants per image is between one and two, the number of detected centroids in the three settings confirms the hypothesis on the nature of the detected centroids belonging mostly to crop plants. We reported the average error for the detected centroids as the Euclidean distance measured in pixels from the ground truth point. By knowing that one pixel corresponds to about $0.6mm$ resolution, we can compute the corresponding error in meters, which is below $5cm$ in the worst considered case. Precision and recall are computed as:

$$Precision = \frac{TP}{TP + FP} Recall = \frac{TP}{TP + FN} \tag{6.2}$$

where $TP = TruePositive$, $FP = FalsePositive$, $FN = FalseNegatives$. Precision is to measure exactness and recall is to measure completeness. They both get lower as the maximum distance is lower. We note that precision can be easily achieved by relaxing the distance threshold, while recall remains lower because of the number of not identified centroids which we suppose to belong to weeds.

In the following, we discuss some qualitative results of the clustering and centroids computation algorithm. From Figures 6.1 to 6.7, we propose some examples to show how the algorithm works. For each example, from left to right, we have the original RGB image (brightness has been adjusted for allowing the reader an easier decoding), the smoothed segmentation result (where only crops are visible in blue/violet), the clustered crop plants (different colors identify clusters) and the identified centroids (white dots). We provide some details for each result under each image. Again we note that the algorithm is able to correctly cluster the pixels belonging to the crop plants even if they are not connected because the segmentation result is not completely precise.

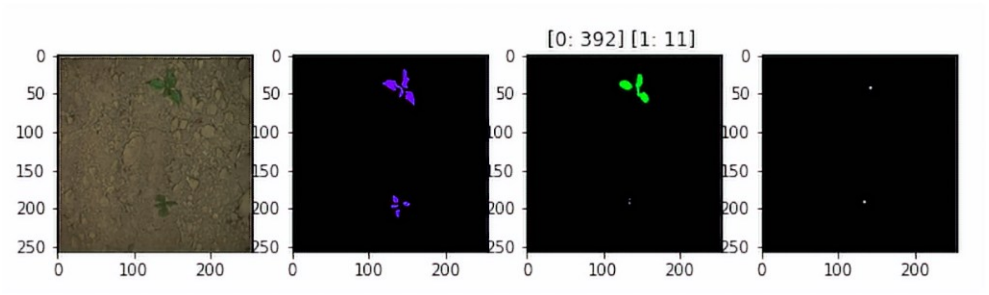


Figure 6.1: The image contains two crops which are correctly identified by both the segmentation and the clustering algorithms. We note that the clustering algorithm is able to identify even the lower crop which is extremely small and to output the correct centroid.

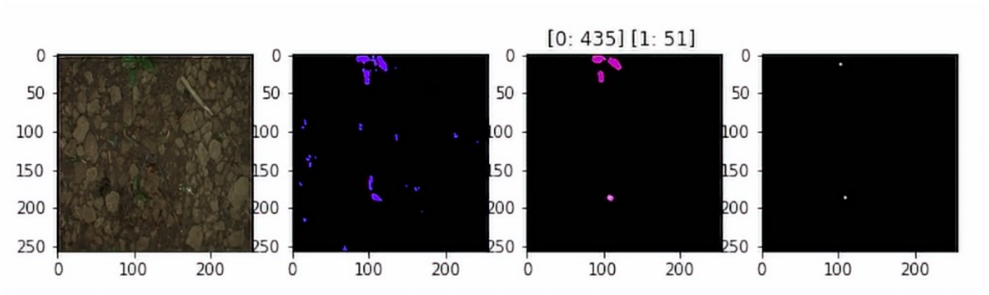


Figure 6.2: An example of correctly identified centroids even in the presence of weeds which are correctly discarded as noise by the clustering algorithm, while the small crop is identified.

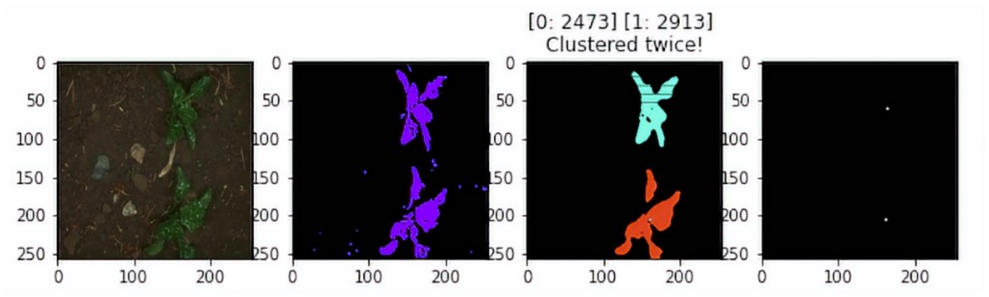


Figure 6.3: This image contains two big crops where leaves are almost touching. To correctly identify the two individual crops the algorithm was run twice as described in Chapter 5 and the two crops were eventually separated leading to the result shown in the last picture.

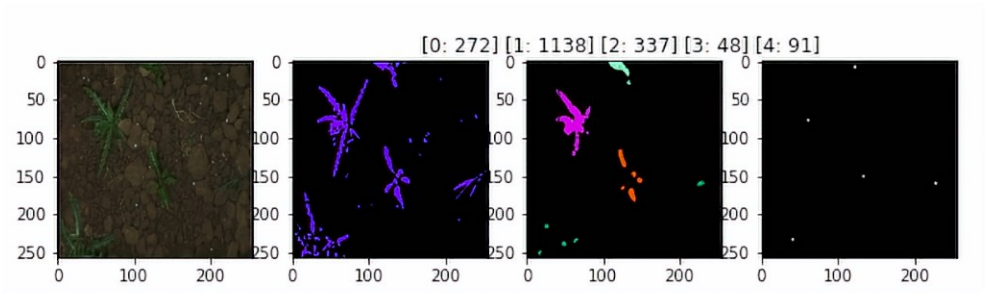


Figure 6.4: This is an example of multiple plants identified in a single image. It is not actually a good result because the algorithm does not manage to discard the pixels belonging to crops and includes weed centroids in the results. However, we note that the error is caused by the segmentation algorithm which did not correctly segment the image.

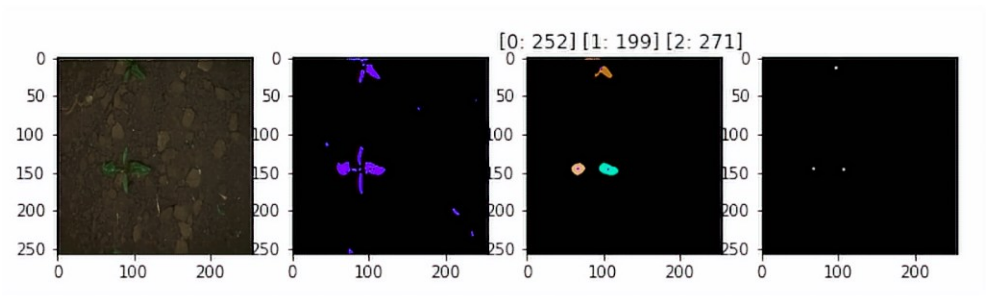


Figure 6.5: This image shows an error in the clustering algorithms, due to parameters setting. The second crop is incorrectly clustered as two independent crops since distance between leaves is too large with respect to the size of the plant.

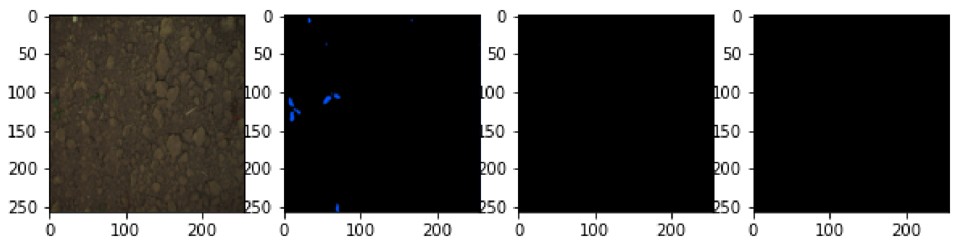


Figure 6.6: This example shows a clustering miss. The crops do not have at least min_samples to be identified as an independent cluster and get discarded as noise. Therefore we do not have the corresponding centroids in the result.

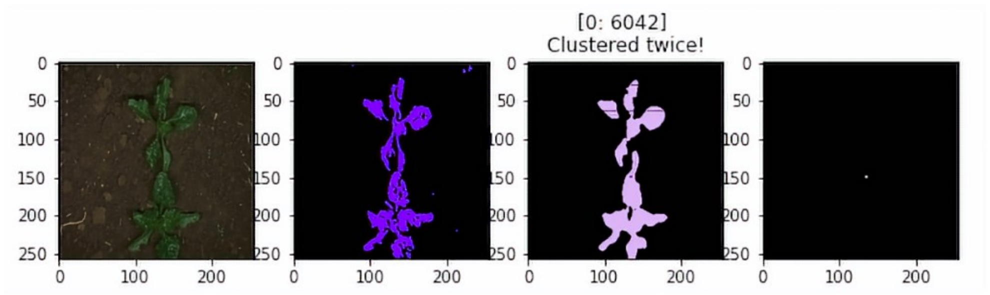


Figure 6.7: This is an example of two fully grown crops where leaves are touching and almost overlapping. The algorithm does not manage to cluster the crops independently, however we can note that the computed centroid is in line with the true ones (which can be inferred by looking at the image) so for the sake of the SLAM algorithm the error is not too heavy.

6.2 SYMULTANEOUS LOCALIZATION AND MAPPING

In this section, we provide some evaluation results for the SLAM component of the system, both online and offline. We start by including some statistics to describe the dataset we used. The tests were performed on four ROS bags from the dataset in Chapter 4, plus a fifth bag that is obtained by merging the first two considered bags. Below, we report the complete list of the bags that we used in the experiments, and that can be used for reproducing the same results:

- bag1: bonirob_2016-05-23-10-52-28_3.bag,
- bag2: bonirob_2016-05-23-10-57-33_4.bag,
- bag3: bonirob_2016-05-23-11-02-39_5.bag,
- bag4: bonirob_2016-05-23-11-07-45_6.bag,
- merged_bags: bag1 + bag2 (sequentially merged).

In order to assess the obtained results, we computed the RMSE error for each bag with the basic approach described in Chapter 5, considering as a first guess of the robot trajectory the sequence of robot poses computed by the wheel odometry. The errors are reported in Table 6.2.

6.2.1 Offline optimization

Since the application developed in this thesis allows to enable various constraints for the trajectory optimization, we performed a series of tests in which we alternately enabled or disabled the constraints to make a comparative analysis. Table 6.3 illustrates the testing cases with the enabled constraints for the offline optimization of the robot trajectory. The results are shown in Table 6.4.

First of all, we have to clarify the way of evaluating different test cases. The cases from a to d were assessed by computing the error from interpolation, while the cases from e to h were assessed by computing the error from roto-translation to align the optimized trajectory to the ground truth trajectory, as described in Chapter 5. The difference in error calculation is due to the lack of any absolute information in the last four test cases, which would make the direct comparison with ground truth extremely unfair and the

	bag1	bag2	bag3	bag4	merged_bags
RMSE	16.0176m	13.0196m	18.0933m	17.9625m	16.4874m

Table 6.2: Initial RMSE for bags used for testing, in meters.

	lines	prior_lines	gps_odom	direction	line_planarity	pose_planarity	ackermann	along_line_translation
a	1	1	1	0	0	1	0	0
b	1	1	1	0	0	1	1	0
c	1	1	1	0	0	0	0	0
d	1	1	1	0	0	0	1	0
e	1	0	0	1	1	1	0	1
f	1	0	0	0	0	0	0	0
g	1	0	0	1	1	1	0	0
h	1	0	0	1	1	0	0	1

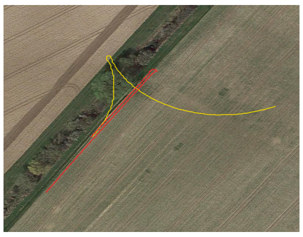
Table 6.3: Offline optimization settings: for each case we have the list of enabled (1) and disabled (0) constraints. Cases from a to d are absolutely oriented and can be directly compared to ground truth trajectory. Cases from e to h must be roto-translated before error computation, since they do not include GPS measurements among the enabled constraints.

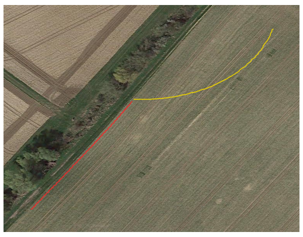
	a	b	c	d	e	f	g	h
bag1	0.72404	0.733592	0.826417	0.902248	2.07743	1.16676	2.00094	2.0777
					2.1355	0.277534	0.882197	0.231654
bag2	0.605515	0.653883	0.537156	0.558119	5.9235	1.70137	4.88119	5.92392
bag3	1.33371	1.46418	1.05927	1.08636	5.61197	1.60688	4.93594	5.61206
bag4	0.937765	0.824105	0.79978	0.709764	7.19206	2.23313	5.52116	7.19243
merged_bags	1.24495	1.27235	1.2409	1.27517	2.07683	1.16676	2.13703	2.0771
					2.78669	18.021	11.9879	2.78137

Table 6.4: Offline optimization results for each setting. For each test case and test bag we show the RMSE in meters. The relative error computation for the second group of test cases processes the trajectory along single crop rows, therefore datasets along two crop rows have to lines in the table.

	bag1	bag2	bag3	bag4	merged_bags
original	81.122	87.535	90.393	90.952	168.672
optimized	80.276	87.364	114.462	109.035	195.579

Table 6.5: Trajectory statistics from ROS evo_traj package [85]. For each bag, we provide the trajectory length in meters.


(a) Mapviz visualization of original merged_bags (bag1 and bag2).


(b) Mapviz visualization of original bag3.

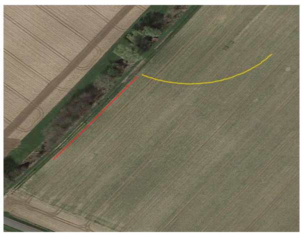

(c) Mapviz visualization of original bag4.

Figure 6.8: For each bag, we have in red the trajectory given by the ground truth (Leica sensor measurement), and in yellow the initial trajectory made of the concatenation of successive robot poses as measured by the wheel odometry.

relative advancements impossible to evaluate. Obviously, we cannot compare all the obtained results, but we can still perform a comparative analysis for the first group and for the second group independently. Also, we note that in the second group of test cases bag1 and merged_bags have two rows because the bag data correspond to the trajectory along two lines and the error computation algorithm computes the roto-translation for the trajectory along single lines and then compares the obtained trajectory with the ground truth.

Now we give a brief overview of the possible constraints, which also applies to the online optimization test settings. prior_lines is a constraint on crop rows built starting from the GPS measurements, while gps_odom is the direct constraint on the pose translation given by the GPS measurement. direction is a constraint on the direction of crop rows, which are assumed to be parallel, line_planarity is a constraint on the plane on which lines lie, and pose_planarity is a constraint on the plane on which robot poses lie. ackermann constraint computes the roto-translation between successive poses by applying the Ackermann motion model, and along_line_translation is a constraint, only valid offline, which computes the rectified trajectory from the first pose after a turn as if the robot travelled straight along the line.

From the results, we note that the developed algorithm performs very well in the first group of test cases, with RMSE always being less than 2 meters and almost in all cases less than 1 meter. We report the length of the trajectory for each bag in Table 6.5, from which we can estimate that the error is about 1% or 2% of the total trajectory length. As a general trend we can say that test case a and c perform better than b and d, so we can safely discard the ackermann constraint. However, it should be noted that the weights

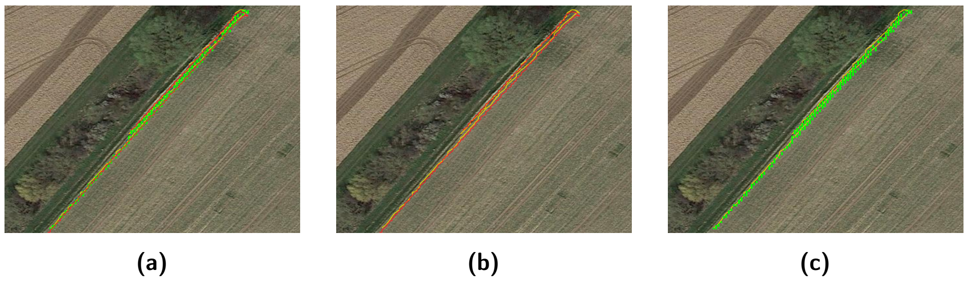


Figure 6.9: Mapviz visualization of optimized merged_bags (bag1 + bag2) in test case a. a shows the ground truth in red, the optimized robot trajectory in yellow and the landmarks (crops) in green. b shows only the ground truth and the optimized trajectories and c shows the trajectory with landmarks.

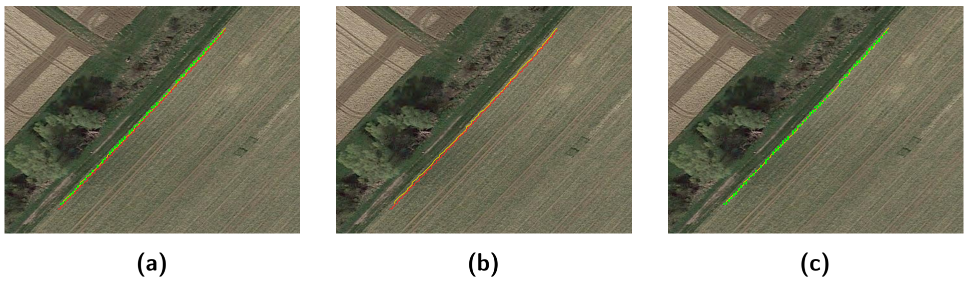


Figure 6.10: Mapviz visualization of optimized bag3 in test case a. a shows the ground truth in red, the optimized robot trajectory in yellow and the landmarks (crops) in green. b shows only the ground truth and the optimized trajectories and c shows the trajectory with landmarks.

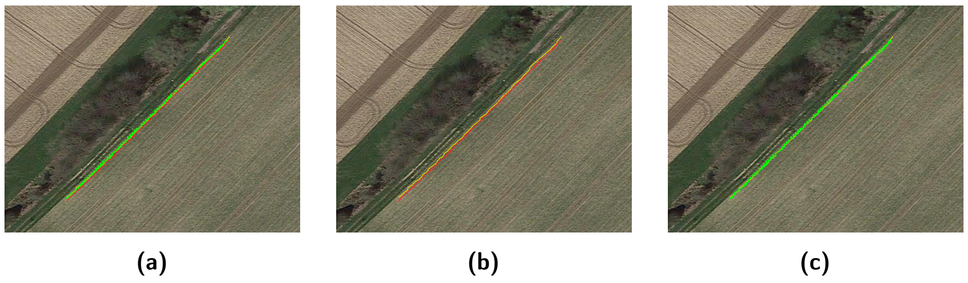


Figure 6.11: Mapviz visualization of optimized bag4 in test case c. a shows the ground truth in red, the optimized robot trajectory in yellow and the landmarks (crops) in green. b shows only the ground truth and the optimized trajectories and c shows the trajectory with landmarks.

of the information matrices were tuned on the first bag and suitable fine tuning might reduce the error in some test cases. line_planarity was never enabled because the prior_lines already intrinsically include the planarity constraint. pose_planarity did not improve the result (enabled in case a and disabled in case c) for most of the dataset, since the ground truth is

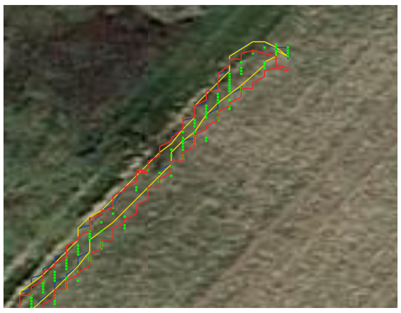
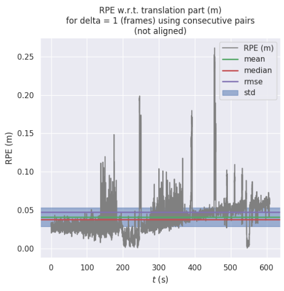


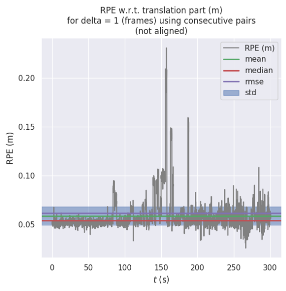
Figure 6.12: Zoomed mapviz visualization of the optimized trajectory in merged_bags on the robot turn, where it also changes the tracked crop row. We observe the optimized trajectory in yellow, the ground truth trajectory in red, the GPS measurement in blue and the landmarks as green dots.

not entirely planar and we did not have any reliable information on the z-axis, since the GPS measurement on altitude has an error up to 2 meters. The *pose_planarity* constraint should be explored in a setting where the altitude is fixed or known with a prior map. As an example, in Figures 6.9, 6.10, 6.11 we show the optimized robot trajectories for bags merged_bags, bag3 and bag4, respectively. All optimization examples are taken from the absolute test cases, since the to trajectory is absolutely oriented and can be directly compared with ground truth. The visualization precision is slightly rough, especially for the landmarks position, due to a numerical issue when transforming relative data to the absolute frame for latitude and longitude representation. Figure 6.8 shows the original trajectory as initialized by the wheel odometry estimates and transformed to absolute frame. The images give us a qualitative measure of the results achieved by the application and represent a coarse mapping of the environment. We also show a zoomed picture (Figure 6.12) of the robot turn in bag1 and merged_bags which illustrates how the algorithm controls the change of direction. In the proposed example, the algorithm identifies the turning point once the yaw change of the robot attitude is greater than a given threshold, and initializes a new crop row to be tracked.

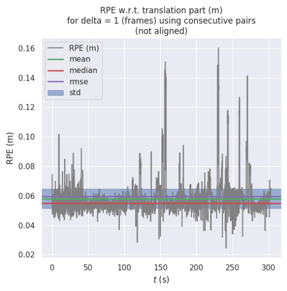
As for the non-absolute test cases, the *direction* constraint did not have much success. The first observation is that the *along_line_translation* constraint negatively affects the estimation in the case of single crop row datasets (bag1 contains the trajectory along one crop row and a very small portion of trajectory on next crop row) and highly improve the results in the case of the merged cases (two crop rows). On average, when *along_line_translation* is disabled, the algorithm performs better on test cases with planarity constraints disabled as well. When it is enabled, it performs better with *pose_planarity* constraint enabled.



(a)



(b)



(c)

Figure 6.13: Statistics on the error on test bags obtained by evo_rpe [85].

We include some statistics on the error computed on merged_bags in Figure 6.13a, on bag3 in Figure 6.13b, and on bag4 in Figure 6.13c for test case a. Statistics include RPE, which is the relative pose error, mean and standard deviation, and RMSE. RMSE is the same error we computed with the developed application, however the statistics over time show how the error varies on the computed trajectory.

6.2.2 Online optimization

In this subsection, we discuss the results obtained by the online optimization of the robot trajectory. Even if the component is designed to perform the optimization online, we could not optimize the code to actually run real-time. Therefore, we halved the rate of the data stream with respect to the rate at which measurements were registered. As for the offline optimization, we performed experiments on a set of test cases enabling or disabling the various constraints of the application, that can be divided into absolute, a to d, and non-absolute, e to h, test cases. The constraints are the same that we explained for the offline optimization settings, the test cases are summarized in Table 6.6 and the results are shown in Table 6.7. The same observations on error computation methodology hold as well.

Online optimization achieves slightly worse results than offline optimization, as data included in each optimization are a smaller amount. For the group of absolute test cases, error is included between 1% and 3.5% of the total path length of the corresponding bag. We note that on average case a is the one performing the best or still relatively good, which means that $line\_planarity$ and $pose\_planarity$ constraints positively influence the optimization. On the contrary, $ackermann$ constraint in general affects negatively the optimization. The gps_odom constraint has a different impact on the results based on the co-enabled constraints in each test case. In particular, we note that on the first two bags, it works quite well, while on the last two bags it does affect negatively the results. The reason of the described behaviour

	lines	prior_lines	gps_odom	direction	line_planarity	pose_planarity	ackermann	along_line_translation
a	1	1	0	1	1	1	0	0
b	1	1	1	1	1	1	0	0
c	1	1	0	0	0	0	0	0
d	1	1	0	1	1	1	1	0
e	1	0	0	1	1	1	0	0
f	1	0	0	0	0	0	0	0
g	1	0	0	1	1	0	0	0
h	1	0	0	1	0	1	0	0

Table 6.6: Online optimization settings: for each case we have the list of enabled (1) and disabled (0) constraints. Cases from a to d are absolutely oriented and can be directly compared to ground truth trajectory. Cases from e to h must be roto-translated before error computation, since they do not include GPS measurements among the enabled constraints.

	a	b	c	d	e	f	g	h
bag1	1.46967	1.27404	1.42826	1.28221	0.919922	1.22728	1.10818	1.17306
					0.309403	0.394355	0.233279	0.567267
bag2	1.72429	1.20716	2.39444	3.05694	1.58775	1.50914	1.37933	3.28948
bag3	2.55806	4.83699	2.42266	3.04434	13.9128	1.84828	23.6233	9.80207
bag4	2.57354	4.60034	2.78905	2.83024	2.19596	1.80023	1.77261	2.18003
merged_bags	3.35566	3.07285	3.03656	3.50627	-	-	-	-

Table 6.7: Online optimization results for each setting. For each test case and test bag we show the RMSE in meters. The relative error computation for the second group of test cases processes the trajectory along single crop rows, therefore datasets along two crop rows have to lines in the table.

	lines	prior_lines	gps_odom	pose_planarity	point_planarity
case 1	1	1	1	1	1
case 2	1	1	1	1	0

Table 6.8: Test cases performed for each GPS sensor on the dataset provided by Sapienza University.

is the high error in the GPS measurements on the z-axis with respect to the ground truth in bag3 and bag4.

The non-absolute test cases group shows some very bad results for bag3, because the trajectory starts oscillating on z-axis, when no altitude information is provided. Excluding this bag, which might be considered as an outlier and needs fine-tuning for weighting better the constraints, the best performing average results are on the test case g. This means that the pose_planarity constraint negatively affects the results overall. The line_planarity constraint, instead, slightly improves the optimization precision as case g (enabled) is on average performing better than f and h, where it is disabled.

6.2.3 Comparison with the state of the art

The project in [16] was applied to a different dataset [86] from the one we used for training. To make our discussion complete, in the following we discuss the results on the same dataset. In particular, we selected the DatasetA_100_200.bag and run a series of tests with the two provided GPS sensors. Table 6.8 shows the test cases where the enabled constraints are the same as for previous test cases, plus the point_planarity which constraints the SEP to lie on the desired plane. The test cases were repeated for the two GPS sensors: an RTK GPS and a PPP GPS. We report both the error statistics and the 3D map with ground truth and optimized robot trajectory.

The obtained results were fairly good for the trajectory optimization, while the mapping did not work as expected. The developed algorithm, indeed, monitors the yaw delta in wheel odometry attitude estimation to detect a turn in the robot trajectory. In the new tested dataset the wheel odometry estimation is extremely inaccurate, so the threshold for identifying a curve is never passed. The best way to sense for a robot turn in this case would be to fuse the IMU readings in the estimation, which are included in this dataset.

Figure 6.14a and 6.14b show optimized trajectory for test case 1 using the RTK and PPP GPS sensor, respectively. Errors are reported in Table 6.9 and can be directly compared with the errors reported in [16], even if our results are relative to one bag only. We note, that for the PPP test cases, error is lower than the corresponding one in the state of the art. This means that the crop row constraints positively affect the results.

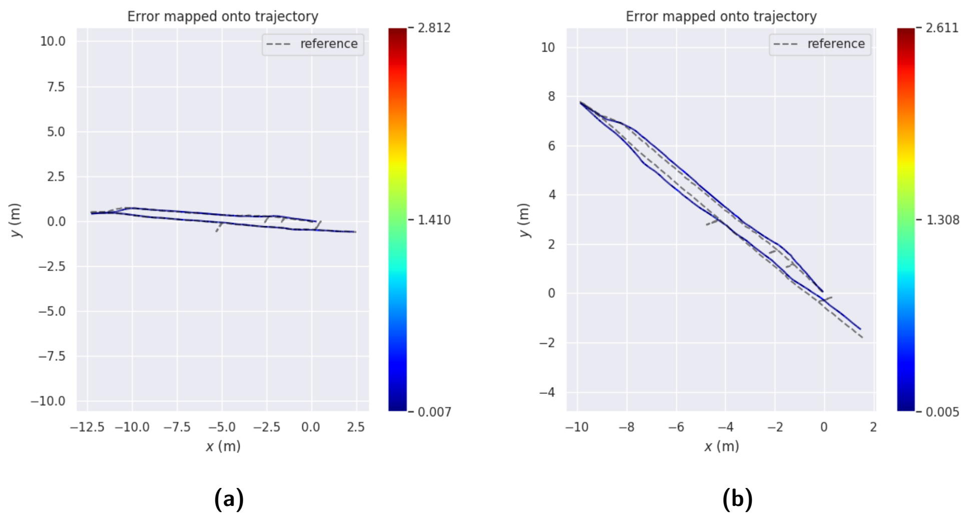


Figure 6.14: Optimized trajectories of Sapienza dataset, visualized by [85], on first test case for RTK GPS (a) and for PPP GPS (b).

	Current application				State of the art	
	RTK		PPP		RTK	PPP
	test 1	test 2	test 1	test 2	best	best
RMSE	0.138506	0.138437	0.296376	0.296737s	0.075	0.401

Table 6.9: Errors of test cases on dataset provided by Sapienza University, in meters, compared with the best results achieved by the state of the art. However, the state of the art results are computed on all merged bags of the dataset, while we only tested one.

7 CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

This thesis was developed with the goal of reducing the dependence of robot localization systems on the expensive RTK-GPS sensor. Agricultural localization algorithms mostly rely on this sensor as it is highly precise, however it requires the availability of a second base station close to the field of operation and is quite expensive. Therefore, we explored possible localization methods that did not entirely rely on GPS. Another motivation is the lack of prior maps in the agricultural environment, therefore agricultural robots need to produce a map of the environment simultaneously while they are localizing themselves.

By investigating the literature, we discovered that methods performing Simultaneous Localization And Mapping (SLAM) in agricultural fields commonly exploit two kinds of landmark recognition methods: they can either track the entire crop row or detect single plants on the ground. Both methods usually exploit a vision or a laser range sensor. We also found proof that landmarks in the agricultural environment are difficult to detect because it is visually homogeneous and repetitive. Since vegetation changes appearance due to plants growth, it is difficult to track reference points in the agricultural environment. Therefore, we investigated the method suggested by one of the analysed papers to identify time-invariant landmarks in the agricultural field by extracting the Stem Emerging Point (SEP) from each crop plant.

We developed an application to identify landmarks in the agricultural field and to use them in a SLAM algorithm. The first component includes a convolutional neural network for image segmentation and a clustering module to identify individual crop plants in images taken by the robot. Then, landmarks consist in the crop plants SEP, computed as the clusters center of mass. Landmarks can be used by the SLAM algorithm as reference points for the localization system and to produce a map of the observed crop plants along the trajectory.

The core component of the application performs 3D localization and mapping for an autonomous robot in an agricultural field. It fuses wheel odometry estimate, GPS readings and landmarks observations into a graph-based optimization framework. Localization and mapping can be performed both online and offline. A graphical interface was adapted to enable and disable different constraints in the optimization. The main contribution given by our application is the study of crop rows constraints in the trajectory optimization. Crop rows are represented by 3D Plücker lines interpolating plant SEPs observed by the robot.

The performance of the developed system was assessed by computing the RMSE relative to the ground truth trajectory. The results showed that the algorithm achieves a performance aligned with the state of the art, and it is even better when using a consumer grade GPS sensor. However, we tested the algorithm on a limited set of data and the robot turn detection method proved not to be robust when the yaw estimate provided by the wheel odometry is not reliable. This also affects the performance of the environment mapping, which is completely dependent on the identification of crop rows, which in turn are detected only when a robot turn is successfully identified.

7.2 FUTURE WORKS

In this section we propose some future works that may be performed in order to improve the results obtained in this thesis. The analysis we carried out in this work is aimed at improving SLAM estimation. However to make the SLAM algorithm complete, one more step should be performed: loop closure. Loop closure consists in recognizing the same landmark when already present in the map and optimizing the robot trajectory and the environment map considering that the robot already observed the same reference point. Thus, the trajectory 'closes' and another optimization is performed.

Moreover, we propose the development of some features that may improve the results.

- Improving and testing the analytic formulation of the Jacobians of the error for edges in g2o which should speed up the computations.
- Exploring alternative clustering methods to speed up the computations.
- Improving the SEP detection method, which currently relies on the domain assumption that weeds have narrower shape relative to crop plants and so they are discarded by the clustering algorithm. However, if the relative shapes are different from what we expect, the clustering algorithm is not guaranteed to identify the correct clusters.
- Integrating different sensor measurements in the SLAM optimization algorithm. An example could be the integration of IMU readings to better detect the curves in the trajectory and the change of crop rows.

In conclusion, we think that precision agriculture is gathering attention, but the full transition to Agriculture 4.0 is still far away to reach. Several studies on agricultural 3D localization and mapping exist, however, the complete solution will rely also on the future scientific and technological advancements.

We foresee that sensors will become more accurate and affordable and hardware will support faster computations, therefore the optimization algorithm will be able to take into consideration a larger number of constraints while still being real-time. The advancements will allow to address the issues which arise due to the specific environment and gradually solve many of those.

Agriculture will face critical challenges in the future, because of the population growth. It will be able to correspond the demands only when the problem will be addressed systematically. Farmers which will adopt technological innovation will be the ones able to satisfy the request, but way more investment is needed. To widespread the use of sophisticated technological tools it is necessary to make them more affordable. This can be achieved only by a great effort in the study of the technological systems supporting agriculture, whose results are entirely based on a precise localization and mapping component.

BIBLIOGRAPHY

[1] Feng Lu and Evangelos Milios. "Globally consistent range scan alignment for environment mapping." In: *Autonomous robots* 4.4 (1997), pp. 333–349 (cit. on p. 21).

[2] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. "A tutorial on graph-based SLAM." In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43 (cit. on pp. 21, 23).

[3] Achim Walter, Raghav Khanna, Philipp Lottes, Cyrill Stachniss, Roland Siegwart, Juan Nieto, and Frank Liebisch. "Flourish-a robotic approach for automation in crop management." In: *Proceedings of the international conference on precision agriculture (ICPA)*. 2018.

[4] Klas Nordberg. "Introduction to representations and estimation in geometry." In: (2018) (cit. on pp. 25, 26, 29, 32).

[5] *METRICS website*. `https://metricsproject.eu/about/` (cit. on p. 3).

[6] *METRICS Agri-Food Competition*. `https://metricsproject.eu/agri-food/` (cit. on p. 3).

[7] *ROS documentation*. `http://wiki.ros.org/Documentation` (cit. on p. 45).

[8] *Image Sensor Message*. `http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Image.html` (cit. on p. 47).

[9] *NavSatFix Sensor Message*. `http://docs.ros.org/melodic/api/sensor_msgs/html/msg/NavSatFix.html` (cit. on p. 47).

[10] *Odometry Navigation Message*. `http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html` (cit. on p. 46).

[11] *GeoPoint Geographic Message*. `http://docs.ros.org/en/jade/api/geographic_msgs/html/msg/GeoPoint.html` (cit. on p. 48).

[12] *Robot Localization Package*. `http://docs.ros.org/melodic/api/robot_localization/html/index.html`.

[13] *Policy-Based Synchronizer*. `http://wiki.ros.org/message_filters/ApproximateTimhttp://wiki.ros.org/message_filters#Policy-Based_Synchronizer_.5BROS_1.1.2B-.5D` (cit. on p. 48).

[14] *Approximate Time Filter*. `http://wiki.ros.org/message_filters/ApproximateTime` (cit. on p. 48).

[15] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. "g 2 o: A general framework for graph optimization." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613 (cit. on pp. 8, 33, 55).

[16] Marco Imperoli, Ciro Potena, Daniele Nardi, Giorgio Grisetti, and Alberto Pretto. "An effective multi-cue positioning system for agricultural robotics." In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3685–3692 (cit. on pp. 5, 8, 53, 84, 99).

[17] Florian Kraemer, Alexander Schaefer, Andreas Eitel, Johan Vertens, and Wolfram Burgard. "From plants to landmarks: Time-invariant plant localization that uses deep pose regression in agricultural fields." In: *arXiv preprint arXiv:1709.04751* (2017).

[18] Wera Winterhalter, Freya Veronika Fleckenstein, Christian Dornhege, and Wolfram Burgard. "Crop row detection on tiny plants with the pattern hough transform." In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3394–3401.

[19] Qingkuan Meng, Xiayi Hao, Yingmei Zhang, and Genghuang Yang. "Guidance line identification for agricultural mobile robot based on machine vision." In: *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE. 2018, pp. 1887–1893.

[20] Chunling Tu, Barend Jacobus Van Wyk, Karim Djouani, Yskandar Hamam, and Shengzhi Du. "An efficient crop row detection method for agriculture robots." In: *2014 7th International Congress on Image and Signal Processing*. IEEE. 2014, pp. 655–659.

[21] Nived Chebrolu, Philipp Lottes, Alexander Schaefer, Wera Winterhalter, Wolfram Burgard, and Cyrill Stachniss. "Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields." In: *The International Journal of Robotics Research* 36.10 (2017), pp. 1045–1052 (cit. on pp. 41–44).

[22] Jonathan Ragan-Kelley and Ren Ng. *Raytracing lecture*. `https://cs184.eecs.berkeley.edu/sp19/lecture/9-20/raytracing` (cit. on p. 27).

[23] Danping Zou. *Projective Geometry lecture*. `http://drone.sjtu.edu.cn/dpzou/teaching/course/lecture04-projective_geometry.pdf` (cit. on p. 30).

[24] Jorge J Moré. "The Levenberg-Marquardt algorithm: implementation and theory." In: *Numerical analysis*. Springer, 1978, pp. 105–116 (cit. on p. 35).

[25] Nived Chebrolu, Thomas Labe, Olga Vysotska, Jens Behley, and Cyrill Stachniss. "Adaptive robust kernels for non-linear least squares problems." In: *IEEE Robotics and Automation Letters* (2021) (cit. on p. 36).

[26] Benoît Thuilot, Christophe Cariou, Lionel Cordesses, and Philippe Martinet. "Automatic guidance of a farm tractor along curved paths, using a unique CP-DGPS." In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*. Vol. 2. IEEE. 2001, pp. 674–679 (cit. on pp. 5, 9).

[27] Albert Stoll and Heinz Dieter Kutzbach. "Guidance of a forage harvester with GPS." In: *Precision Agriculture* 2.3 (2000), pp. 281–291 (cit. on pp. 5, 9).

[28] Clément Fouque and Philippe Bonnifait. "On the use of 2D navigable maps for enhancing ground vehicle localization." In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 1885–1890.

[29] Henrik S Midtiby, Thomas M Giselsson, and Rasmus N Jørgensen. "Estimating the plant stem emerging points (PSEPs) of sugar beets at early growth stages." In: *Biosystems engineering* 111.1 (2012), pp. 83–90 (cit. on pp. 15, 18).

[30] Andrew English, Patrick Ross, David Ball, and Peter Corke. "Vision based guidance for robot navigation in agriculture." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1693–1698 (cit. on pp. 14, 15).

[31] Kumar Vishal, CV Jawahar, and Visesh Chari. "Accurate localization by fusing images and GPS signals." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 17–24 (cit. on pp. 5, 9).

[32] Vignesh Raja Ponnambalam, Jaime Pulido Fentanes, Gautham Das, Grzegorz Cielniak, Jon Glenn Omholt Gjevestad, and Pål Johan From. "Agri-Cost-Maps-Integration of Environmental Constraints into Navigation Systems for Agricultural Robots." In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE. 2020, pp. 214–220 (cit. on pp. 5, 9, 10).

[33] F Auat Cheein, G Steiner, G Perez Paina, and Ricardo Carelli. "Optimized EIF-SLAM algorithm for precision agriculture mapping based on stems detection." In: *Computers and electronics in agriculture* 78.2 (2011), pp. 195–207 (cit. on pp. 15, 18).

[34] Ulrich Weiss and Peter Biber. "Plant detection and mapping for agricultural robots using a 3D LIDAR sensor." In: *Robotics and autonomous systems* 59.5 (2011), pp. 265–273 (cit. on pp. 5, 10).

[35] Sebastian Haug, Peter Biber, Andreas Michaels, and Jörn Ostermann. "Plant stem detection and position estimation using machine vision." In: *Workshop Proc. of Conf. on Intelligent Autonomous Systems (IAS)*. 2014, pp. 483–490 (cit. on pp. 15, 18).

[36] David Schleicher, Luis M Bergasa, Manuel Ocaña, Rafael Barea, and María Elena López. "Real-time hierarchical outdoor SLAM based on stereovision and GPS fusion." In: *IEEE Transactions on Intelligent Transportation Systems* 10.3 (2009), pp. 440–452 (cit. on pp. 6, 11).

[37] Ignacio Parra, Miguel Angel Sotelo, David F Llorca, Carlos Fernández, A Llamazares, Noelia Hernández, and Iván García. "Visual odometry and map fusion for GPS navigation assistance." In: *2011 IEEE International Symposium on Industrial Electronics*. IEEE. 2011, pp. 832–837 (cit. on pp. 5, 9).

[38] Thiago Teixeira Santos, Luciano Vieira Koenigkan, Jayme Garcia Arnal Barbedo, and Gustavo Costa Rodrigues. "3D plant modeling: localization, mapping and segmentation for plant phenotyping using a single hand-held camera." In: *European Conference on Computer Vision*. Springer. 2014, pp. 247–263 (cit. on pp. 5, 10, 11).

[39] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. "6D SLAM—3D mapping outdoor environments." In: *Journal of Field Robotics* 24.8-9 (2007), pp. 699–722 (cit. on pp. 6, 11).

[40] Shangjie Shi, Wei Zhou, and Shuang Liu. "A high efficient 3D SLAM algorithm based on PCA." In: *2016 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2016, pp. 109–114 (cit. on pp. 6, 12).

[41] Shangjie Shi, Wei Zhou, and Shuang Liu. "An Efficient Multi-Robot 3D SLAM Algorithm." In: 2017, pp. 1200–1205 (cit. on pp. 6, 11).

[42] Ciro Potena, Daniele Nardi, and Alberto Pretto. "Fast and accurate crop and weed identification with summarized train sets for precision agriculture." In: *International Conference on Intelligent Autonomous Systems*. Springer. 2016, pp. 105–121 (cit. on pp. 16, 19).

[43] Ji Li and Lie Tang. "Crop recognition under weedy conditions based on 3D imaging for robotic weed control." In: *Journal of Field Robotics* 35.4 (2018), pp. 596–611 (cit. on pp. 16, 19).

[44] Jacqueline Libby and George Kantor. "Deployment of a point and line feature localization system for an outdoor agriculture vehicle." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1565–1570 (cit. on pp. 5, 10).

[45] Songmin Jia, Boyang Li, Guoliang Zhang, and Xiuzhi Li. "Improved KinectFusion based on graph-based optimization and large loop model." In: *2016 IEEE international conference on information and automation (ICIA)*. IEEE. 2016, pp. 813–818 (cit. on pp. 6, 12).

[46] João Carlos Virgolino Soares and Marco Antonio Meggiolaro. "Keyframe-based rgb-d slam for mobile robots with visual odometry in indoor environments using graph optimization." In: *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. IEEE. 2018, pp. 94–99 (cit. on pp. 6, 12).

[47] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. "Visual odometry and mapping for autonomous flight using an RGB-D camera." In: *Robotics Research*. Springer, 2017, pp. 235–252 (cit. on p. 12).

[48] Motilal Agrawal, Kurt Konolige, and Robert C Bolles. "Localization and mapping for autonomous navigation in outdoor terrains: A stereo vision approach." In: *2007 IEEE Workshop on Applications of Computer Vision (WACV'07)*. IEEE. 2007, pp. 7–7 (cit. on pp. 6, 11).

[49] Tuan Le, Jon Glenn Omholt Gjevestad, and Pål Johan From. "Online 3D Mapping and Localization System for Agricultural Robots." In: *IFAC-PapersOnLine* 52.30 (2019), pp. 167–172 (cit. on pp. 6, 12).

[50] Ashwin Vasudevan, D Ajith Kumar, and NS Bhuvaneswari. "Precision farming using unmanned aerial and ground vehicles." In: *2016 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*. IEEE. 2016, pp. 146–150.

[51] Xingxing Zuo, Xiaojia Xie, Yong Liu, and Guoquan Huang. "Robust visual SLAM with point and line features." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1775–1782 (cit. on pp. 7, 13).

[52] Fangwen Shu, Paul Lesur, Yaxu Xie, Alain Pagani, and Didier Stricker. "SLAM in the Field: An Evaluation of Monocular Mapping and Localization on Challenging Dynamic Agricultural Environment." In: *arXiv preprint arXiv:2011.01122* (2020) (cit. on pp. 6, 12).

[53] Chunling Tu, Barend Jacobus Van Wyk, Karim Djouani, Yskandar Hamam, and Shengzhi Du. "An efficient crop row detection method for agriculture robots." In: *2014 7th International Congress on Image and Signal Processing*. IEEE. 2014, pp. 655–659 (cit. on pp. 14, 15, 17).

[54] Li-Ying Zheng and Jing-Xue Xu. "Multi-crop-row detection based on strip analysis." In: *2014 International Conference on Machine Learning and Cybernetics*. Vol. 2. IEEE. 2014, pp. 611–614 (cit. on pp. 14, 15).

[55] R Aravind, M Daman, and BS Kariyappa. “Design and development of automatic weed detection and smart herbicide sprayer robot.” In: *2015 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*. IEEE. 2015, pp. 257–261 (cit. on pp. 16, 19).

[56] Mark A Post, Alessandro Bianco, and Xiu T Yan. “Autonomous navigation with ROS for a mobile robot in agricultural fields.” In: *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. 2017 (cit. on pp. 6, 12).

[57] Qingkuan Meng, Xiayi Hao, Yingmei Zhang, and Genghuang Yang. “Guidance line identification for agricultural mobile robot based on machine vision.” In: *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE. 2018, pp. 1887–1893 (cit. on pp. 15, 17).

[58] Jing Dong, John Gary Burnham, Byron Boots, Glen Rains, and Frank Dellaert. “4D crop monitoring: Spatio-temporal reconstruction for agriculture.” In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3878–3885 (cit. on pp. 7, 13).

[59] Philipp Lottes, Markus Hoeferlin, Slawomir Sander, M Müter, P Schulze, and Lammers C Stachniss. “An effective classification system for separating sugar beets and weeds for precision farming applications.” In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 5157–5163 (cit. on pp. 16, 19).

[60] Guoxuan Zhang, Jin Han Lee, Jongwoo Lim, and Il Hong Suh. “Building a 3-D line-based map using stereo SLAM.” In: *IEEE Transactions on Robotics* 31.6 (2015), pp. 1364–1377 (cit. on pp. 7, 13).

[61] Wera Winterhalter, Freya Veronika Fleckenstein, Christian Dornhege, and Wolfram Burgard. “Crop row detection on tiny plants with the pattern hough transform.” In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3394–3401 (cit. on pp. 15, 17).

[62] Florian Kraemer, Alexander Schaefer, Andreas Eitel, Johan Vertens, and Wolfram Burgard. “From plants to landmarks: Time-invariant plant localization that uses deep pose regression in agricultural fields.” In: *arXiv preprint arXiv:1709.04751* (2017) (cit. on pp. 15, 17, 18).

[63] BS Blackmore and S Choudhury. “Fuzzy Landmark Detection in Simultaneous Localisation and Mapping for Agricultural Robots.” In: () (cit. on pp. 16, 19).

[64] Andres Milioto, Philipp Lottes, and Cyrill Stachniss. “Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in CNNs.” In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 2229–2235 (cit. on pp. 16, 19, 57, 58).

[65] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163 (cit. on p. 11).

[66] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. "OpenVSLAM: a versatile visual slam framework." In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 2292–2295 (cit. on p. 12).

[67] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. "Pixelwise view selection for unstructured multi-view stereo." In: *European Conference on Computer Vision*. Springer. 2016, pp. 501–518 (cit. on p. 12).

[68] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation." In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2011 (cit. on p. 12).

[69] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation." In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446 (cit. on p. 12).

[70] David G Lowe. "Distinctive image features from scale-invariant keypoints." In: *International journal of computer vision* 60.2 (2004), pp. 91–110 (cit. on p. 13).

[71] Shinji Umeyama. "Least-squares estimation of transformation parameters between two point patterns." In: *IEEE Computer Architecture Letters* 13.04 (1991), pp. 376–380 (cit. on pp. 49, 85).

[72] *Eigen Library website*. https://eigen.tuxfamily.org/dox/GettingStarted.html (cit. on p. 49).

[73] *Pose graph library - merge bags*. https://bitbucket.org/cirpote/pose_graph/src/master/src/nodes/mergeSequentialBagsExample.cpp (cit. on p. 64).

[74] *Pose graph library - TSS tools*. https://bitbucket.org/cirpote/pose_graph/src/master/src/library/temporalgraph.cpp (cit. on p. 66).

[75] *Kaggle: a data science platform*. https://www.kaggle.com/ (cit. on p. 45).

[76] *DBSCAN - Density-Based Spatial Clustering of Applications with Noise*. https://scikit-learn.org/stable/modules/clustering.html#dbscan (cit. on pp. 40, 61).

[77] *scikit-learn: machine learning in python*. https://scikit-learn.org/stable/ (cit. on p. 40).

[78] *Smoothing images with opencv*. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html (cit. on p. 61).

[79] *ROS Mapviz*. http://wiki.ros.org/mapviz (cit. on p. 49).

[80] *ROS rqt*$_g$raph. http://wiki.ros.org/rqt_graph.

[81] *Tensorflow*. https://www.tensorflow.org/api_docs/python/tf (cit. on p. 50).

[82] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241 (cit. on pp. 38, 39, 57).

[83] *Qt OpenGL*. https://doc.qt.io/qt-5/qtgui-index.html (cit. on p. 82).

[84] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. "Design of an image edge detection filter using the Sobel operator." In: *IEEE Journal of solid-state circuits* 23.2 (1988), pp. 358–367 (cit. on p. 60).

[85] *Evo Trajectory*. https://github.com/MichaelGrupp/evo (cit. on pp. 94, 97, 100).

[86] *Mapping dataset Sapienza*. http://www.diag.uniroma1.it//~labrococo/fds/mappingdatasets.html (cit. on p. 99).