FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cost-effective robot for steep slope crops monitoring

Filipe Peixoto Mendes

Master in Electrical and Computers Engineering

Supervisor: Prof. Pedro Luís Cerqueira Gomes da Costa

Co-Supervisor: Prof. José Luis Sousa de Magalhães Lima

Co-Supervisor: Dr. Eng. Filipe Baptista Neves dos Santos

July 25, 2019

Resumo

A necessidade de eficiência no setor agrícola surge naturalmente de um progressivo aumento populacional e de crescente preocupação ambiental. Com este fim, existem esforços tanto da parte da investigação pública como da privada, focados no desenvolvimento de agricultura mais precisa e eficiente, onde recentes avanços tecnolóogicos permitem melhor gestão da atividade e uma utilização ótima de recursos. Apesar dos primeiros esforços no desenvolvimento de robôs para agricultura serem em tratores autónomos, com o preço reduzido dos veículos autónomos leves, especialmente *drones*, originou-se um aumento da atratividade deste tipo de veículos no novo paradigma de agricultura de precisão.

Os robôs terrestres agrícolas possuem a possiblidade de realizar uma multitude de tarefas no entanto, o seu desenvolvimento é particularmente desafiante devido a condições impostas pelo cenário agrícola, nomeadamente temperaturas elevadas, terrenos de forte inclinação, capacidade de comunicação limitada e estruturas não geométricas. Na região do Douro encontram-se vinhas instaladas ao longo das encostas, onde o grau de adoção de mecanização é baixo devido ao reduzido espaço para executar manobras e outras condições que tenham impacto na operação das máquinas.

O objetivo deste projecto é o desenvolvimento de uma plataforma robótica capaz de lidar com as dificuldades mencionadas. O robô utiliza um LIDAR 2D, um módulo GNSS, odometria e um IMU para realizar as tarefas de localização e mapeamento. Todos os sub-sistemas desenvolvidos, incluindo software, hardware e sistemas mecânicos são detalhados, juntamente com os procedimentos de teste e os resultados obtidos. De forma a reduzir o custo total, foi utilizada uma Raspberry Pi como computador principal. Também é conseguida a validação do algoritmo Cartographer SLAM em cenários diferentes e realizada uma análise de performance do algoritmo em componentes de baixo custo.

O resultado final é uma plataforma robótica open-source capaz de executar missões autonomamente, onde é possível instalar componentes adicionais. Foi conseguido uma solução robusta de localização com componentes de baixo custo, uma solução de navegação e uma interface de utilizador com integração da aplicação QGroundControl.

Abstract

An increasing population and the negative impact of agriculture on the environment drive the need for an agriculture more precise and efficient. Private and public research is emerging in the field of precision agriculture, where technological advancements enable better farm management and an optimal use of resources (e.g. Water, soil). Although first robotic developments focused on autonomous tractors, the cost reductions of light autonomous vehicles, especially drones, make these types of vehicles more attractive in the new paradigm of precision agriculture.

Whilst mobile agricultural ground robots bring many capabilities, their design is intrinsically challenging due to the harsh conditions imposed by the agricultural scenarios, such as high temperatures, terrain with slopes, difficult communications and non-geometrical location structure. In the Douro region, vineyards can be found installed in the steep-slope hills, where mechanization has a very low adoption rate due to lack of space for maneuvering and other conditions that impact machinery.

This work aims the development of a cost-effective open-source robotic platform, capable of handling the mentioned difficulties. The robot considers a 2D LIDAR, a Global Navigation Satellite System (GNSS) receiver module, odometry and an Inertial Measurement Unit (IMU) for self-localization and mapping. All the developed sub-systems, software, hardware and mechanical components are detailed, along with the testing procedures and obtained results. To reduce the overall cost, a Raspberry Pi computer is considered as the main processing unit. A validation of state-of-the-art algorithm Cartographer SLAM is done in different settings, along with an analysis of it's performance on cost-effective hardware.

The final result was an open-source platform, capable of executing mission plans, which is to be fitted with additional hardware for the intended task. A robust localization solution with cost-effective hardware was achieved, along with a navigation stack and a user interface with integration of the application QGroundControl.

Agradecimentos

Gostaria de agradecer aos meus orientadores, Professor Pedro Costa, Professor José Lima e em especial ao Engenheiro Filipe Neves dos Santos, por me aceitarem no INESC TEC e orientarem a minha tese, foram imprescindíveis nesta jornada. Um agradecimento também à restante equipa do departamento de robótica, pelo feedback dado ao longo do semestre.

Aos meus amigos e amigas por todos os momentos de descontração, dificuldade, euforia e trabalho. Acompanharam-me neste último desafio de 5 anos, conhecem o meu melhor e o meu pior, e onde quer que a vida nos leve daqui para a frente, espero continuar a contar com o vosso apoio, da mesma forma que podem contar comigo.

Por fim, agradeço aos meus pais, Maria e Júlio, por todas oportunidades que, a muito sacrifício, me puderam providenciar. O mais profundo agradecimento por toda a compreensão, pelo conhecimento, pela forma de encarar a vida que me transmitiram, a independência e a luta constante por um amanhã melhor, valores que moldam quem eu sou hoje e garantem o sucesso do meu futuro daqui para a frente.

Filipe Mendes.

"Doubt is an uncomfortable condition, but certainty is a ridiculous one."

Voltaire

Contents

List of Figures

List of Tables

Abbreviations and Symbols

2D	Two-dimensional
3D	Three-dimensional
API	Application Programming Interface
BLDC	Brushless DC
DGNSS	Differential Global Navigation Satellite System
EKF	Extended Kalman Filter
FRE	Field Robot Event
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
IO	Input/Output
IoT	Internet of Things
LED	Light Emitting Diode
LIDAR	Light Detection And Ranging
RGB-D	Red Green Blue - Depth
ROS	Robot Operating System
RTK-GNSS	Real-Time Kinetic GNSS
SLAM	Simultaneous Localization and Mapping
TOF	Time-of-flight
UTM	Universal Transverse Mercator

Chapter 1

Introduction

An increase in population and growth rates, doubling since the 1960's, combined with fast improvement of life expectancy in a world where approximately half of the global population's calorie intake originates from agricultural products [1], strongly drives the need for an increased and more efficient agriculture activity. In an activity highly dependent on manual labor and amount of land, scalability becomes constrained, and trades between sustainable growth and environmental damage surge [2].

Technological advancements in robotic technologies, allied to a sensor price decrease, enable the development of cost effective robotic solutions, not only to overcome manual labor costs, but also to improve crop quality, resource efficiency, as well as environmental sustainability.

> *In the last years, multiple groups around the world have applied different automation solutions (e.g. sensor networks, manipulators, ground vehicles and aerial robots) to diverse agricultural tasks (e.g. planting and harvesting, environmental monitoring, supply of water and nutrients, and detection and treatment of plagues and diseases).* [3, chap. Introduction]

Robots play an important role in the shift towards precision agriculture, especially in open-field farming (opposed to greenhouse farming). However, agricultural environments present a need for more robust and adaptable robots, capable of operating in varying conditions, harsh terrain with high unpredictability and other special requirements (e.g. careful harvesting of produce). Solutions often used in the industrial settings, where it's possible even to design the plant to reduce the robot's limitations, are often not appropriate for agriculture due to the reduced structured scenario and control. The lack of permanent natural features (wall, corners) challenges the localization algorithms, imposing the usage of complex localization algorithms to deal with dynamic changes.

Some of the common challenges when designing open-field agricultural robots reside firstly in the careful design of locomotion systems and the corresponding mechanical project. Ground robots must be able to traverse terrain with a variety of perturbations and changes in ground texture (e.g. bumpy or soft terrain), unpredictable obstacles (e.g. branches or stones) and in some cases changes in terrain (e.g. from gravel to dirt).

Vision-based technologies are challenging in a setting naturally filled with noise, for example when detecting a fruit, it must be able to distinguish between it and leaves, branches, stems, stones and other naturally present elements in an environment with different lighting conditions and weather [4]. Recent improvements in vision-based control, algorithms and technology bring better performance in vision dependent applications, such as harvesting [5], or localization using natural feature detection in locations where the Global Navigation Satellite System (GNSS) signal isn't always available or accurate [6].

1.1 Motivation and Objectives

One of the challenges in agricultural robotics is achieving a cost-effective ground robot, able to fulfill monitoring and other tasks autonomously, such as acquiring crop quality maps, apply pesticide, harvest or prune.

The steep-hill environment of river Douro's mountain vineyards imposes bigger challenges than typical agricultural environments. Since Global Navigation Satellite System (GNSS) availability is low in steep-hills due to loss of line-of-sight in high elevation terrains (some moments with less than five satellites in view), the usual localization problem solution that involves Differential GNSS (DGNSS) is out of question. This brings the need of using other localization strategies and approaches.

To deal with these issues, in 2017, surges the project RoMoVi - Modular and Cooperative Robot for Vineyards. This project is being developed by Tekever (leader), INESC TEC and ADVID. The RoMoVi project aims to develop robotic components as well as a modular and extensible mobile platform, which will allow in the future to provide commercial solutions for hillside vineyards, capable of autonomously executing operations such as monitoring and logistics.

RoMoVi is aligned with the agricultural and forestry Robotics and IoT R&D line, called AgRob, being developed at INESC TEC (`http://agrob.inesctec.pt`). AgRob R&D focuses on developing robotic and IoT solutions for the agricultural and forestry sectors, considering the Portuguese reality. Based on both the RoMoVi and AgRob's results and vision, a necessity for an open-source robot was identified, with the following features:

- Wheeled differential locomotion;
- Platform with basic sensors: LIDAR, IMU and GNSS receiver;
- Assembling cost lower than 3000€;
- Capacity to carry a payload of 20kg; and,
- Small dimension so it can be used on Field Robot Event competition;

To accomplish this goal, it was required to develop robotic software sub-systems in the Robotic Operative System (ROS), select their interfaces and integrate with selected mechanical and electrical components. The project can be divided into three main tasks:

- Project and implementation of a robot platform for agriculture.
- Analysis and integration of existing ROS packages for localization and mapping.
- Development of a user interface for mission planning and tracking the robot's status.

1.2 Main contributions

The main contribution of this work is an open-source, low cost, simple and robust robot able to autonomously monitor crops using simple sensors, also usable on the Field Robot Event competition. This work is publicly available at https://gitlab.inesctec.pt/CRIIS/public/agrobv19.

The second contribution of this work was the integration and validation of Cartographer - Simultaneous, Localization and mapping (SLAM) module - considering a cost-effective agricultural robot. Further, a comparison between Cartographer and Hector SLAM was performed.

The third contribution of this work is the development of a ROS package containing an API for sending commands via a CAN interface, for the Nanotec C5-E motor controller. This package is currently configured for translating ROS standard velocity commands and actuating on a two-wheel differential drive robot.

The fourth contribution is a ROS package which integrates the application QGroundControl, using the MAVLINK protocol over UDP, accepting mission plans and publishing the waypoints in the robot's local coordinate frame.

The final contribution is the integration of GNSS data with Cartographer's 2D SLAM problem. A pull request was submitted and is currently awaiting review.

1.3 Thesis structure

Besides the introduction, this thesis contains 5 more chapters. In chapter 2, fundamentals of modern robotics are explored along with state-of-the-art technologies. In chapter 3, the developed algorithms, solutions and software are presented. Chapter 4 details the hardware and components used, along with their interfaces and assembly. Chapter 5 describes the conducted experiments and their results. Finally, chapter 6 presents the conclusions, limitations, and suggestions for future work.

Chapter 2

Background and Related Work

First and foremost, *Autonomous Mobile Robots* by Siegwart and Nourbakhsh [7] clearly provides the essentials of mobile robotics, and a launch point into more advanced topics. As such, it was used as a solid basis for comprehension of the topics described in the following chapter.

Autonomous navigation is one of the most challenging abilities required by mobile robots, since it depends on it's four fundamental tasks: **Perception**, the robot must acquire and interpret the necessary data; **Localization**, the robot must determine it's position; **Cognition**, the robot must take the decision on how to achieve it's mission; and finally **Motion Control**, where the robot must correctly adjust it's actuators in order execute the planned path.

This chapter was divided based on the aforementioned tasks and it's content was tailored based on the systems that will be used in this project. First, an examination of the sensors is presented, followed by an analysis of localization techniques then, a view of the state-of-the-art estimation algorithms concluding then with motion planning strategies. Finally, existing solutions for the proposed problem are examined, both academic and commercial.

2.1 Sensors

Any autonomous mobile system must be able to acquire knowledge about itself and the surrounding environment. This challenge only worsens when considering outdoor operating environments due to various reasons, but mainly due to the necessary increase in range of the robot's sensors and the introduced complications that naturally derive from an unstructured environment. In order to better understand how these technologies support mobile robotics, Everett's book *Sensors For Mobile Robots: Theory and Application* [8] was essential as it presents an overview of both the theory of sensor operation as well as practical considerations when designing a system.

There are a variety of sensors for many different applications in robotics, as shown in table 2.1. The sensors approached in this section were chosen for their relevance to the thesis, since these are the most popular sensors in low-cost agricultural ground mobile robotics, as will be later shown in this report, and also since these will be used in the development of the project. First, general considerations about the choice of sensors in mobile robots is presented, followed by an analysis

of operating principle, their main specifications, their limitations and their use in mobile robotics, both general and similar applications to this thesis.

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers Optical barriers Noncontact proximity sensors	EC EC EC	P A A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders Potentiometers Synchros, resolvers Optical encoders Magnetic encoders Inductive encoders Capacitive encoders	PC PC PC PC PC PC PC	P P A A A A A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes Inclinometers	EC PC EC	P P A/P
Ground-based beacons (localization in a fixed reference frame)	GPS Active optical or RF beacons Active ultrasonic beacons Reflective beacons	EC EC EC EC	A A A A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors Ultrasonic sensor Laser rangefinder Optical triangulation (1D) Structured light (2D)	EC EC EC EC EC	A A A A A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar Doppler sound	EC EC	A A
Vision-based sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s) Visual ranging packages Object tracking packages	EC	P

Table 2.1: From Siegwart's book [7, chap. Perception]. Classifies sensor by use, the technology used, proprioceptive/exterocpetive (PC/EC) and active/passive (A/P)

2.1.1 General Design Considerations

Each type of sensors possess their own unique specifications which should be balanced against the final system's application, but there are still general traits that can be used to define and compare sensors. The following considerations were extracted from Everett's [8] book:

- **Field of View** - Width and depth of sensing field.
- **Range Capability** - Minimum, maximum and maximum effective range.
- **Accuracy and resolution** - Must fulfill the system's requirements.

- **Ability to detect all objects in environment** - Some objects or surfaces may interfere with normal sensor operation. Environment conditions, noise and their interference should also be considered.

- **Real-time operation** - Update frequency, often related with the system's speed. For example, an obstacle avoidance system much detect the object with a certain advance in order to execute security measures.

- **Concise, easy to interpret data** - The output of the data should be adapted to the application (too much data can also be meaningless). Problems here can be reduced by preprocessing the input data.

- **Redundancy** - Multi-sensor configurations should not be useless in case of loss of a sensing element. The extra sensors can also be used to improve the confidence level of the output.

- **Simplicity** - Low-cost, modularity, easy maintenance and easy upgrades are all desirable traits.

- **Power Consumption** - Must be in check with the intended system's power limitations on a mobile vehicle (Both power and battery capacity wise).

- **Size** - It's weight and size should be in check with the intended system.

However, theoretical characterization of the sensors is not enough since system's frequently have restrictions in integration and field operation. Therefore expert information and field testing provided by the manufacturer should be consulted when designing such a mobile system.

2.1.2 Laser rangefinder

Laser rangefinders, also known as *LIDARs*, uses a laser beam to measure distances to objects using a time of flight (TOF) principle, either using a pulsed laser and measuring the elapsed time directly (not commonly used since it requires resolution in the picoseconds range) or by measuring the phase shift of the reflected light. Another manner is using a geometrical approach and a photo-sensitive device, or array of devices, in order to accomplish optical triangulation.

Phase-shift measurement One approach for measuring the TOF where the sensor transmits amplitude modulated light at a fixed frequency, is measuring the phase shift between the emitted beam and the received reflection. A diagram explaining phase shift measurements is presented in figure 2.1, where distance to target is calculated as such:

$$D = \frac{\lambda}{4\pi}\theta \tag{2.1}$$

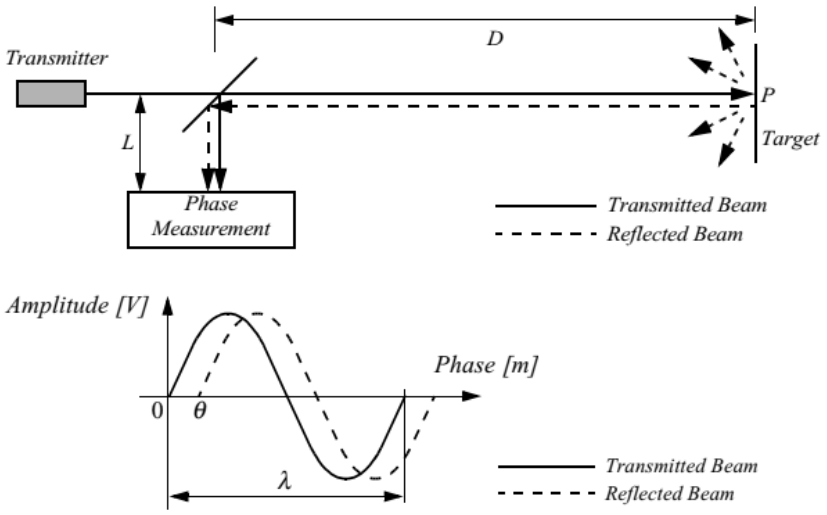

Figure 2.1: Range estimation by measuring phase shift. Image from Siegwart's book [7].

Optical triangulation Uses a geometric approach by projecting a known light pattern (e.g single point, multi-point, stripe pattern, etc.) and capturing it's reflection using a position-sensitive device. Then, with known geometric values, the system is able to use triangulation in order to measure the object's distance to sensor (Fig. 2.2). Optical triangulation is also used in 2D and 3D LIDARs, by projecting known light patterns and estimating the target's shape and position by evaluating the distortion caused in the pattern.

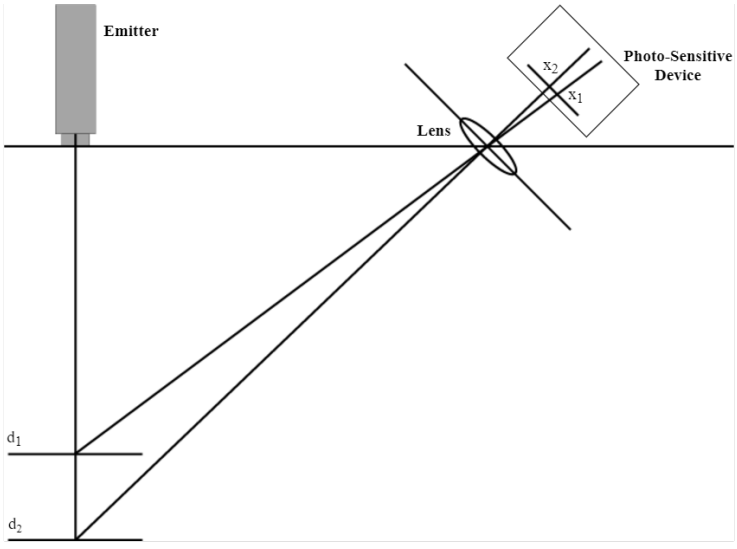

Figure 2.2: One dimensional rangefinder measuring two objects and different distances and how it reflects and projects to the sensor.

Laser rangefinders are some of the most precise sensors used in robotics with precision in the millimeter range but, its precision comes at a high cost in comparison with other rangefinding technologies (e.g. ultrasound). Even though laser-based sensors have relatively high immunity

to noise, it's use in outdoor, unstructured or somewhat structured environments is limited due to varying factors, from environmental conditions such as fog or systematic limitations when detecting, for example, an optically transparent object.

The main characteristics to be considered when considering laser rangefinders should be its minimum and maximum range, it's scan angle, the angular (when applicable) and distance resolution.

2.1.3 Global Navigation Satellite System

GNSS or Global Navigation Satellite System is a satellite beacon navigation system that provides location and time information worldwide. One example is the NAVSTAR Global Positing System (GPS), owned by the United States government, launched in 1978 by the U.S. Department of Defense, and started exclusively for military applications. It was then progressively allowed for civilian use during the 1980s.

The GPS is usually composed by three segments:

- **Space Segment**: Composed by 24 to 32 satellites, in medium Earth orbit, with an orbital period of approximately 12 hours. Their orbits are arranged in a way that there is direct line-of-sight with at least 6 satellites at any time of the day.
- **Control Segment**: Provides support for GPS users and the space segments. Synchronizes space and user segment, so that transmissions are done with syncronized time stamps, guaranteeing that the system performs as intended.
- **User Segment**: GPS receiver with an antenna, processing and a precise clock.

This system uses trilateration with the time-of-flight of principle in order to determine it's current position. In theory, in order to obtain the current location, 4 satellites are required (intersection of 4 spheres), although with 3 satellites (intersection of 3 spheres is two points) the location is deduced since one of the points will be far above or below the earth's surface and can be excluded. In practice, a higher number of satellites within line-of-sight of the receiver will substantially increase it's precision, although it's still around a couple meters.

Since each satellite continuously sends it's current location and time, a user only needs a passive GNSS receiver. The continuous transmission is low-power so one of it's limitations is requiring direct line-of-sight to at least 4 satellites, making this system unusable in indoor systems without further adaptations. Theses factors make the use of GNSS popular for robots used in open-spaces and unmanned aerial vehicles.

GNSS cannot be used as a standalone localization technology, but developments around overcoming some of the precision limitations have used a fixed terrestrial station that is calibrated to know it's position with high accuracy (e.g. 24 hour calibration by use of a constellation map) and providing the data to the mobile robot, which corrects it's position [9]. This technique is known as Differential DGSS (DGSS).

2.1.4 Dead Reckoning Sensors

Dead Reckoning, in mobile robotics, is the process of estimating one's current position by taking a previous known position and evaluating it's evolution based on the robot's dynamics (i.e. speed and acceleration). Dead reckoning in robotics is typically achieved using wheel odometry and an IMU (Inertial Measurement Unit). Most ground-based autonomous robots use Dead Reckoning as a baseline for navigation due to it's high update frequency and then periodically correct it using other localization technologies (e.g. GNSS).

2.1.4.1 Optical Encoders

Optical incremental encoders are rotary encoders that use a light emitter (typically LED), a patterned mask, a disk and a sensor in order to determine the angular velocity and position of a motor, shaft or a wheel. Largely used in mobile robotics for motor control and Dead Reckoning. Since they are the most popular solution for measuring angular speed and shaft position and have avast range of applications, the development was pushed for high-quality and low-cost encoders, which greatly benefited mobile robotics.

The main specification when considering optical encoders is its resolution, which usually comes at a higher cost. Typical limitations are noise sensitivity at extremely low-speed, its finite resolution and errors introduced by discretziation as well as other approximations in odometry equations.

2.1.4.2 Inertial Measurement Unit

Inertial Measurement Units combine three sensors in order to acquire enough information to estimate the orientation, position, and velocity. Typical 9 degrees of freedom IMUs incorporate:

- **Three-axis Magnetometer** - Provides three dimensional orientation referenced to the earth's magnetic field. Has limited performance in indoor and high EM noise environments.
- **Three-axis Gyroscope** - Provides an angular speed referenced to the corresponding axis, which can be then integrated to estimate the robot's current pose and relative motion.
- **Three-axis Accelerometer** - Measures it's proper acceleration, with an output that gives the direction and magnitude of the acceleration vector. Single and double integration provides us with complimentary speed and position information, respectively.

Although there are heavy limitations in the IMU's performance, especially regarding sensitivity to noise and integration errors, their small size, low power consumption, and low cost make these sensors extremely popular in a number of robotics applications.

2.1.5 Cameras

Vision provides us, as well as robots, large amounts of information about our environment and allows for task realization in dynamic environments. It plays a role in many of a robot's typical tasks

such as localization, mapping, object recognition, obstacle avoidance and navigation. Therefore significant efforts were made in the development of vision-based systems and it's first stage, the camera.

Focusing on navigation, the pinhole model (fig: 2.3) is essential in order to better understand how cameras can be used to obtain three-dimensional space information and how it's projected into the image plane. Although the model doesn't account for some of the sensor's distortions (e.g. discretization and lens deformation), it is a sufficient approximation for a variety of applications in computer vision.

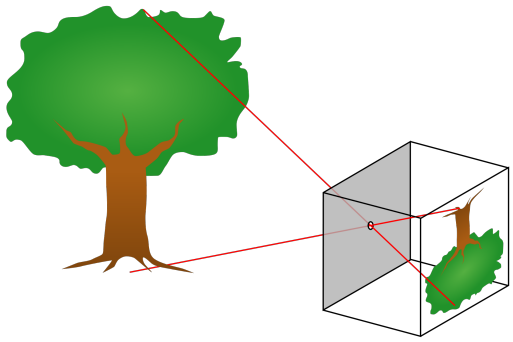

Figure 2.3: Visual diagram of the pinhole camera model. The camera's aperture is simplified to a point and no lens is assumed, so it becomes a first-order approximation for the two-dimensional mapping of a three-dimensional scene. Image from Wikipedia [10].

There are three type of camera's typically used in mobile robotics:

- **Projective cameras** - Most common cameras, approximated by the pinhole projective model, from which they get their name, with it's focal point being the center of the lens.
- **Omnidirectional camera** - Has a field of view that either covers the entire sphere of view or a circle in it's horizontal plane. Used for visual odometry and visual SLAM [11].
- **Fisheye Camera** - Composed of a projective camera and a fisheye lens, in order to increase the field of view or make it omnidirectional. The introduction of the lens requires appropriate models, such as the proposed by Courbon et al. [12], as the pinhole model cannot be used.

Using cameras to extract three-dimensional space information (including depth) is also possible. Murray and Little propose a **Stereo Vision** based system [13] for autonomous grid occupancy mapping of an indoor environment, a setup where two cameras are used and the relative position of the detections is triangulated. Stereo and monocular setups will be discussed later in the Visual Odometry subsection.

2.2 Localization and Motion Estimation

Before exploring each technique, the general robot localization problem must be considered along with a solution strategy in context of the considered setting in this thesis. As a robot moves from

a well-defined starting point, the robot keeps track of it's movement through odometry. But, due to accumulation of the odometry errors (e.g. wheel slippage, undefined point of contact with the ground), the robot's uncertainty about it's current position increases. So, the robot must periodically correct it's position with other localization sensors (LIDAR, GNSS or visual sensors) and use them to estimate it's location relative to to it's environment. This two-step process allows the robot to determine it's position relative to a map. Here, the distinction between proprioceptive and exteroceptive sensors is evident, as they play roles in the two distinct steps, **prediction** and **update**.

In this section, Dead Reckoning, the evolution of wheel odometry calibration and error estimation methods are studied, both On-line and Off-line. Following, comes an analysis of the more recent area Visual Odometry, a description of state-of-the-art of 2D laser rangefinder techniques (here the focus is on 2D, since it will be used in this project), and finally Beacon Based Localization. IMU and GNSS were not be addressed in this section since their operational principles are simple and well-defined.

2.2.1 Dead Reckoning

Odometry is the use of information from motion sensors for estimating a robot's change in position over time. Dead Reckoning uses information from proprioceptive sensors, wheel encoders and IMU, while visual odometry uses visual techniques and the change in images in order to estimate motion.

One of the most popular techniques for estimating the internal state and dynamics of a mobile robot is the use of wheel sensors in order to measure rotation speed and integrate in order to obtain the robot's current position. Two categories can be identified of error sources in odometry: **Systematic** errors, caused by the system's internal problems, which usually presents as a bias. **Non-systematic** errors, on the other hand, are caused by external factors and are independent of the system's characteristics, making it unbiased (random).

2.2.1.1 Wheel Odometry Calibration

Calibration is the process of comparing the outputs of a device under test to a reference value and correcting the parameters of the system in order to minimize errors (systematic or non-systematic). Wheel Odometry calibration can be divided into two groups.

Off-line methods These methods use a previously established test path that is followed by the robot without corrections and then, using the difference between the actual position and the estimated position, compensate it's systematic errors. Off-line methods' main advantage is that they can be done without requiring any additional sensors, although the final pose must be still measured manually which may introduce further errors. The most popular method is UMBmark presented by Borenstein [14], where a closed rectangular trajectory allowing for calibration of some systematic errors, but it was surpassed by more recent calibration procedures such as Antonelli's [15] least-squares methods and the procedure *PC-method* developed by Doh et al. [16].

On-line methods Here, calibration is accomplished in real-time by using external sensors to periodically correct the odometry. The probabilistic approach also makes them more robust and less drift biased, but intimately ties their performance to sensor performance and the used models. One of the first proposed algorithms was by Roy and Thrun [17], an algorithm that automatically calibrates as it operates, with the concept of *life-long calibration*. Most recent trends rely on the assumption of the Gaussian distributions of error and the extremely popular Kalman filters. Martinelli et al. [18] developed an algorithm that uses laser and vision information to simultaneously calibrate the estimations and characterize the systematic error.

2.2.2 Visual Odometry

Visual Odometry in robotics is a process used to estimate a robot's pose and movement gby comparing a sequence of camera images. Recent developments in processors and graphic processors along with general increases in computational power have increased the attractiveness of Computer Vision and, to the mobile robotics community specifically, the research into motion estimation. The term *Visual Odometry* was first introduced by Nister [19] which uses a stereo (multi-camera) setup and presents his algorithm based on the Harris detector. In monocular setups, Yamaguchi et al. implemented a monocular camera with feature detection in order to realize ego-motion estimation. Both these approaches are separated into two main categories, **Stereo Vision** which uses more than one camera in order to capture depth information and **Monocular Setups** which use only one camera. These methods can also then be divided into **feature-based** methods or **dense** methods.

Monocular Camera Setups In single camera visual odometry systems, three-dimensional information must be extracted from two-dimensional data, allowing us to retrieve relative information but not in an absolute scale. Translating the data to a three-dimensional absolute scale is possible using extra input from other exteroceptive sensors (IMU, LIDAR, wheel odometry,...). Here, the computation of the new frames and the new camera pose estimation must be done for every new image. Forster et al. [20] provides a method for motion estimation with small number of features and extracted and without costly matching techniques. One of the most currently popular SLAM monocular algorithm based on visual odometry was proposed in 2015 by Mur-Artal [21].

Stereo Vision Uses multiple cameras and triangulation of the detected features for estimation of motion. Since the distance between the two cameras (stereo baseline) is fixed and known, the triangulation and estimation process is more accurate than it's monocular counterparts. The most important developments for stereo vision were done in the Mars Rover context, where the final version was developed by Cheng et al. [22]. A second version of Mur-Artal's work called ORB-SLAM2 provides an expansion of his own method to stereo and RGB-D cameras [21]. One of the main disadvantages of stereo setups is the need for very precise calibration and baseline definition, since variations in the baseline measure strongly effects motion estimation accuracy. Another important consideration is the relation between the baseline distance and the distance of

features in the target scene (application dependent), where if the stereo baseline is much shorter, these methods become inaccurate.

Feature-based methods In computer vision, a feature is a fragment of data which is relevant for solving a specific task. In the visual odometry pipeline, research has focused on **Feature Extraction**, with the groundbreaking AlexNet as developed by Krizhevsky [23] and furthered by Szegedy et al. [24], both based on convolutional neural networks. Novel **Feature Matching** methods that are robust or invariant to scale, rotations and lighting conditions were also developed, the most notable being SURF [25] and SIFT [26].

Dense methods Also known as **Optical Flow methods**, use information from the whole image or an image's subregions in order to perceive motion from sequential variation in images. Theses methods are losing popularity against their feature-based counterpart since dense methods are only suitable when dealing with motion in a small scale environment.

2.2.3 2D Laser Rangefinder Methods

Since a robot cannot use data solely from odometry for precise navigation due to unbounded error, one of the possible solutions for correction is the use of a 2D laser rangefinder, which is also commonly used for localization [27],[28],[29], map building [30] and collision avoidance [31].

Thrun suggests a model and intrinsic parameter estimation algorithm for laser rangefinders that include four types of measurement errors: small measurement noise, errors due to unexpected objects, errors due to failures to detect objects, and random noise [32].

A popular application of laser rangefinders in localization is line extraction for map building. The classic method for line extraction is the Hough Transform [33], but better performing alternatives came through such as the RANSAC [34], and the Split-and-Merge which came from the computer vision community [35]. Feature extraction for map building, motion estimation with image matching, and SLAM have been the focus of recent developments in 2D laser rangefinders [36].

2.2.4 Beacon based localization

Another absolute localization strategy, useful in this case for the periodic correction of the odometry errors, is the use of beacons which act as absolute references in the map. Beacon based localization comes as an alternative to DGNSS (more expensive, GNSS signal dependent) or landmark based localization (detection of landmarks can be unreliable). This strategies' biggest limitation is associated with indoor use, especially in narrow corridors and close-quarters complex maps. Also, a beacon alone is not enough to accurately ascertain the robot's localization, where at least three are required. In order to deal with this situation beacon placement should also be carefully considered in order to maximize signal coverage on the operation area. Next, most common positioning methods are presented: **Trilateration** and **Triangulation**.

Trilateration Uses a method based on the distance to the beacons which is measured using the TOF principle. Theoretically, only 3 beacons are necessary to deduce the robot's position, since the intersection of the three circles drawn from the distance will intersect in one point (fig. 2.4). In practical applications the measurements are subject to noise and the returned location will be a region instead of a single point, so generally speaking an increase in the number of beacons will increase the accuracy of the localization prediction.

Triangulation This method uses the Angle of Arrival (AoA) principle to ascertain the robot's position. Since it relies on the geometric properties of triangles, it theoretically only needs 2 beacons to function but in practice (Fig. 2.4) more beacons are desirable. Although this technique requires less beacons and doesn't require time synchronization between the beacons, the hardware necessary to precisely determine the angle of arrival of a transmitted wave is very expensive, which make these techniques undesirable for the robotics community.

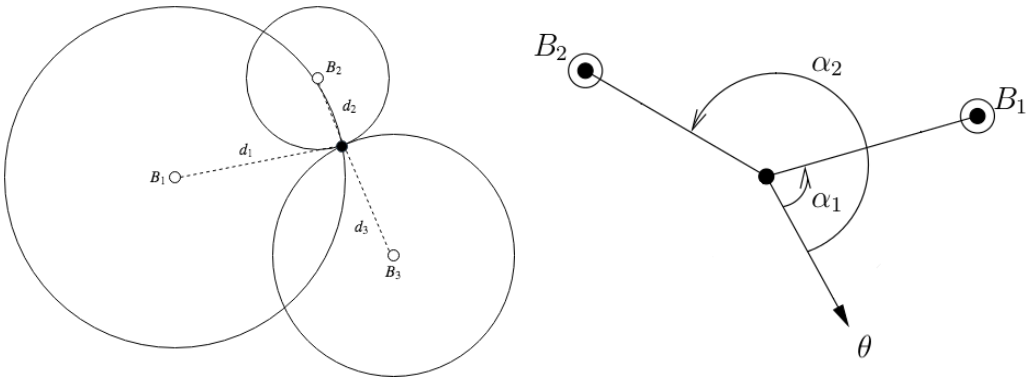

Figure 2.4: Trilateration using 3 beacon (left) and Triangulation using 2 beacons (right)

Since both these techniques are subject to measurement errors, and typically return a point region instead of a definitive solution, another estimation algorithm must be used such as the EKF localization estimator which will described in the following section.

2.3 Fusion and Estimation Algorithms

When facing uncertainty in the study of the algorithms presented below, the book *Probabilistic Robots* [32], considered the standard reference for robotic navigation and mapping, provides a clear-cut analysis to the standard approaches used in robotics for estimation and fusion, SLAM, navigation and decision making.

First the fundamental concepts of modern localization methods are presented, followed by an analysis of the two popular categories of probabilistic localization methods, Kalman and Markov. Finally, the SLAM problem is defined along some of the most popular solutions.

2.3.1 State and Belief

As robots move away from the tightly controlled industrial setting into more dynamic and unstructured environments, coping with uncertainty becomes a more important requirement in robotic systems. A probabilistic robot uses probability theory concepts to explicitly model the system and it's uncertainty, posteriorly using the space of probabilistic possibilities to control the robot, instead of using one unique set of information that is assumed to be true. Since sensor outputs are only measurements of states and can be affected by errors, probabilistic methods prove themselves to be more robust and accurate.

The probability theory formulated by Bayes describes the robot's **Beliefs** referenced to it's **States**, **Recursive Bayesian Estimation** then estimates a system's state based on it's past state and beliefs.

2.3.1.1 State

State is a set of variables that describes a robot as well as the environment's aspects that impact a known system. These variables may be **static** or **dynamic** and are used to describe both **external** or **internal** information. Examples of state variables used in robotics are:

- A robot's pose given by six state variables, three Cartesian coordinates that indicate the robot's position (length, width, height) and three Euler angles to describe the pose (roll, pitch and yaw). Also known as **Kinematic state**.
- The configuration of a robotic manipulator joints (linear, orthogonal, rotational,...) and their position at any point and time.
- Information regarding velocity and forces applied to a robot and it's joints. Also known as **Dynamic State**.
- Location and features of objects, beacons, and landmarks in an environment.
- Other information, such as the battery level of a battery-powered system.

2.3.1.2 Belief

As discussed before, a sensor doesn't measure a state directly. Therefore, a probabilistic robot does not adjust it's outputs directly based on the sensor data, but through the state information "guessed" from the sensor data. Thrun [32] defines beliefs as *"A belief reflects the robot's internal knowledge about the state environment."*, and these beliefs are defined using probability theory as conditional probabilities. In this example state variables are represented as x_t, all past measurements as $z_{1:t}$ and all past controls as $u_{1:t}$.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.2}$$

2.3.1.3 Bayes Filter

In order to better understand how more complex filters and algorithms work and how they bridge over to robotics, the analysis of the **Bayes Filter Algorithm** or **Recursive Bayesian Estimator** is essential since it provides the basis of the most popular estimation techniques such as the Kalman and Particle filter.

Algorithm 1 Bayes Filter

1: **procedure** BAYESFILTER($bel(x_{t-1}), u_t, z_t$)
2: **for** all x_t **do**
3: $\bar{bel}(x_t) = \sum p(x_t|u_t, x_{t-1}) bel(x_{t-1})$
4: $bel(x_t) = \eta\, p(z_t|x_t) \bar{bel}(x_t)$
return $bel(x_t)$

In line 3, a belief of the current state is given based on the state's transition model, it's belief in the past state and the control inputs. It's belief assigned to the state x_t is based only on the past belief of the state and the probability that the control u_t induces a change from x_{t-1} to x_t. This step is usually called **prediction** as it only uses the state's transition model and it's inputs, with no regards to external measurements.

In line 4, the belief of the robot is updated taking into consideration the belief of the robot's current state and the probability that the measurements z_t were observed, which is then multiplied by a normalization constant η. This adds an a correction to the predicted belief and is typically callled **measurement update**.

2.3.2 Markov and Kalman

Before diving deeper into each class, a brief introduction and comparison of both methods is necessary. First, **Kalman filter localization** models the system's belief using a matrix containing Gaussian probability density functions, which can be defined by only its mean μ and it's standard deviation σ. To update the Kalman Filter is to update the parameters of it's Gaussian distribution. These algorithms are suited for continuous world representations, but are not stable in face of large displacements in position (e.g. collision, kidnapping). Second, Markov localization represents multiple points or regions and assigns probabilities for each of the robot's possible poses. In the **update** step, the algorithm must update every considered point or region making Markov localization methods usually more computationally demanding.

2.3.3 Kalman Filters

The **Kalman's Filter** [37] basic assumption is that the considered systems are linear, stochastic and ergodic, this means that although it's output will be affected by random white noise, it's statistical properties will be retained. Kalman Filter uses linear equations to deal with Gaussian distributions defined simply by their mean and variance. Kalman Filter's proves effective in systems that can accurately be modeled by linear models and whose sensor error can be approximated to a Gaussian

distribution. But this restricts the algorithm to simple robotics problem in a controlled environment, so in order to empower the Kalman Filter, the Extented Kalman Filter was developed.

2.3.3.1 Extended Kalman Filter

In the Extended Kalman Filter, state transition and measurement models are first-order, differentiable functions. Instead of evaluating only linear equations, EKF evaluates the Jacobian (matrix of partial derivatives) of the models, where the state prediction and updates propagate through a non-linear system, while the errors propagate through a linearized Taylor's series.

The EKF generally retains the benefits of the Kalman Filter regarding simplicity and computational efficiency, but are still dependent on the accuracy of the approximation, since the filter's performance and stability depend on keeping the uncertainty small.

It is also the most popular algorithm for state estimation and data fusion in robotics [38],[39], although EKF still faces performance issues when dealing with highly non-linear systems.

2.3.4 Markov Filters

Markov Filters Remove the assumptions used in Gaussian filters and use a finite set of points distributed throughout a specific region of space, each with an assigned probabilistic value. This makes Markov Filters more suitable when dealing with global uncertainty, feature matching and complex non-linear systems, trading off with computational demand tied to the number of parameters used in the model. Markov Filters can generally be divided into two classes.

Histogram Filters Which divide state-space regions and assign each one with a probability. A well-known implementation of this filter is the **Grid Localization**, where the regions are divided into squares each with an assigned probability. This filter has a particular trade-off: cell-size (and number) vs. computational cost. More modern solutions have proposed algorithms with variable grid sizes and shapes according to the robot's current state [40].

Particle Filters Uses a distribution of samples, points called particles, spread around the robot's localization (the denser, the more probable that the state falls into that region), each representing a possibility with assigned probability of the robot's pose. The most popular particle filter technique, **Monte Carlo Localization**, was developed by Thrun et al. [41] in 2001, solving the global localization and the kidnapping problem in a robust and efficient way, while needing no feedback of it's initial position if a map is provided. Although particle filters are generally a better solution than Kalman Filters, the required computational cost makes it unfit for low-cost systems.

Figure 2.5 shows a visual state representation of these the Kalman filter, the grid Markov filter and the particle filter.

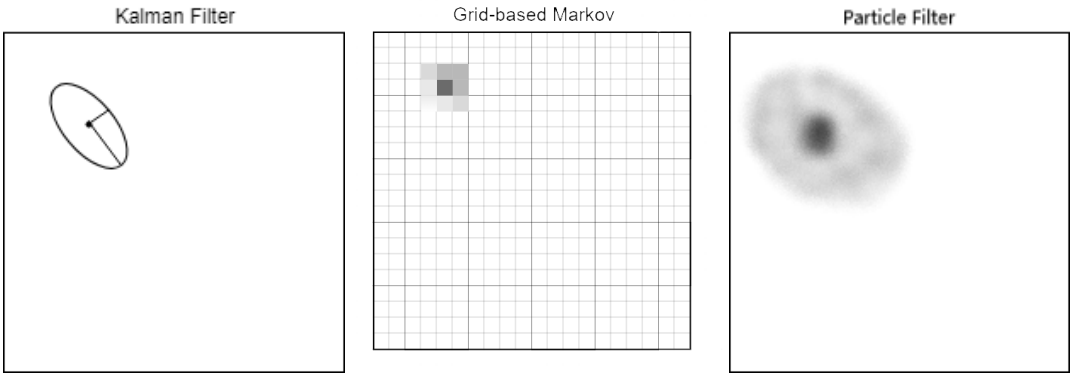

Figure 2.5: State representations of Kalman, Grid and Particle estimation, respectively.

2.3.5 Simultaneous Localization and Mapping

The aforementioned algorithms are closely tied with mapping procedure, and are appropriate for problems either where a robot's pose is known and the environment has to be mapped, or where the map of the environment is provided and the pose must be determined.

If a robot neither has been provided a map, neither it's own pose, we arrive at a problem known as **Simultaneous Localization and Mapping**, which has been widely researched by the robotics community. One of the earliest and popular algorithms is based on the Extended Kalman Filter and is known as EKF-SLAM, but due to it's high complexity and need for non-ambigous landmarks, it has been replaced as the de facto standard by FastSlam [42]. Great methods have also risen from the robotics vision community for solving the SLAM problem using visual odometry and mapping, such as the ORB-SLAM [21], PTAM [43] and RGBD-SLAM [44].

2.4 Robot Motion

Robot Motion is where the processed measurement data and the mission details are interpreted and used to make a decision, along with the following execution. Typical robot motion architectures are separated into three parts which will be addressed in this section: **Motion Planning** The robot's path is planned; **Obstacle Detection** The robot must override the plan and take security measures in case of an obstacle; **Task Execution** The robot executed the planned path. In this work defined as Motor Control, since they are this system's outputs.

2.4.1 Motion Planning

In the autonomous navigation process, path planning is one of the simpler challenges and was already heavily studied due to it's applications in industrial robotics, where high speeds and big payloads require a precise control of the forces exerted on the manipulator's joints. Since mobile robotics generally deals with low speeds, dynamics are not considered during path planning. Path planning should be adapted to the robot's specific application which is also strictly tied with the

mapping process. A variety of strategies over the years emerged for solving this problem, and the most popular ones are described here.

Road map Theses techniques decompose the map into a graph network where the robot's possible motions are simplified to a one-dimensional connection, simplifying the decision making to a shortest path graph problem. The complexity of Road map techniques lie on the obtained map decomposition, here the most popular techniques are the Visibility Graph [45], which returns the shortest possible path, and Voronoi diagrams [46] which calculates a path that maximizes the distance between the robot and obstacles.

Fuzzy navigation Approaches using Fuzzy logic are by far the most popular techniques used today, since their simplicity and versatility in defining heuristics and incorporating subjectivity allows the modification of conventional algorithms making them more robust [47], [48].

Cell decomposition Decomposes the space intro geometric areas and classifies them as either free or occupied by objects, then using a graph computes the trajectory by searching the graph. Recent successful methods use cells that change in shape and size over time, reflected on the uncertainty of the estimations [49]

2.4.2 Obstacle Avoidance

Obstacle avoidance focuses on a more local view of the trajectory, where paths are planned locally and giving particular importance to close rangefinding technologies. The most common approaches to obstacle avoidance are Bug algorithms [50], grid map occupancy avoidance [51] and potential fields [52].

2.5 Agricultural Robots

The analysis of existing solutions focused on the 15th and 16th editions of the *Field Robot Event*, an annual competition launched in 2003 by the University of Wageningen that puts robots to the test in a series of challenges (e.g. in-row navigation, mapping, weeding) that must be completed autonomously, with the caveat that GNSS is forbidden. Around 15 teams compete in the FRE every year and their solutions along with technical specifications of the robots are provided in the event's proceedings [53]. From these proceedings standout examples were chosen and described below and then general conclusions about the competing robots are presented. Finally, three standout commercially available solutions are explored.

2.5.1 Field Robot Event entries

Floribot uses two front differential drive wheels with two swivel wheels behind, allowing for turns on it's own axis. Highly scored (third place) using only a 2D laser rangefinder for navigation and

a camera for object recognition. It's cost-effectiveness is surprising and is also merited to the implemented navigation algorithm based on potential field, where the plants create a repulsive field and the robot follows a preset list of a instructions introduced via mobile app.

The Great Cornholio uses a robot-platform "Volksbot" by the Fraunhofer Institute which is powered by two 150W DC motors using a skid-steer system which, while providing higher payload capacity, is still an inefficient method of locomotion. This solution uses encoders, an IMU and a 2D laser scanner for navigation as well as a camera for object recognition. Their navigation strategy uses a varying size grid occupancy method where the grid is considered occupied if the laser "hits" a plant more times than the threshold value.

Finally, *Beteigeuze* stands out as a heavy-weight (literally, the robot weights 40 kg) in the competition. It brings two 2D laser rangefinders, encoders, an IMU, 2 stereo cameras (camera + structured light) and a fisheye camera. Earning first place in the 2017 edition, it uses a single central motor to power it's 4-Wheeldrive system. It also uses a central camera for the image processing and spatial information extraction for the "weeding" tasks.

Figure 2.6 shows the pictures of these three entries.

Figure 2.6: Floribot, The Great Cornholio and Beteigeuze respectively.

2.5.2 Conclusions

Here, the general conclusions of the agricultural robot's market survey are presented:

Software Tools ROS is the most popular framework for developing the robot's software. Another commonly used tool is MATLAB/Simulink and the robotics toolbox for developing algorithms then testing them inside MATLAB. Since MATLAB is based on C++, there is code portability directly to ROS which completes the development environment. Standout mentions in simulation software go to Gazebo and V-REP.

Localization Since in the competition the use of GNSS is banned, most entries use a 2D laser rangefinder for row detection fused with the odometry and IMU data for position estimation. But this is the setup which is used for the competition, since most agricultural robots cannot confidently rely on odometry and lasers only for navigation, most robots are also fitted with a DGNSS module that allows for precise localization within an open crop field.

Navigation Most of the implemented navigation algorithms were custom-made for this competition and examples are found based on potential fields, cell decomposition and mathematical methods such as using a least square estimator for regression fitting the line between the rows. Here, relevant solutions to the competition are found but real-world robust and reliable navigation is often more complex.

Motors and Traction The two most popular traction methods found were **Ackermann Steering** with it's simple control, higher weight capacity and high efficiency (no wheel slip when turning); and the simple **Differential Drive** with passive casters, providing fast closed turns but also a lower weight capacity and more susceptibility to rough terrain (rough terrain makes the passive casters constantly bounce). Typical total motor power for both these systems are 200W to 300W.

Computers and Processors Most robot's use a low-cost PC for high level control and algorithm processing (e.g. Raspberry Pi) combined with a micro-controller for low-level actuation and PID motor control. When the system requires visual algorithms, not only for object recognition but also odometry and extraction of 3D information, an Intel NUC, a desktop PC or other equivalent component is also installed.

2.5.3 Other solutions

Bosch offers a robot development platform that includes a robot designed for hard terrain, modular, with 4 independent wheel drive along with other software tools for simulation, logging,... This model uses a Differential GNSS technique with a base station and a receptor module called RTK-GNSS which is then fused with IMU data, enabling precision around $2cm$. *Bonirob* is popular in both private and academic projects [54], [55].

Other important mentions which are focused on the vineyard context is the VinBot [56], combining a 3D LIDAR and an RTK-GNSS fused with IMU and wheel odometry data and VineRobot [57]. Both these robots allow for the user to monitor in real-time the vineyard's quality parameters, improving both quality and optimizing management, and are closest to the proposed solution in this thesis.

Chapter 3

Approach and Developed Algorithms

The main challenges of this thesis rely on the restrictions imposed by the project which can be divided into two: environmental and cost restrictions. The steep-hill environment of river Douro's mountain vineyards pose three big challenges: Accuracy reduction from dead reckoning systems that naturally derive from the intended environment's harsh terrain; Reduced GNSS availability and accuracy due to the steepness of the Douro's wine-making regions; Difficult localization and navigation for safe route planning, avoiding damage to both the robot and the surrounding crops. Focusing on a low-cost system also brings many restrictions, but it's main drawback is the reduction of computational power which strongly limits the choice of both localization and navigation algorithms.

The proposed solution is here divided into two categories: Hardware, where the details of the components are exposed along with their electrical interfaces; And Software, where the focus is on the composition of the individual nodes and their responsibilities, then defining their inter-communication. This chapter will focus on the software development as well as the reasoning behind the chosen methods. Below (Fig. 3.1, is the general software architecture composed by blocks that correspond to each of the robot's main functions.

The localization and mapping system will use IMU, Odometry, GNSS and the LIDAR for solving the SLAM problem. The SLAM module will process the sensor data and feed it to the mission planning module. The mission planning module evaluates the tasks given and the current state of the robot in order to determine the robot's next action, which is then fed to the navigation module along with position information from the SLAM module. The navigation module will then output to the motor driver in order to act on the robot.

3.1 Simultaneous Localization and Mapping

SLAM is the most challenging task in the mobile robotics field and each solution should be adapted to the intended environment. According to the specifications of the project, the SLAM algorithm should be reliable and robust as the solution for the absolute localization, able to process real-time measurements, while not being too computationally expensive as a solution, for the considered

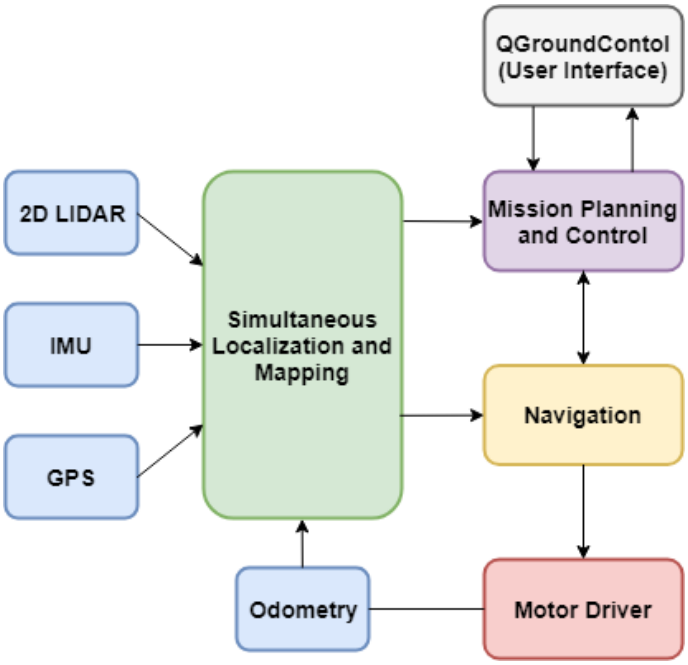

Figure 3.1: General Software Architecture.

hardware. This section will present a comparison between two state-of-the-art algorithms, it's choice, integration in the system, calibration, and developed modules along with obtained maps containing trajectory estimation.

3.1.1 SLAM methods analysis and comparison

SLAM methods for this project must fulfill two main criteria: Low computational cost and robustness when mapping and localizing in an unstructured environment. Two popular SLAM algorithms were identified that meet (or can be adapted) to fit this criteria. First, the massively popular Hector SLAM [58] developed in Dramstadt University in 2011 for the RoboCup Rescue competition, with the goal of providing sufficiently accurate mapping and self-localization while minimizing computational costs. It combines a 2D LIDAR and and IMU for full 2D motion estimation. Alternatively is Google's Cartographer [59] introduced in 2016. Cartographer's strong suits are the facilitation of integration of different sensors using only raw data (e.g. GNSS, IMU, 2D/3D LIDAR, Kinect, etc.) and its robustness for large-scale maps using loop-closure between pose and map. Cartographer's SLAM can actually be divided into two, first the local SLAM, which frequently publishes sub-maps of the surrounding area, and global SLAM, whose job is to tie all its sub-maps consistently by periodically providing map constraints from the LIDAR scan matching and the odometry.

The logs used were provided by INESC TEC and extracted from the AGROBv16 in April 20 2018, in the vineyards of the University of Trás-os-Montes and Alto Douro, for comparisons in the intended environment. This data set is available at `http://vcriis01.inesctec.pt`, named *Data acquired by Agrob V16 @UTAD (20/04/2018)*, containing all the registered topics in the robot

at that time: Controller commands, video, pose and localization information, maps, and the data output of the sensors. The information used in these experiments was the IMU raw data, the laser scanner raw data and the filtered odometry.

3.1.1.1 Map comparison

All the maps extracted use the same thresholds for free and occupied cells in the published occupancy grid in order for a reasonable comparison to be established. In this case free cells (in white) have an occupancy probability lesser than 35% while occupied cells' probability is larger than 65%.

Here three procedures are compared:

- Cartographer using a 2D LIDAR, IMU and wheel odometry;
- Cartographer using a 2D LIDAR and IMU; and,
- Hector SLAM using a 2D LIDAR and provided odometry from wheel odometry and IMU.

As can be seen in figures 3.2 and 3.3, both of Cartographer's maps are generally more immune to noise and low-height obstacles, attributing a lower probability to objects that are not repeatedly and consistently matched which as a consequence produces maps with finer detail of the environment's strongest features (in this case the vineyards' trunks). Considering scan matching depends on the pose estimation of the robot, the map produced by Cartographer when running with all sensors is more consistent with the environment due to the stability of the pose estimation when using wheel odometry. Second, although not observable in this map since it was only a short forward and back path in a larger vineyard, Cartographer's periodic map loop closing allows for large and non-collinear robot paths while keeping the robustness of both the map and its pose estimation.

Hector map is more consistent when mapping features at a longer distance as can be seen in the map's extremes, but since this project will be using a rangefinder with only a $5.6m$ range, it doesn't present as an advantage.

3.1.1.2 Trajectory and pose estimation comparison

Since there is no ground truth provided, only a qualitative analysis is possible. Pose error comparisons can be found in [60]. Both Cartographer and Hector provided a similar and consistent global pose estimation. In figure 3.4, Cartographer's pose estimation without wheel odometry becomes highly susceptible to IMU noise and vibrations originating from the rough terrain.

3.1.1.3 Computational cost

In tables 3.1 and 3.2, the average latency between the laser scans and the output of the estimated pose is presented for the three methods considered, firstly in a modest machine, the laptop MSI CX-62 6QD with an Intel i5-6300, and secondly in the Raspberry Pi 3B which will be used for the

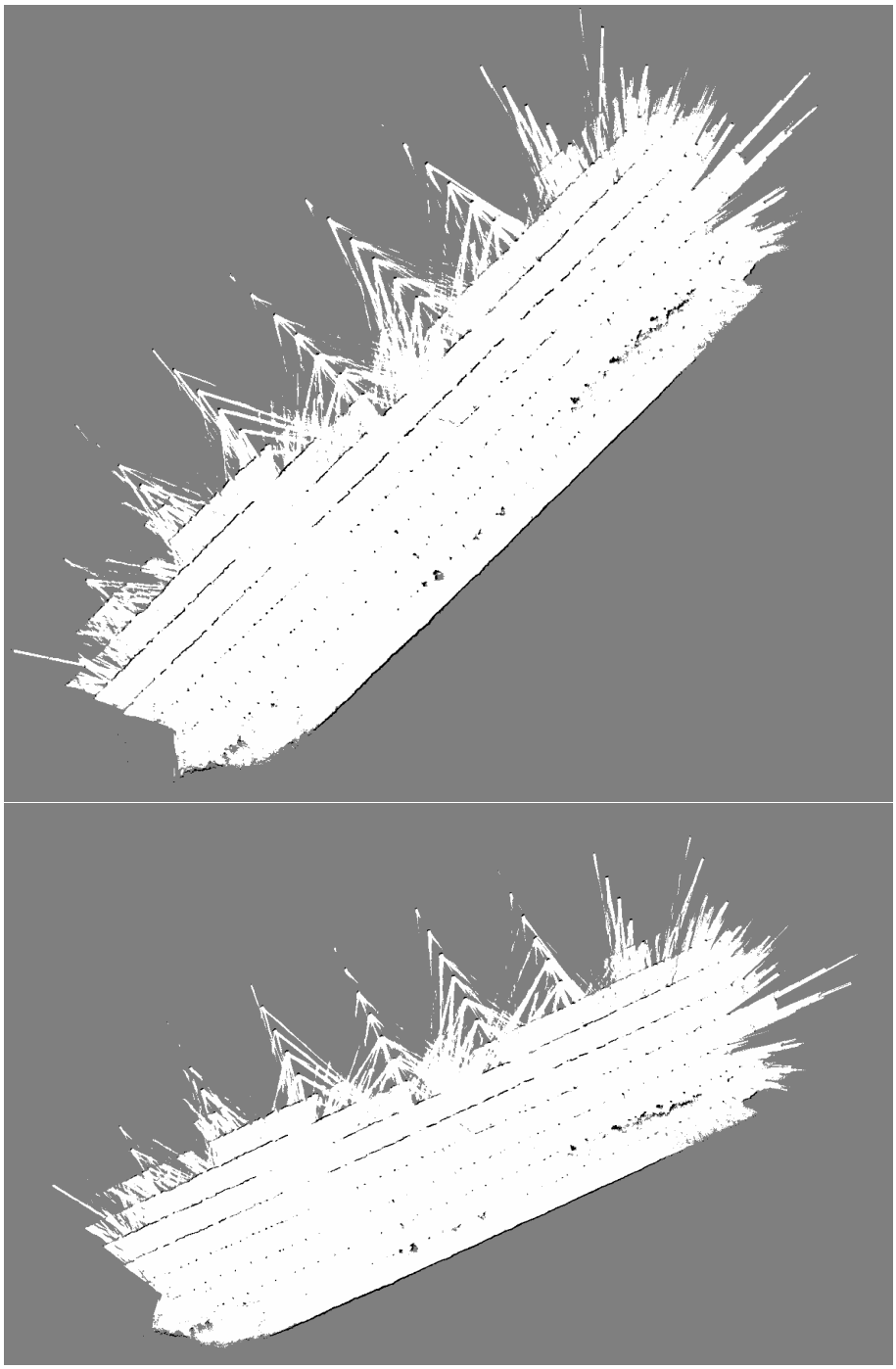

Figure 3.2: Cartographer map with wheel odometry (Top) and Cartographer map without wheel odometry (Bottom)

project. The difference is negligible in modest machines between Hector and Cartographer, however in a Raspberry Pi this difference should be noticeable. But unexpectedly, there is a large increase in the Hector SLAM algorithm's latency, probably due to its implementation, which bottlenecks when using the Raspberry Pi's resources. Cartographer then becomes the obvious choice in terms of computational cost, especially since Cartographer's performance can be tuned to the hardware

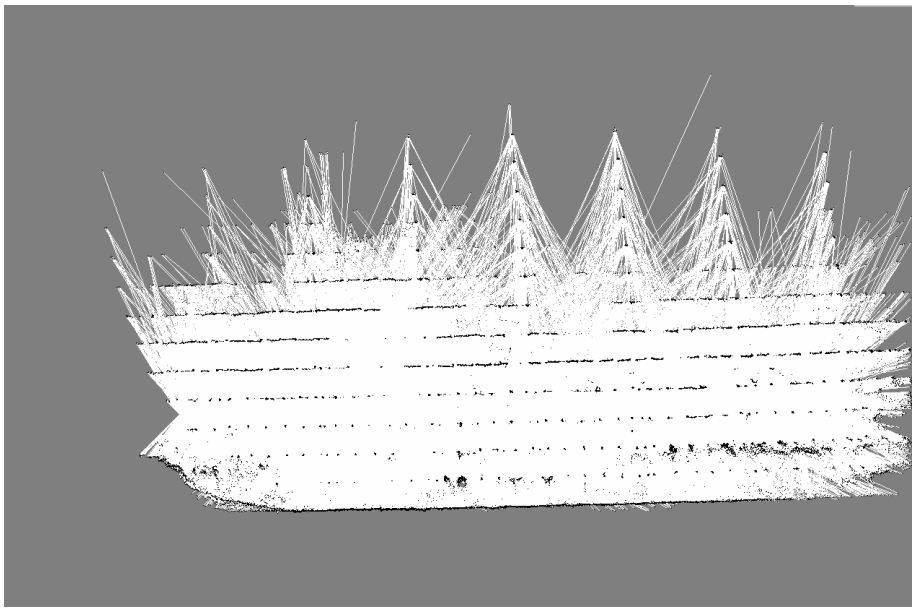

Figure 3.3: Hector SLAM map

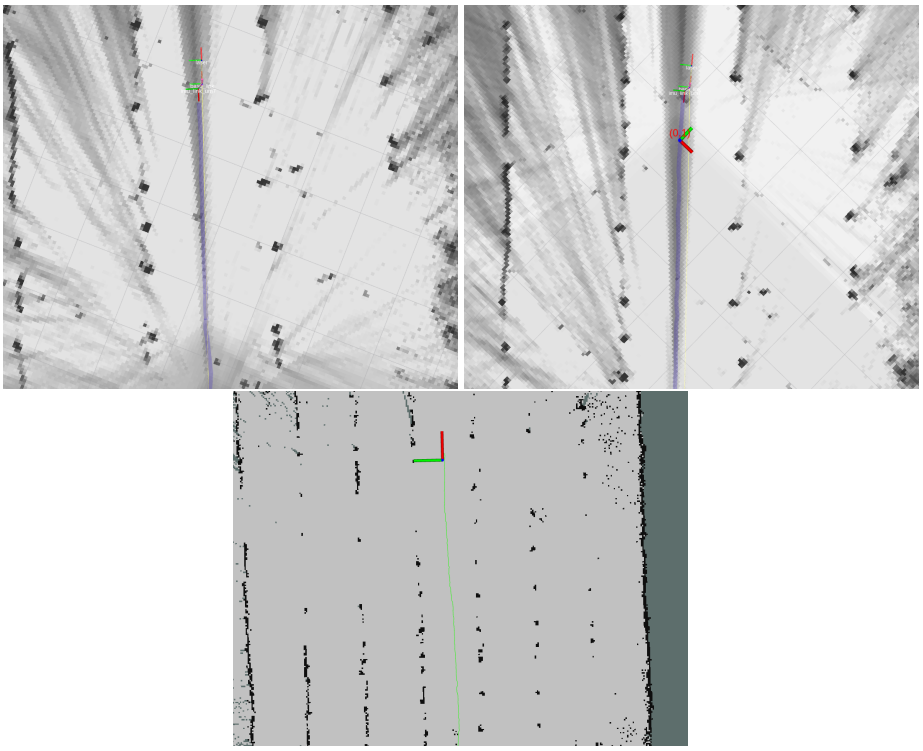

Figure 3.4: Trajectory estimation in Cartographer with wheel odometry (top left), without wheel odometry (top right) and Hector SLAM (bottom)

by following the walkthrough provided in the documentation.

SLAM Method	Average Latency
Cartographer w/ IMU, 2D LIDAR and WO	58 ms
Cartographer w/IMU and 2D LIDAR	46 ms
Hector SLAM	40 ms

Table 3.1: Average latency between laser scans and pose output running on a laptop.

SLAM Method	Average Latency
Cartographer w/ IMU, 2D LIDAR and WO	124 ms
Cartographer w/IMU and 2D LIDAR	61 ms
Hector SLAM	217 ms

Table 3.2: Average latency between laser scans and pose output running on a Raspberry Pi 3B.

3.1.1.4 Conclusions

Hector SLAM was designed and is appropriate for small scale environments where large loops of the robot's SLAM do not have to be closed, but since the aim of the robot is producing crop quality maps on large semi-structured environments, closing large map loops becomes a requirement that only Cartographer can verify. Since Cartographer's package provides more functionality at the cost of a slightly larger computational power and a more complex configuration and tuning, this trade-off becomes especially beneficial due to the complexity of the environment.

3.1.2 SLAM Configuration and Tuning

Cartographer is a complex system and tuning it is a complicated task which requires some theoretical knowledge of the system and hands-on experience in judging map quality and how it's affected by the parameters. Luckily, the developers provide documentation on how to approach the tuning problem, with a practical case for better comprehension.

As previously mentioned, the Cartographer SLAM can be divided into local and global, where the local part runs in real-time and keeps publishing sub-maps and the global part runs in the background and is responsible for tying the sub-maps together. Therefore, tuning is required in both these components, although the biggest focus will be on the quality of the local, real-time SLAM and only after will the global SLAM be configured for low latency (between scans and sub-map publishing).

Following, is the list of the changed parameters with an explanation of their influence in the system:

- **TRAJECTORY_BUILDER_2D.num_accumulated_range_data** - Distances are measured over a certain period of time, and the messages are compiled into batches for processing and scan-matching. The higher the number of accumulated scans, the higher the quality

of the matching but, this also brings an increase in computational demand. This parameter was tuned by observation of the matched points and by guaranteeing that the system uses the entirety of the environment for feature matching while still performing consistently in real-time.

- **TRAJECTORY_BUILDER_2D.ceres_scan_matcher.translation_weight** - Defines the cost of deviating from a prior measure, position-wise. The higher the weight, the higher the score the scan matching has to generate in order for another position to be accepted. This parameter can be calibrated by observing any "drag" in features perpendicular to the direction of the robot.

- **TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight** - Defines the cost of deviating from a prior pose, specifically rotation. As before, the higher the weight, the higher the score the scan matching has to generate in order for another pose to be accepted. This parameter can be calibrated by observing incorrect angles when performing curves or rotations (e.g. observing a right angled wall).

- **TRAJECTORY_BUILDER_2D.min_range** and **.max_range** - In order to eliminate wrong measures that escape the range indicated by the manufacturer's datasheet, the limits for the scan matcher are set here. The distances are defined in meters.

- **TRAJECTORY_BUILDER_2D.use_imu_data** - Cartographer can perform using only a 2D LIDAR and an IMU for a gravity reference, but in this case the full IMU data is utilized for pose estimation. This also allows for a better initial pose estimation, which strongly facilitates the scan-matching process.

- **TRAJECTORY_BUILDER_2D.use_nav_sat** - This enables the GNSS data to be used in the global SLAM problem. The GNSS only aids in the global SLAM since it has a strongly reduced accuracy.

- **TRAJECTORY_BUILDER_2D.voxel_filter_size** - A voxel filter is applied in order to subsample a group of points into a cube of set size (in meters), which facilitates the scan-matching algorithm. The size of the filter is here increased trading a more dense data representation for computational demand.

- **POSE_GRAPH.optimize_every_n_nodes** - Similar to the first parameter, but considers batches of trajectory nodes for building constraints for the global SLAM problem. Decreasing this parameter lowers the complexity of the operation but decreases global SLAM solving capacity. Since the system is intended to perform with a finely tuned local SLAM and small global SLAM correction, this global SLAM will be limited to gain computational resources.

- **POSE_GRAPH.optimization_problem.local_slam_pose_translation_weight** - The larger the pose graph weight is, the larger impact it has in the output of the global SLAM. This weight is attached to the output of the local SLAM, specifically the translation.

- **POSE_GRAPH.optimization_problem.local_slam_pose_rotation_weight** - Similar to the previous parameter but for the rotation of the robot.
- **POSE_GRAPH.optimization_problem.odometry_translation_weight** - This weight is tied to the same global SLAM optimization problem. It is attached to the global output of the odometry, specifically the translation
- **POSE_GRAPH.optimization_problem.odometry_rotation_weight** - Similar to the previous parameter but for the rotation of the robot.
- **POSE_GRAPH.constraint_builder.min_score** - Represents the minimum confidence in the global constraint generation for the state to be updated. The higher the value of this parameter, the higher the quality of the constraints between submaps, but a high value can slow the output frequency of map corrections.
- **POSE_GRAPH.optimization_problem.huber_scale** - The influence of outlier measures is handled by a error regression known as a Huber loss function, which is tuned with a Huber scale value, The higher the value in the Huber scale, the higher the influence of the outliers.

The distinction in weights between the global and local SLAM is useful for adapting the values to the confidence in different ways. In this case specifically, since the odometry wasn't great, it was tuned to have a smaller impact in the local SLAM, only having a high impact when there are no LIDAR readings. On the other hand, since the LIDAR has a lower scan publishing frequency than optimum, it accumulates error and drifts when at higher speeds. Here, the odometry should have a higher impact on the output of the global SLAM, providing it's quality is sufficient, guaranteeing that the distances between different zones in the map are consistent when fused.

The maps obtained in figure 3.5, using only LIDAR and IMU, represent the before and after of the tuning process. After tuning, most of the drag is corrected and the position estimation is precise which results in a smooth trajectory. There is still a slight deviation in the angle estimation as visible in both the alignment between the corridor and the rest of the floor and the center square to the right, but it is expected that with the integration with odometry this problem will be corrected.

3.1.3 Integration of GNSS data

Since the intended operation environment is large, an effort was made into improving the quality of this task by introducing GNSS data into the problem. Cartographer uses scan and sub-map matching as well as other sensor data to guarantee it provides a consistent global solution. When testing the SLAM algorithm it was noticed that although there is an integration of GNSS data in 3D SLAM mode, this sensor was not fused into the 2D SLAM problem.

Therefore, after further analysis of the problem and the 3D implementation developed by *Lyft*, the generation of constraints based on **sensor_msgs/NavSatFix** was adapted to the 2D problem and height (z-axis) information was removed from this problem, adding the generated constraints

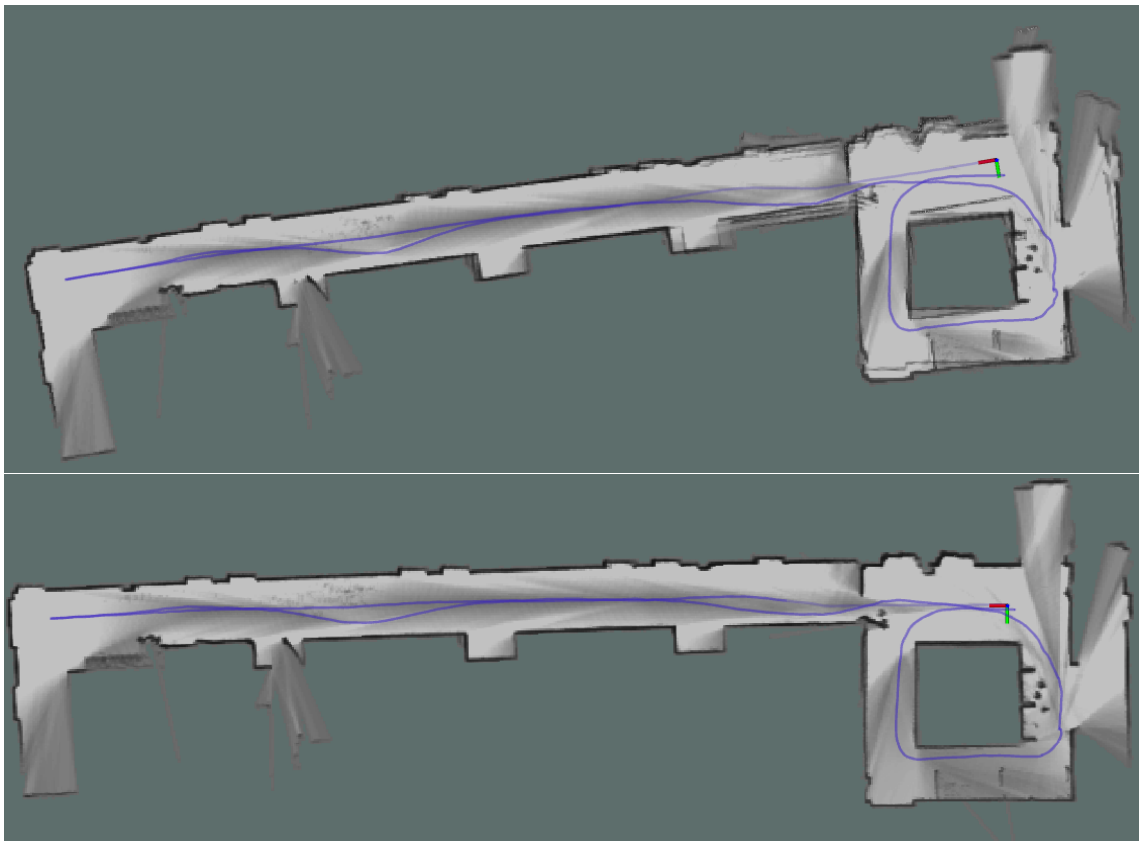

Figure 3.5: On the top, the not tuned result. On the bottom is the tuned version. The blue line represents the estimated trajectory of the robot.

to the global SLAM problem. The adapted part corresponds to the red box in figure 3.6. The pull request is currently awaiting review and is available at [61].

Since Cartographer limits the sub-map publishing frequency to the frequency of the lowest frequency sensor to simplify generating consistent constraints, firmware changes to the GNSS module were done to increase the publishing frequency to $10Hz$, putting it on par with the publishing frequency of the laser scanner.

3.2 Motor Configuration and Control

This section details the motor configuration along with the developed software for interfacing with the motor via CANopen. The goal of this phase was to obtain an accessible interface for the motor and simultaneously translate velocity commands that are published by the system into target motor velocities, while maintaining correct motor operation. The procedure for configuration of the motor, the details of the developed ROS node, and the used odometry model are presented below.

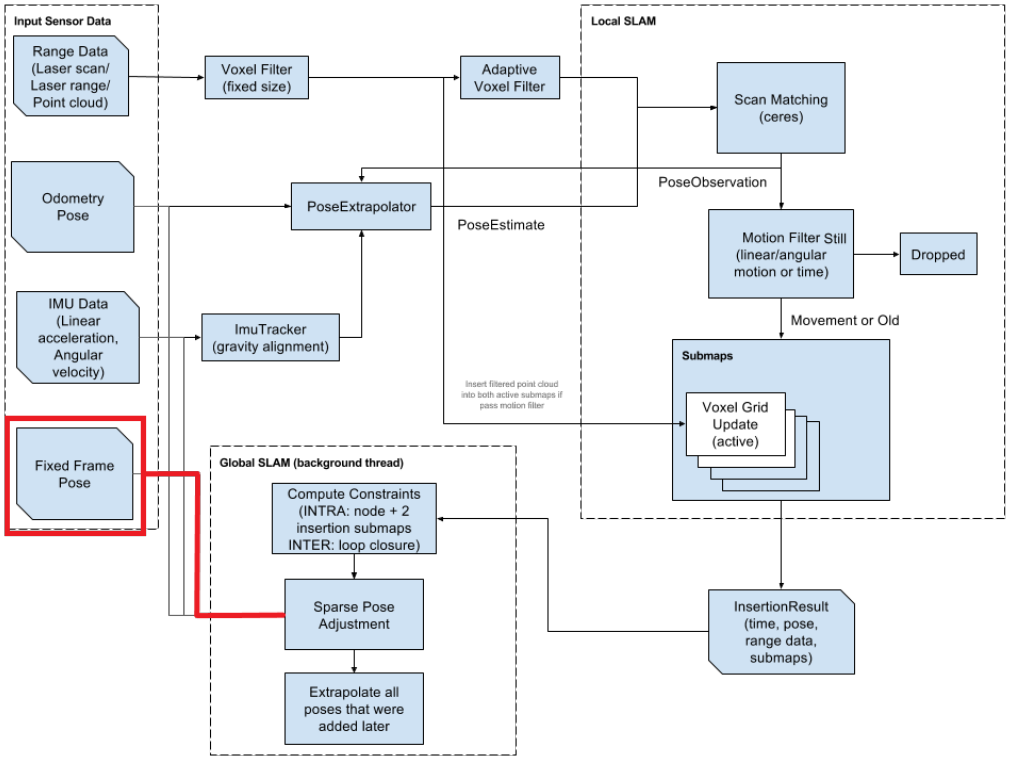

Figure 3.6: System overview of the Cartographer SLAM-

3.2.1 Controller Configuration

The configuration was done according to the manufacturer's instructions [62] using the software provided. First, the motor parameters have to be inserted according to the datasheet into the CANopen object dictionary along with the maximum current limited to the capabilities of the power supply. Following, an auto-configuration is performed in order to extract the motor's, encoder's and brake's intrinsic parameters for the calibration of the control parameters. Finally, a test was performed by varying the target velocity, switching throughout power state machine as well as engaging and disengaging the brake.

3.2.2 Motor Driver ROS Node

The control of the motor via CANopen is done using what is defined in the documentation as a *Statusword* and a *Controlword*, one is used to read the actual state of the machine and another to request changes between the states. The power state machine (Fig. 3.7) is defined according to the CANopen standard 402 which specifies functional behaviour for motors, also including operation modes and configuration parameters. The developed ROS node for driving the motors implements an interface that allows interaction according to the state machine, specifically the five states that make up the center (Not ready to switch on, Switched on disabled, Ready to switch on, Switched on and Operation enabled).

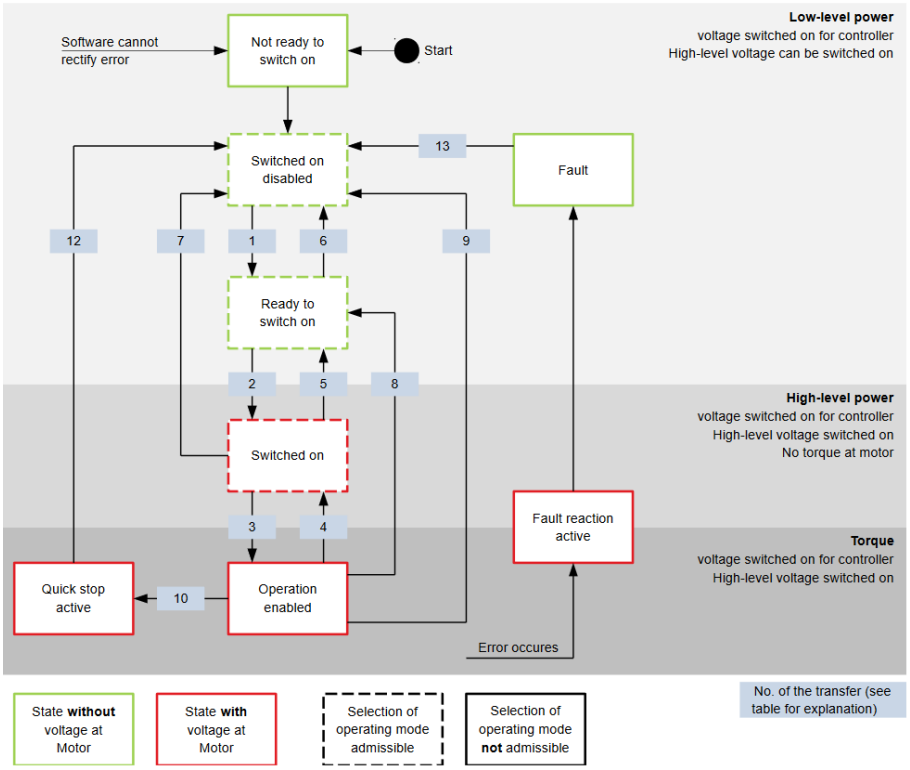

Figure 3.7: Motor power state machine according to the CAN in Automation standard.

The node was developed in C++, with the exception of the CAN libraries which are in C. Below is figure 3.8 containing the class diagram of the node.

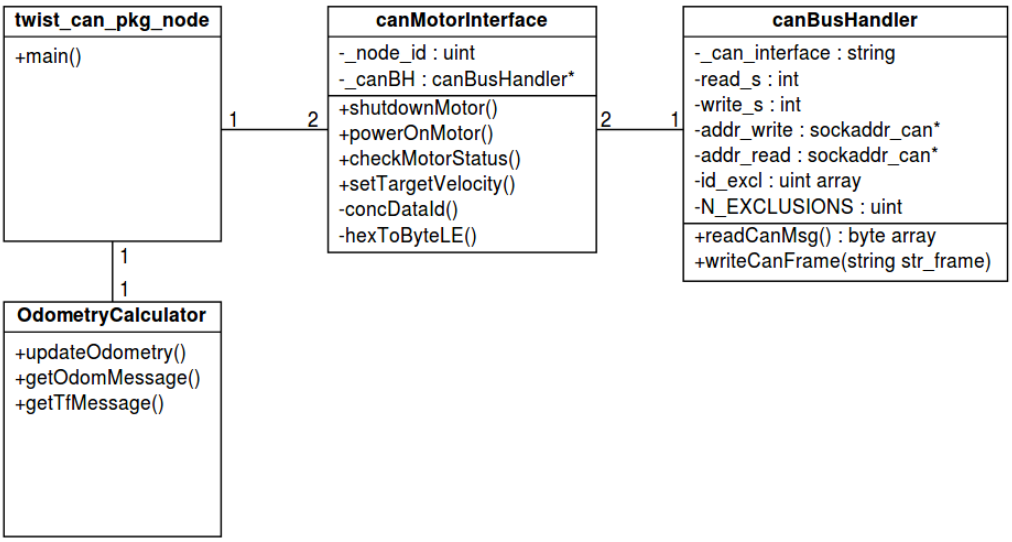

Figure 3.8: Class Diagram of the twist_can_pkg node.

For each class it's responsibilities and the technologies used are defined:

- **twist_can_pkg_node**: This class runs the main code, containing a loop that performs two

tasks: Publish the odometry topic and translate velocity commands into motor instructions. It is also responsible for reading the encoders, then passing the data to calculate the odometry, and guaranteeing correct motor initialization and shutdown. The motors automatically shut-off in case of critical failure in the ROS system or kill call (such as the bash command *ctrl-C*).

- **canMotorInterface**: This class works as an abstraction from the message building and simplifies motor interaction using simple methods, such as power on/off and set velocity. There is an interface for each of the two motors separately, where the ID of the motor and the CAN bus handler must be passed to the constructor. This class uses pre-defined strings which contain the CAN messages necessary for this application, be it changing states in the power state machine, reading encoder value or setting the motor's target velocity. When reading CAN messages, the messages are either crossed with pre-determined bit masks for verification or parsed into values which will be used for other functions in the node.

- **canBusHandler**: Responsible for opening/closing the read/write sockets, as well as reading and writing from the socket. Since the code instructions are faster than writing to the socket, the handler has protections against reading messages that it writes itself using the processors own CAN-ID. In order to avoid excessive resource allocation, there are two instances of the motor interfaces (one for each motor) but only one bus handler, which maintains two open sockets for reading and writing. The writing implementation is done by parsing a string into a CAN frame, which allows to set up the known static messages in the motor interface and abstract from the protocol itself.

- **OdometryCalculator**: This class uses reading from the motor encoder along with their timestamps in order to estimate the robot's position. The main cycle then only uses getters for fetching the messages and publishing.

Besides the ROS libraries used to create the node and standardize messages between nodes, the open source CAN library **SocketCAN**, developed by Volkswagen Research for the Linux kernel, which extends the classic socket API, was also used for interfacing between the Raspberry Pi and the motor controller. The code was based on the provided examples with adaptations to the intended applications. The node's source code including the used libraries is available in my personal repository [63].

3.2.3 Odometry Model

Here is presented the model used for odometry of the differential drive robot. In this equation, d_1 and d_2 represent the distance each wheel moves between each time interval ΔT and finally b indicates the distance between wheels. Note in equation 3.1 that it is used a discretization model with advanced differences. Here d_1 and d_2 represent the distance traveled by each wheel, b

represents the wheelbase distance.

$$\begin{aligned}
&\Delta d(t) = \frac{d_1(t) + d_2(t)}{2} \qquad \Delta\theta(t) = \frac{d_1(t) - d_2(t)}{b} \\
&\Delta x = \Delta d(t) * cos(\theta(t) + \frac{\Delta\theta(t)}{2}) \\
&\Delta y = \Delta d(t) * sin(\theta(t) + \frac{\Delta\theta(t)}{2})
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
&v_x(t) = \frac{\Delta x(t)}{\Delta T} \quad v_y(t) = \frac{\Delta y(t)}{\Delta T} \quad v_\theta(t) = \frac{\Delta\theta(t)}{\Delta T} \\
&x(t) = x(t-1) + \Delta x \quad y(t) = y(t-1) + \Delta y \quad \theta(t) = \theta(t-1) + \Delta\theta
\end{aligned}$$

The results are then published into the standard ROS odometry message **nav_msgs/Odometry** which represents an estimate of the position and velocity of the robot in a specified coordinate frame. The odometry is then used by the SLAM module for pose estimation.

3.2.4 Odometry Calibration

In order to attenuate error introduced in the odometry by systematic errors, such as incorrect hand made measurements or deviations from theoretical in the structure of the vehicle, the previously mentioned procedure known as UMBMARK was done. This method was chosen for it's capability to deal with systematic and repeatable errors, and it's simplicity.

UMBMARK specifically targets two problems, unequal wheel diameters and uncertainty about the wheelbase (defined as the distance between the contact points of both wheels), it's output being an estimate of the wheelbase's value and the ratio between the difference of wheel diameters. The procedure consists of measuring the robot's final pose after it performs a square of fixed size, repeated for 5 times, twice in the clockwise and counter-clockwise direction. Figure 3.9 shows the procedure using a wall as a reference for the measures and the resulting points.

3.3 Sensor Acquisition

Open-source ROS packages were used in order to transform the raw output from the sensors and convert them into standard ROS messages. The following packages were used for each sensors: bosch_imu_driver [64] which outputs sensor_msgs/IMU messages; nmea_navsat_driver [65] which outputs sensor_msgs/NavSatFix; urg_node which outputs sensor_msgs/LaserScan [66]; and finally twist_can_pkg node which outputs both the tf/tfMessage and the nav_msgs/Odometry.

3.4 Mission

The system considers a mission to be a collection of sequential waypoints, where the autonomous vehicle must move towards and reach within a defined acceptance radius, repeating the process

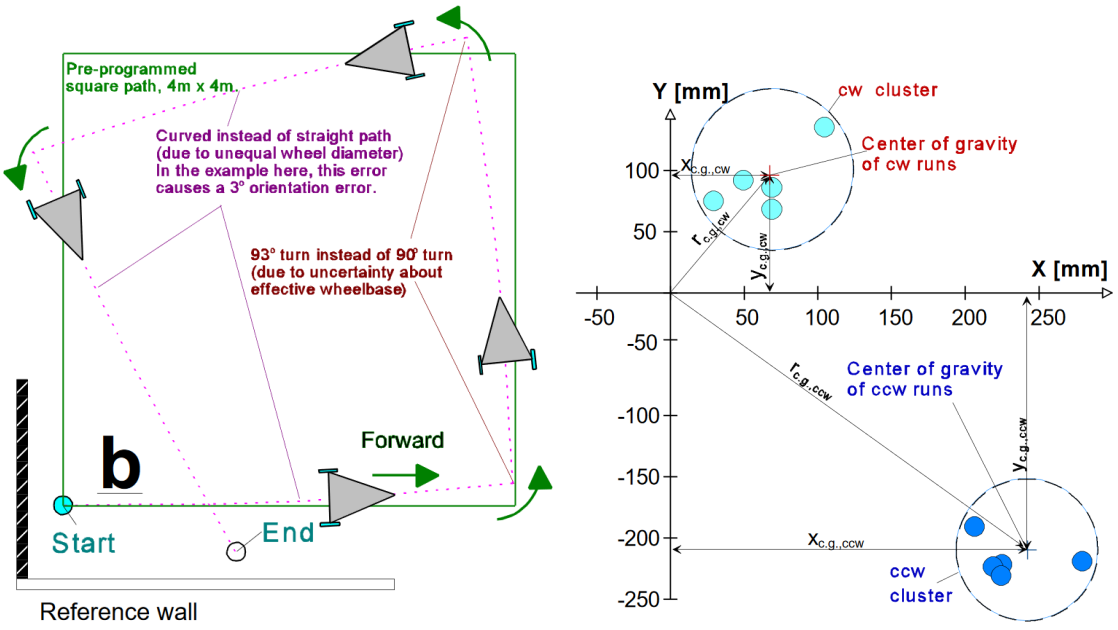

Figure 3.9: Illustration of the UMBMARK procedure and the resulting gravity centers of the errors. [14]

until the mission is done. In this case, a robot can only have one mission at a time.

Therefore, the goal is to develop a node which is capable of managing the mission, maintain connection with the mission control station, and translate the global waypoints into points within the robot's coordinate frame. To accomplish this, the system first uses the GNSS and the IMU's magnetometer to determine a global starting latitude, longitude, altitude and yaw. Afterwards, the SLAM's location output is compareed with latitude/longitude of the goal in the local frame, to calculate the distance and the angular error, which will the be fed to the navigation algorithm. Although the GNSS could be used alone to execute the missions, it's unreliability makes using the localization output the preferable option, although there is a possibility of a drift in the position.

The user interface will be done via the open-source application QGroundControl (QGC) [67].

3.4.1 Mission Control Node

This node was developed in C++, with the exception of the MAVLINK libraries which are in C. Below, is figure 3.10 containing the class diagram of the node. This node is also available in my personal GitHub repository [68].

For each class it's responsibilities and technologies are defined:

- **qgc_interface_node**: On startup, this node runs the initial global estimation location and pose using the IMU and GNSS data. Following, it runs the state machine in figure 3.11 which manages the connection and passes the trajectory and pose data to the user interface, manages the mission state and publishes the current error from the robot's pose and location to the goal's location.

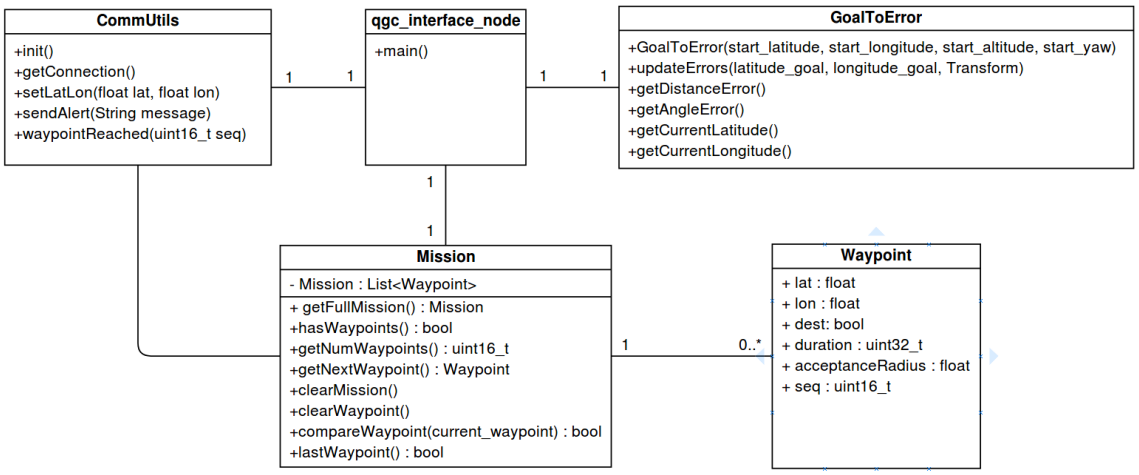

Figure 3.10: Class diagram of the qgc_interface node.

- **CommUtils**: Must manage the connection with QGC, including initial configuration and trading heartbeats which are used to establish it. Secondly, it must translate the incoming MAVLINK packages into a mission list and respective details. This includes adding and removing missions, clearing the mission's list and monitoring mission status. It also guarantees the vehicle and control station missions are synchronized, dumping incorrect information if necessary. Finally, it should forward to QGC some vehicle data, such as GNSS data and the vehicle's status.

- **Mission**: Container class with a list of waypoints and the methods necessary to interact with, such as adding/removing missions or verifying if a mission is complete.

- **Waypoint**: Each waypoint contains it's latitude and longitude as well as an identification integer which represents it's sequence in the mission.

- **GoalToError**: The main purpose of this class is to output the distance and angle errors from the vehicle to the goal. To do this, when instantiating this class, the initial global pose must be inserted including latitude, longitude, altitude and it's yaw (in East, North, Up coordinates). Afterwards, it uses the transform of the robot's base link relative to the local map's origin (coinciding with the robot's starting pose) to both define a goal in XY coordinates and update the robot's current global position (latitude and longitude). Finally, it calculates the errors in the local map frame.

3.4.2 Conversion between coordinates

To convert a position from geodetic coordinates to the local XY frame, allowing for the following utilization of euclidean geometry, a ROS package was used containing an interface which allows to transform to Universal Transverse Mercator (UTM) coordinates. This system divides the earth into grid zones (Fig. 3.12) and approximates the curve including altitude data, reducing distance

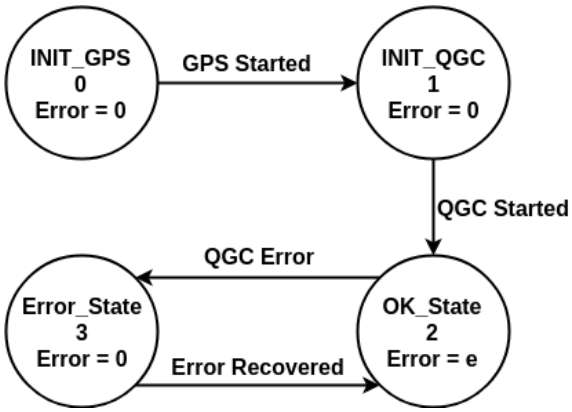

Figure 3.11: State machine of the node that controls the mission. *e* represents the error to the next goal.

and angle errors introduced by typical approximations of the earth to a sphere or an ellipsoid. The package used is *geodesy* [69] and is included in the official ROS package repository.

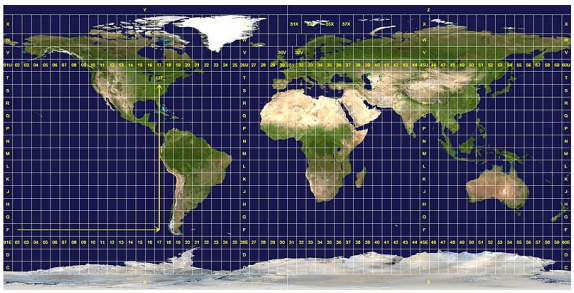

Figure 3.12: The longitude and latitude zones in the Universal Transverse Mercator system..

As mentioned, the GNSS is used to estimate the latitude, longitude and altitude, by initially acquiring 20 measures and averaging. From there, the position is then estimated using the output of the SLAM algorithm. The initial orientation is extracted from the quaternion output of the IMU's magnetometer:

$$\theta_{start} = atan2(2(qw * qz + qx * qy), 1 + 2(qy * qy + qz * qz)) \tag{3.2}$$

After obtaining the initial latitude, longitude, altitude and initial orientation the geodesy interface is used to convert to UTM.

The resulting altitude, band and zone is stored for the inverted conversion between local XY coordinates to geodetic. To convert between local coordinates and UTM, a translation to convert the XY coordinates into UTM is aplied along with a rotation to compensate for the initial global yaw. x_{start}, y_{start} and θ_{start} represent the initial position of the robot.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_{start} \\ y_{start} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \cdot \begin{bmatrix} cos(-\theta_{start}) & -sin(-\theta_{start}) \\ sin(-\theta_{start}) & cos(-\theta_{start}) \end{bmatrix} \tag{3.3}$$

The result is then converted to geodetic coordinates using the package. Using these techniques the same can be inversely applied to the coordinates of the goal.

3.4.3 QGroundControl and the MAVLINK Protocol

QGroundControl is an open-source ground control station for autonomous aerial vehicles which support communication via the MAVLINK protocol. It's goal is to provide a user interface which support mission planning, a map displaying vehicle position, trajectory, waypoints and vehicle instruments (sensors).

The MAVLINK implementation uses the socket API over UDP to communicate and provides a package management API to generate the protocol's messages. To take advantage of this tool, an application of the basic functions of the protocol was built, including initial configurations, alerts, position tracking and most important, the mission protocol which allows for the upload/download of missions and tracking their progress. The implementation is achieved in the class **CommUtils** of the interface package. Three sub-applications over the MAVLINK protocol were implemented, the connection, the mission protocol (Fig. 3.13), and the trajectory and pose data representation.

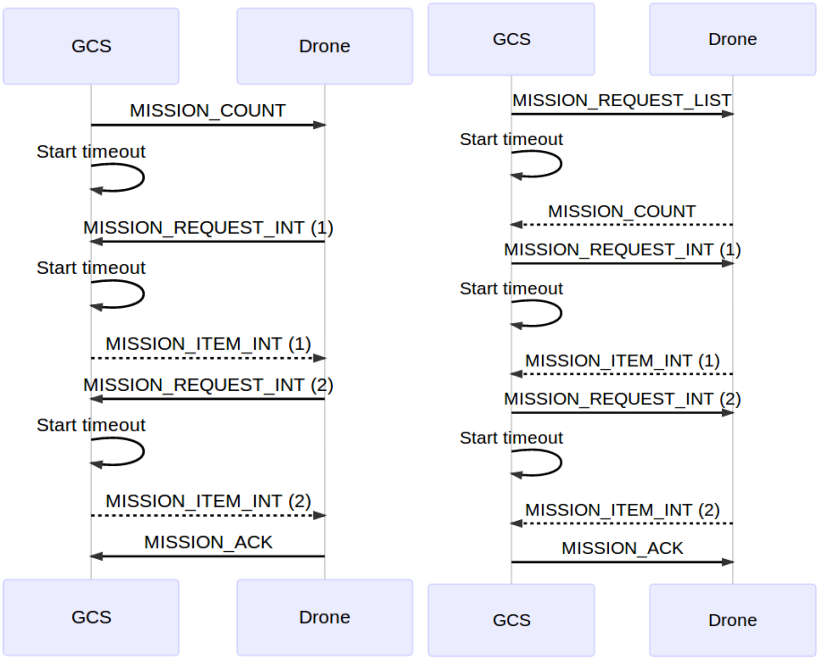

Figure 3.13: Upload and Download of the MAVLINK protocol [70].

3.5 Navigation

The implemented navigation algorithm matches the idea of the user interface, as presented in the state machine of the system in figure 3.14. The typical process when following would be to first rotate towards the goal within an angular margin, then begin moving towards the goal in a straight line while applying minor corrections to the angular error and finally, when entering within a set distance of the goal, approach with a slower speed in order to de-accelerate and stop on the goal. The idea is to create a mission consisting of a sequence of waypoints that the robot would follow. This node outputs linear and angular velocities commands with values between -1 and 1, which are then used by the motor driver node.

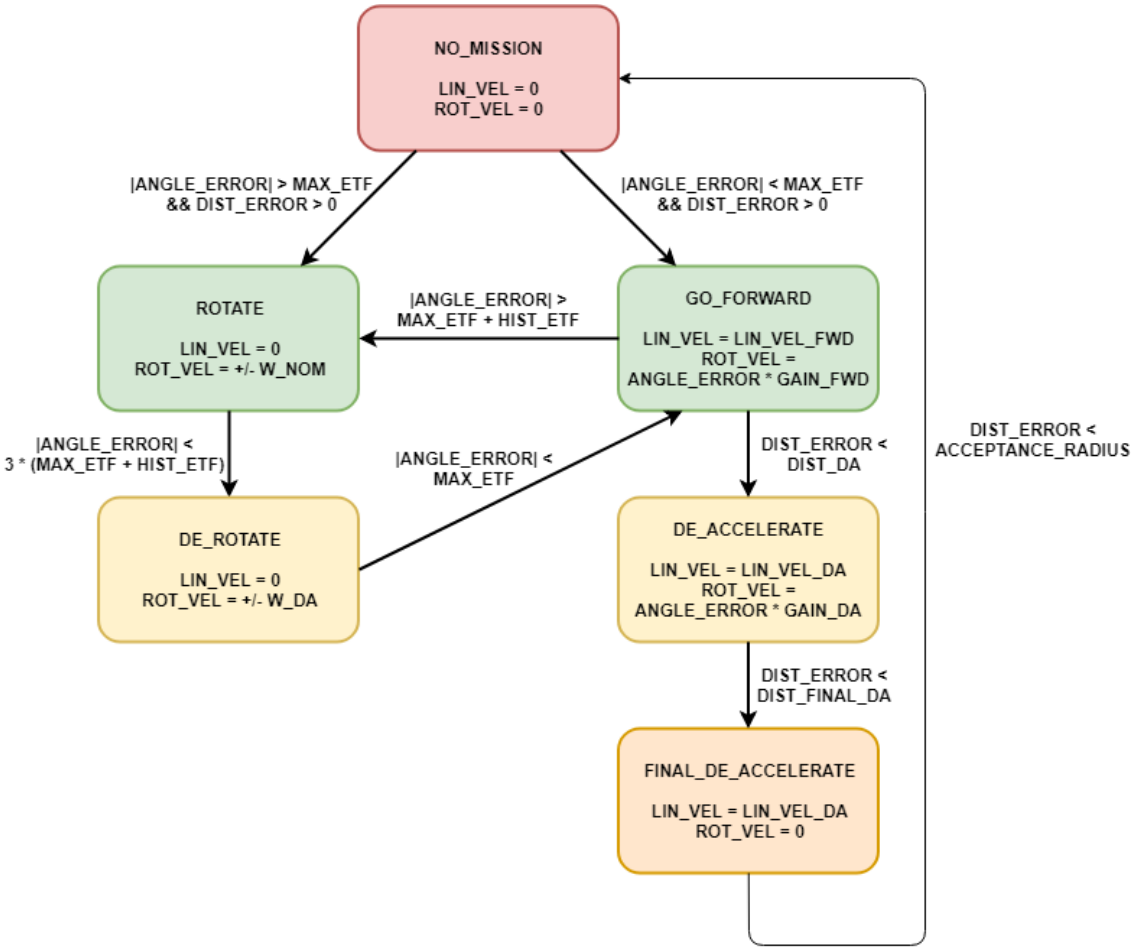

Figure 3.14: State machine of the navigation node.

Below are the considered parameters and what they represent:

- **ANGLE_ERROR** - Angular error to the goal.
- **DIST_ERROR** - Distance to the goal.
- **W_NOM** - Normal rotational velocity.

- **W_DA** - Low rotational velocity.
- **LIN_VEL_FWD** - Normal forward velocity.
- **LIN_VEL_DA** - Low forward velocity.
- **GAIN_FWD** - Proportional gain to the angular error in the GO_FORWARD state.
- **GAIN_DA** - Proportional gain to the angular error in the DE_ACCELERATE state.
- **DIST_DA** - Distance to goal which triggers the DE_ACCELERATE state.
- **DIST_FINAL_DA** - Distance to goal which triggers the FINAL_DE_ACCELERATE state.

3.6 Final ROS Architecture

Figure 3.15 shows the result of the ROS *rqt_graph* command, creating a map of all the nodes in the system and what topics they are publishing/subscribing. In this diagram topic */tf* is localization and pose data, partial or full; Topic */errors* is the distance and angular error to the next goal; Topic */cmd_vel* is the robot's output velocity; Topic */odom* contains the resulting odometry; Topic */scan* contains the LIDAR readings; and topic */imu_bosch/data* contains all data read from the IMU.

The main cycle of this program is composed by the */qgc_node*, the */error_to_twist_node*, the */motor_node* and the *cartographer_node*. First, */cartographer_node* uses the IMU, LIDAR and odometry data which is used by the *qgc_node*. This node then uses it's recorded initial global position, from the GNSS and IMU, it's current local position and it's next goal to compute the distance and angular error to the next goal. Following, the navigation node *error_to_twist_node* converts these errors into velocity commands which are then executed by the motors. After the system moves, the odometry position is updated and published for the SLAM algorithm, completing the cycle.

Unfortunately, no GNSS node is available due to hardware malfunction. One node was cropped out of this image for proper fitting, but it is the map generating node which also takes the output from the */cartographer_node* to generate the occupancy grid.

3.7 Chapter Summary

First, Hector and Cartographer's performance are compared, where Cartographer beat Hector in both map quality and processing in cost-effective hardware, making Cartographer the choice for this project. Afterwards, the software necessary to execute the proposed solution is detailed: Sensor Acquisition, Localization, Mapping, Motor Driver, Navigation and Mission. The packages that were developed or chosen fulfill their functions and concluding, the minimum required packages for making the robotic platform usable were achieved.

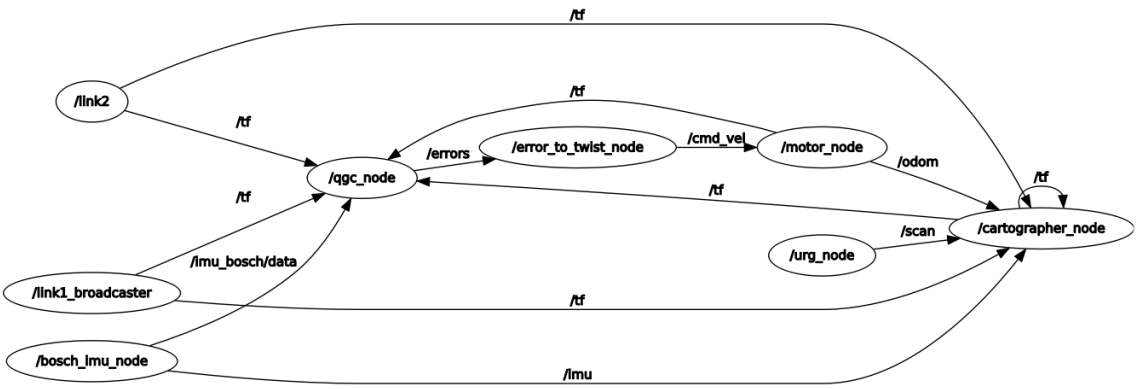

Figure 3.15: Result of the ROS command *rqt_graph* which shows the nodes and topics they exchange.

Chapter 4

Hardware and Components

In this chapter, the custom designed hardware architecture which implements the software functions explained in chapter 3 is presented. The goal is to develop a cost-effective architecture which is able to perform the necessary tasks with a focus on minimizing the computational capacity, the power consumption and processing time. The proposed solution must account for the project's requirements but should also be adapted to the hardware provided by the institution. Listed here are most of the components used in this project: sensors, processors, motors, mechanical components and power systems. Here, not only are presented the hardware's specifications, but also a verification of the system's requirements and a definition of the electrical interfaces between components. Figure 4.1 illustrates the hardware architecture along with the physical interfaces and their respective voltages.

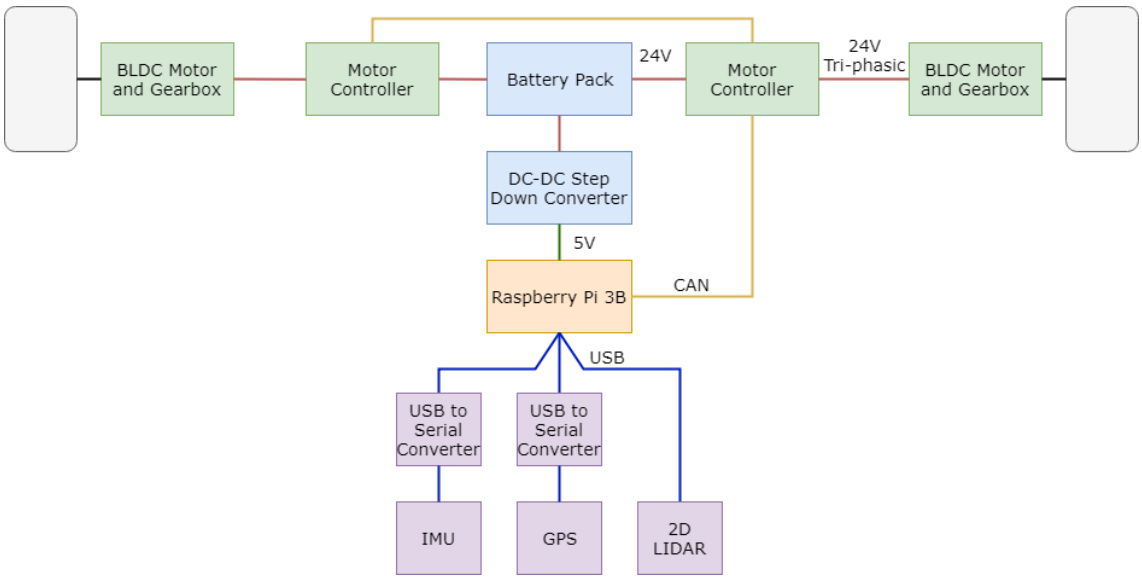

Figure 4.1: Components and electrical interfaces.

.

4.1 Processors

The main processor will be a Raspberry Pi 3B (Fig. 4.2), equipped with a quad core 1.2GHz 64Bit CPU, 1GB of RAM, 40 Pin GPIO, 4 USB 2.0 ports and a wireless LAN + Bluetooth low energy (BLE) module. The CPU and RAM provided by this small computer is enough for the 2D SLAM which is the more computationally demanding task, but this will be verified in the next chapter. The Wi-Fi module allows for easy interfacing (SSH) and communication (TCP/UDP) with the Pi, allowing one to create an access point with the Pi itself and permitting any device to easily connect. Finally, the 40 Pin GPIO and 4 USB 2.0 ports allow for multiple ways of interconnecting components with the Pi, and in this case the pins will be occupied by the **PiCAN 2 – CAN-Bus Board**, for interconnection with the motor controller via Ethernet cable.

Figure 4.2: Raspberry Pi 3 Model B.

4.2 Controllers

To control the motors, the **Nanotec C5-E-2-09 Motor Controller** (Fig. 4.3) is used. This controller can be configured either via CANopen or USB and it can be programmed, using a proprietary programming language, along with a full set of programmable analog inputs and digital input/outputs. It supports an input for the encoder and the output for the brake. Table 4.1 contains the most relevant specifications.

Specification	Value
Operating Voltage	12 V - 48 V
Rated Current	10 A
Peak Current	30 A

Table 4.1: Specifications of the C5-E-2-09 Motor Controller.

4.3 Sensors

The sensors are here listed along with the specifications most relevant to the problem in hand in order to establish the robot's capabilities. Examples of the general design considerations are

Figure 4.3: Nanotec C5-E-2-09 Motor Controller.

specified on section 2.1.1. The robot will possess a 2D LIDAR, an IMU, a GNSS module and encoders.

4.3.1 2D LIDAR

The **Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder** (Fig. 4.4) is a cost-effective, portable, light-weight and low-power sensor typically used by students and researchers in the robotics field. This sensor uses a laser pointing down towards a rotating mirror which is tilted 45^{o} to deflect the beam horizontally. Since the combination of this sensor and the IMU constitute the pillar of localization and mapping, the higher price for the increased robustness and reliability is a trade-off necessary due to its responsibilities and the intended environment. For the SLAM application, the main drawback is it's low frequency. This sensor also contains a mini USB port which will connect via an USB cable to the Raspberry Pi. Table 4.2 contains the most relevant specifications.

Specification	Value
Range	5600 mm
Scanning Angle	240^{o}
Accuracy	± 30 mm or $\pm 3\%$ over 1000 mm
Angular Resolution	$0.36^{o}(360^{o}/1,024$ $steps)$
Scan Time	100 ms/scan
Power Consumption	2.5 W

Table 4.2: Specifications of the Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder.

4.3.2 Inertial Measurement Unit

The provided IMU will be the **Adafruit BNO055 Absolute Orientation Sensor** (Fig. 4.5), which contains a three-axis accelerometer, a three-axis gyroscope, magnetometer and temperature sensor. This board has an integrated processor which filters sensor data, fuses it and outputs it in real-time as either quaternions, Euler angles or vectors, simplifying the orientation problem. The IMU is used by the SLAM task for its absolute orientation (referenced to the earth's magnetic poles), its

Figure 4.4: Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder.

gravity reference, and as a complement for the LIDAR based localization system. This IMU is able to output fused sensor data at a frequency of $1 - 20Hz$.

Figure 4.5: Adafruit BNO055 Absolute Orientation Sensor

Since the Raspberry Pi's GPIO interface is utilized by the CAN shield, the sensor was configured to work over UART and an **USB to Serial Breakout - FT232RL**, which fully implements the USB 2.0 protocol, with a micro USB port used to connect to the Raspberry Pi via USB cable (Fig. 4.6).

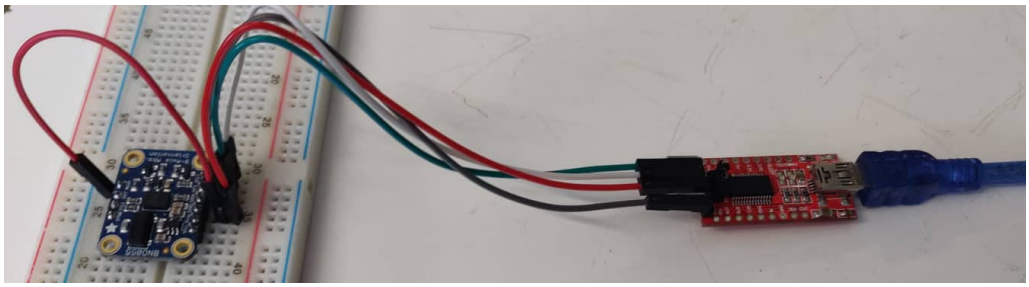

Figure 4.6: IMU connected to the USB to serial adapter.

4.3.3 GNSS Receiver

The provided GNSS sensor is the **GPS MOUSE - GP-808G**, containing a GNSS receiver and a compass, although only the receiver will be used since the IMU is also able to output the global orientation. The GNSS receiver provides a global position which will be used for mission tracking

and execution while also assisting the global SLAM task. Table 4.3 contains the most relevant specifications.

Specification	Value
Output Frequency	$1 - 10\ Hz$
Position Precision	2.5 m at Open Wind
Speed Precision	0.1 m/s
Acceleration Precision	0.1 m/s
Power Consumption	0.2 W
Interface	UART

Table 4.3: Specifications of the GPS MOUSE - GP-808G.

Similarly to the IMU interfacing problem, the GNSS data is also transmitted over UART and thus a similar solution was used (Fig. 4.7). In this case, an USB cable with an in-built USB to serial converter was used, ending with a robust and weatherproof solution.

Figure 4.7: GNSS receiver with a soldered USB cable with an integrated converter chip.

4.4 Traction

The traction method chosen for this robot will be a two-wheel differential drive, where two motors are controlled by two wheels, with one caster wheel in the back. This traction method is simpler, only requiring two motors, while also allowing for rotations in place. The traction system is verified by analysing the worst case scenario and evaluating if the system is capable of handling it. The requirements considered are the weight of the vehicle (Table 4.4), the minimum speed and the maximum inclination.

These are the defined requirements:

- **Total Weight -** 43.7kg
- **Minimum Speed -** 1m/s
- **Maximum Inclination -** 30º

Component	Weight
Robotic Manipulator	20 kg
Motors (2x)	1.6 kg
Gearbox (2x)	5.5 kg
Structure and Eletronics	5.1 kg
Batteries (2x9	11.6 kg
Total	43.7 kg

Table 4.4: Weight of the components and the vehicle's total.

4.4.1 Motor Sizing

Along with the preparation work points, two DC motors were sized for the differential drive traction that will be later implemented. The calculations are hereby presented along with the imposed requirements of the system (eqn. 4.1).

$$\begin{aligned} m &= 43.7kg \\ \theta_{max} &= 30^{o} \\ v &= 1m/s \\ r &= 0.13m \end{aligned} \tag{4.1}$$

First, the result of the forces acting on the wheel is defined. Since this is a simplified model, friction forces will not be considered in the x-axis. The sum of y-axis forces is also considered null (eqn. 4.2).

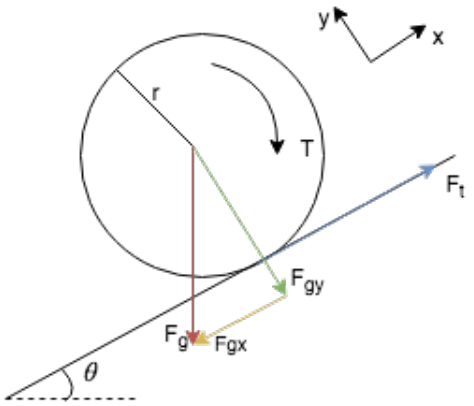

Figure 4.8: One wheel model with no friction force considered. F_g is the force exerted by gravity and F_t is the force that induces forward motion,

$$\begin{aligned} \sum F_x &= m \cdot a \\ F_{gx} &= F_g \cdot sin\theta \\ F_t &= \frac{T}{r} \end{aligned} \tag{4.2}$$

In order to calculate the necessary torque the equation is reworked the parameters replaced with the initially defined values (eqn. 4.1), note that the assumed acceleration was defined by the following rule of thumb, half of the intended velocity. So in summary, the worst case scenario for the robot is considered, which is a 30^o inclination climb from a full stop.

$$\begin{aligned} &\sum F_x = m \cdot a \Leftrightarrow \\ &m \cdot g \cdot sin\theta + \frac{T}{r} = m \cdot a \Leftrightarrow \\ &T = (a + g \cdot sin\theta) \cdot m \cdot r \end{aligned} \tag{4.3}$$

In this case, the total necessary torque is $30.68Nm$ which, divided by two wheels equals $15.34Nm$. The angular velocity of the wheel at $1m/s$ is about $6.66rad/s$ so the output power at the load (wheels) is the multiplication between torque and angular velocity which in this case gives us a useful power output of $174.4W$ per wheel.

Finally efficiencies must be considered. Here two main sources of loss of power can be identified, the gearbox and the general mechanical losses such as friction, wheel slippage and systematic mechanical losses. Here (fig. 4.9) a worst-case scenario is again considered where the gearbox is considered performing at it's worst and there is an additional 10% loss in the rest of the mechanical process, these considerations guarantee that our motor is not undersized and provides staple values for motor and gearbox component selection. Concluding, two motors with around $220.1W$ output power is required, and the transmission must guarantee that at least $15.34Nm$ are applied to each wheel.

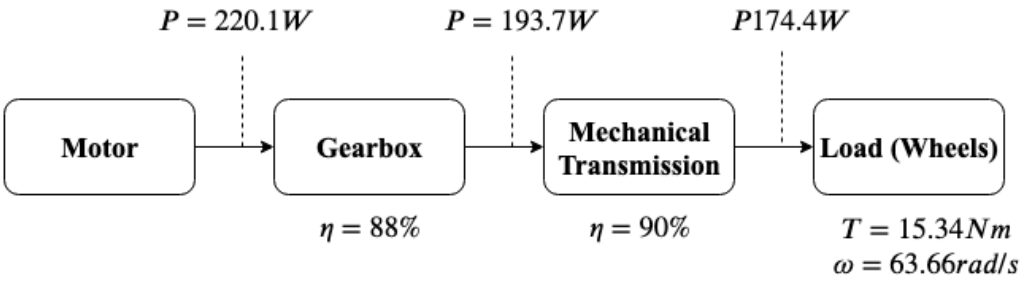

Figure 4.9: Power scheme for a single wheel.

4.4.2 Traction components

The robot is equipped with two Brushless DC (BLDC) Motors, with an attached brake each, connected to a planetary gearbox which is then connected to the wheel. The motor possesses an in-built incremental encoder which then connects to the motor controller that will try to output the speed which is set by the Raspberry Pi. Following is a list of the aforementioned components along with their main specifications.

4.4.2.1 BLDC Motor

The **Nanotec DB59C024035-A** (Fig. 4.10) is an internal rotor BLDC motor optimized for high efficiency, low noise and a large power/size ratio. Table 4.5 provides the main specifications of the motor. The motor has barely enough power to perform the requirements, with it's rated power almost coinciding with the estimated value ($220.1W$).

Specification	**Value**
Rated Power	220 W
Rated Voltage	24 V Tri-phasic
Rated Current	10 A
Rated Speed	3500 rpm
Rated Torque	60 $N.cm$

Table 4.5: Specifications of the Nanotec DB59C024035-A motor.

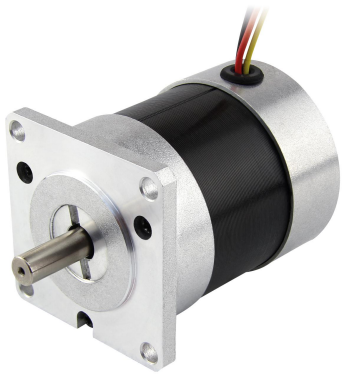

Figure 4.10: Nanotec DB59C024035-A BLDC Motor.

4.4.2.2 Gearbox

The **Neugart WPLE60-32 Planetary Gearbox** is an economic gearbox which is easy to install, maintain, and has a focus on the price/performance ratio. A notable characteristic of this gearbox is the key output shaft (Fig. 4.11), which ensures that the mechanical connection transmits power and rotation without slipping, simultaneously preventing rotations in an unwanted direction. Table 4.6 contain the main specifications. Note that the reduction ratio of the gearbox is ideal (32:1 with a

3500 rpm motor), putting the locomotion system in range of both the speed requirements ($1m/s$) and the traction requirement ($15.34Nm$).

Specification	Value
Torque	$14-44\ Nm$
Max. Input Speed	$13000\ rpm$
Efficiency (full load)	$88-95\%$
Ratio	$32:1$

Table 4.6: Specifications of the Neugart WPLE60-32 planetary gearbox.

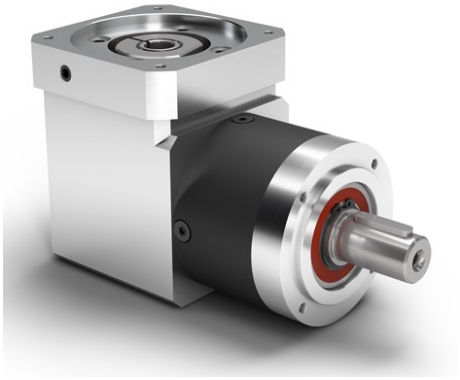

Figure 4.11: Neugart WPLE60-32 planetary gearbox.

4.4.2.3 Brake

The **Nanotec BWA-1.5-6.35** (Fig. 4.12) is an electromagnetic spring-loaded brake which is not meant to slow down the motor's motion, but as a holding brake, inhibiting rotor displacement. Since the magnets pull the springs which are attached to a disk that causes friction, when the brake has no power it automatically engages, for security, stopping the motor. Table 4.7 contains the main specifications.

Specification	Value
Holding Torque	150 N.cm
Switch-on Time	100 ms
Rated Power	11 W

Table 4.7: Specifications of the Nanotec BWA-1.5-6.35.

4.5 Power

The system will be powered by two RS Pro Lead Acid Batteries in series. The batteries combined will have a $24V$ nominal voltage with a capacity of $40Ah$. The motor controller will connect directly to the battery while the processor will require a DC-DC step-down converter with an output of $5V$.

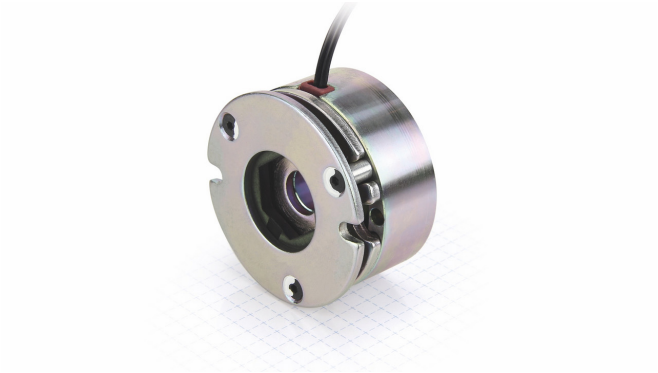

Figure 4.12: Nanotec BWA-1.5-6.35.

An estimation of battery duration is required to verify if the system is able to complete a defined mission. Table 4.8 shows the estimated power consumption of the components and the estimated battery duration. The three columns show the difference when considering an utilization factor on the motor, since the motors will not run in a continuous operation.

	100%	75%	50%
Motors	440 W	330 W	220 W
Brake	11 W	11 W	11 W
Processor	2.4 W	2.4 W	2.4 W
Sensors	2.8 W	2.8 W	2.8 W
Total Power	456. 2W	346.2 W	236.2 W
Duration	2.11 h	2.78 h	4.06 h

Table 4.8: Power consumption of each component and estimated battery duration.

Concluding, a minimum of 2 hours of operation can be guaranteed and 4 hours or more are to be expected, since the motors don't constantly run at full power in this application and can be configured to run at a speed lower than the maximum.

4.6 Costs

Table 4.9 contains the list of components used in this project along with the respective quantities and prices. The sum amounts to 2836.33€, verifying the cost requirement initially established. In this case, the two most expensive components are the motors and gearbox, that depend on the sizing done for the application, and the 2D LIDAR, which is the pillar of this robot's SLAM. The rest of the components were chosen based on their cost-effectiveness. Adding structure and operation equipment to the cost of the system, it's final cost is extremely competitive when comparing to other small-sized vehicles in the market, where prices range around 20000€ to 30000€.

Component	Quantity	Price/Unit (€)	Total (€)
Raspberry Pi 3B	1	29.90	29.90
Nanotec C5-E-2-09 Controller	2	228.50	457.00
Hokuyo URG-04LX-UG01 LIDAR	1	840.00	840.00
Adafruit BNO055 IMU	1	30.95	30.95
GPS MOUSE GP-808G	1	33.96	33.96
Nanotec DB59C024035-A Motor	2	180.00	360.00
Neugart WPLE60-32 Gearbox	2	341.40	682.8
Nanotec BWA-1.5-6.35 Brake	2	52.90	105.8
RS Pro Lead Acid Battery 12V 20Ah	2	57.96	115.92

Table 4.9: Price list of the components used in this project.

4.7 Final Assembly

The platform was assembled with scrap parts that were available in the department, and many tasks such as cleaning, painting and mechanical work were done. The final result is a robust platform that is able to easily support the batteries and defined 20kg payload. Figure 4.13 shows some of the stages of development of the platform, and the different type of tasks done, and figure 4.14 shows the platform equipped with the mentioned electronic components. Finally, figure 4.15 shows the 3D model of the platform that was drawn and added to the open-source repository.

4.8 Chapter Summary

The hardware necessary for the platform was chosen, their interfaces were defined and implemented, and a prototype of the robotic platform was assembled. All the sensors were tested and successfully integrated with the Raspberry Pi, but the GNSS receiver had a manufacturer's defect that made it unusable. Finally, a requirement verification of the mechanical aspects of the platform was done, as well as a cost-analysis of the solution in the context of this thesis. It is believed that the selected hardware, with proper preparation, is able to robustly perform in agricultural settings.

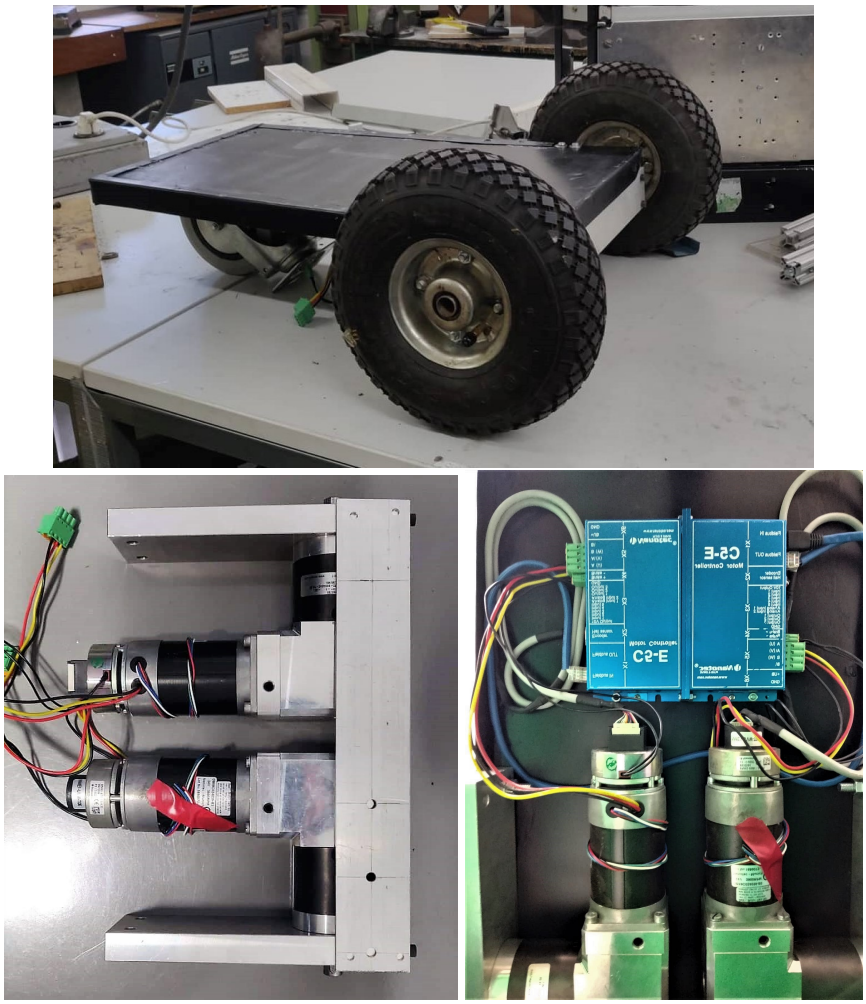

Figure 4.13: On the top, the platform mid-way built. On the left, the built motor support. On the right, the motor electronics mounted under the platform.

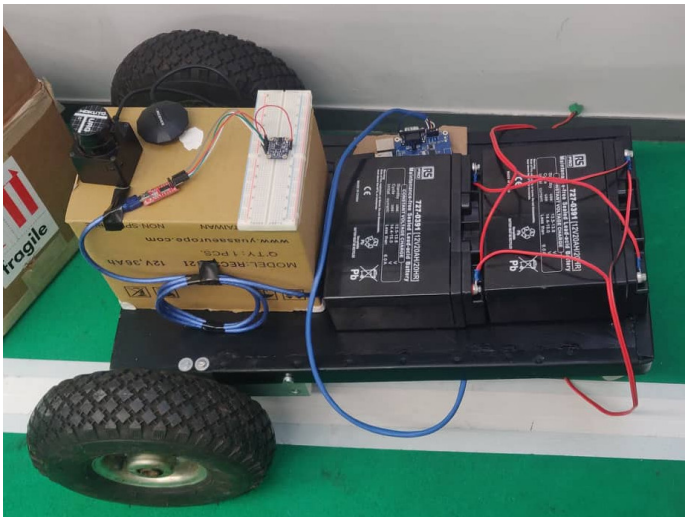

Figure 4.14: Photograph of the final robot assembly.

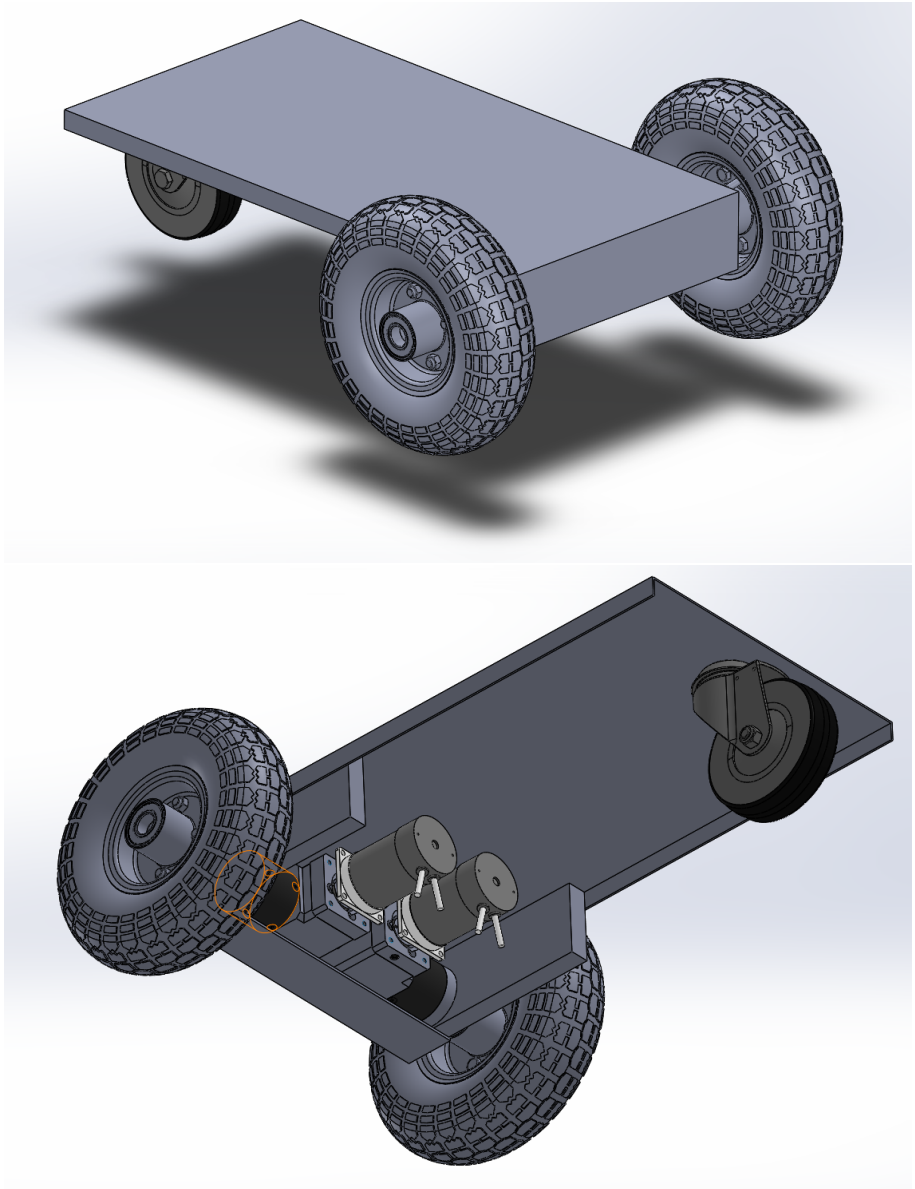

Figure 4.15: 3D CAD model of the platform.

Chapter 5

Experimental methodology

In this section the conducted experiments' methodology, the results and subsequent interpretations, and the conclusions taken in context of the setting of this project are presented. The phases follow the chronological order and each aim to verify some requirement of the system or gain knowledge about the problem. The first section describes a simple indoor situation where the SLAM system is tested using only the LIDAR and the IMU. In the second section, more challenging tasks are presented to the system. Finally, the tests are executed in the final version of the robot.

5.1 Indoor Testing

The main objective of the indoor testing was acquiring data for post-facto tuning of the SLAM system, and evaluating the performance of the Raspberry Pi 3B in a real-time (online) situation. The indoor testing here was performed using the Raspberry Pi, IMU and 2D LIDAR, powered by a power bank. The components were fixed to a piece of square plywood and mounted on a pushcart as shown in figure 5.1.

In this phase two goals were accomplished. First, testing the integrated components in an online setting (instead of a simulated one) and troubleshooting the problems that arose. Secondly, acquiring quality records to further evaluate the performance of the system and further tune the algorithm to the application. The concerns in the choosing of the path were to challenge the algorithm in a straight path with little features (parallel walls) and a way to do a circular/square trajectory to analyse the rotational error. So, the first floor of the electrical engineering department was chosen and the obtained map (with and without IMU data) after tuning is presented in figure 5.2.

The map obtained without using the IMU data and only using it's gravity reference possesses angular distortions even after precise tuning. The angular velocities output by the IMU compensate the limited output frequency of the laser scan by providing a baseline from the pose prediction. These differences are also reflected in the tuned rotation and translation weight, where a higher value imposes that there is a higher confidence in the localization prediction, whereas without the IMU data, lower confidence values have to be used in order to obtain any sort of discernible map.

Figure 5.1: Experimental setup mounted on a plywood, mounted on a pushcart

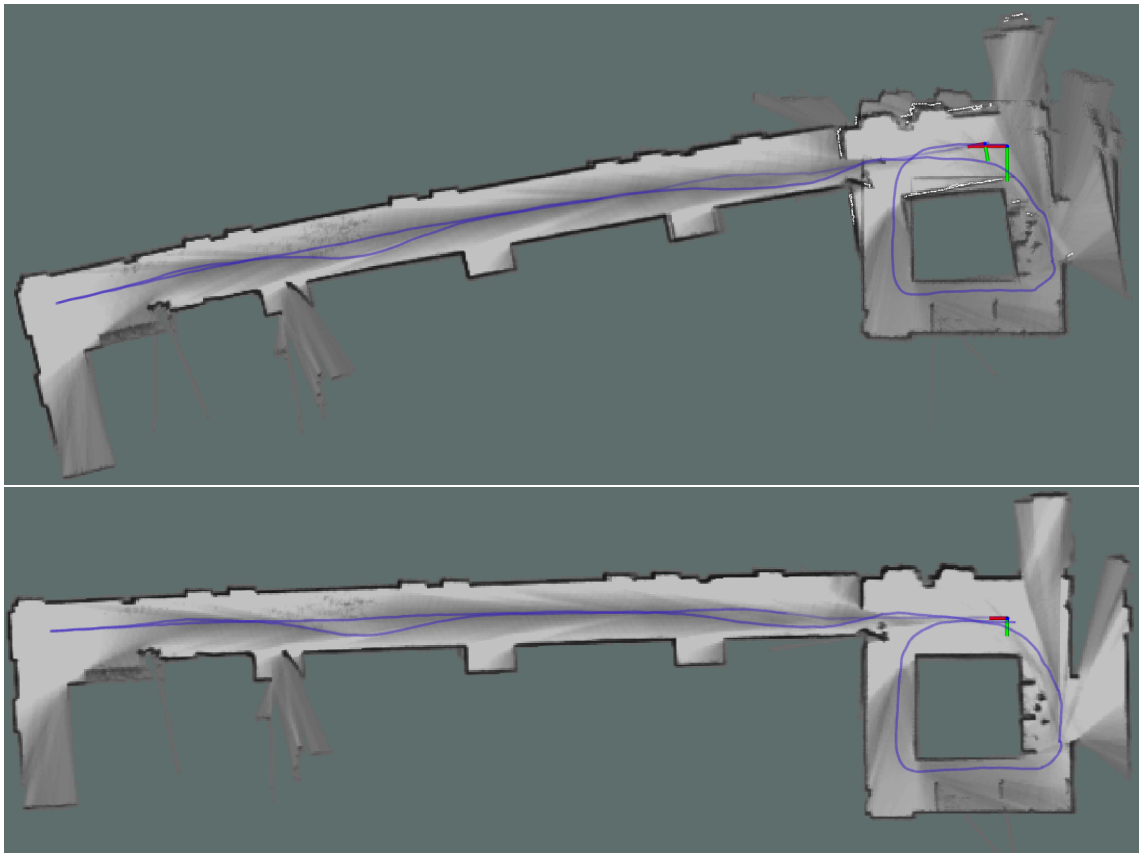

Figure 5.2: Obtained maps from the first floor of the electrical engineering department, on the top is the output without IMU data and below with.

Table 5.1 contains the local SLAM weights obtained from calibrating the algorithm, where the weights reflect the confidence of the output of the LIDAR readings in the task.

	Rotation Weight	Translation Weight
No IMU	2	2
Fused IMU	3	5

Table 5.1: Weights used in the indoor testing.

In the map where there is fusion with IMU data their is still an error in the rotation estimation as can be seen in the squared section on the right of the image and in the angle that the corridor makes with the rest of the floor. This error should be minimized by the introduction of odometry as a baseline for localization and a better pose definition.

Also shown in figure 5.3 is the screen capture of the result of the *htop* Linux commands which gives runtime information about the computer. Running the SLAM algorithm along with the rest of the ROS functions associated with it occupies 264 MB of RAM and takes up about the half the processing capabilities of the Raspberry Pi, therefore the processing capabilities requirements are verified for the chosen processor.

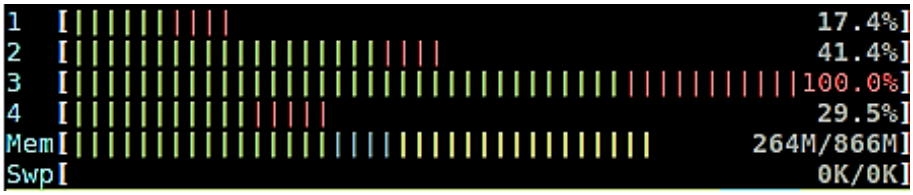

Figure 5.3: Capture of Linux *htop* command. Each number on the left represents a core in the processor, the percentage represents their current utilization.

5.2 Outdoor Testing

Although the SLAM was successful in the previous experiments, the setting is far from the intended one. The SLAM task is easy with strong features such as perpendicular walls and chairs on a smooth floor, but how does it perform in a more challenging setting? For this, two locations were chosen (Fig. 5.4): First, the faculty's parking lot which introduce harsh terrain with strong vibrations on plastic wheels and secondly, a path behind a dirt parking lot next to the INEGI building. From these tests, the aim is to validate the processor's SLAM quality as well as it's online performance while running the rest of the software and cross-comparing against a simulation using a PC. The mission system and the user interface are also tested, especially the coordinate conversion, trajectory representation and goal error calculations.

Figure 5.4: Google maps capture of the parking lot and photograph of the area next to the INEGI building.

5.2.1 SLAM Testing

The map obtained in the parking lot is presented in figure 5.5. It shows the system starting from a point, traversing a row with cars on both sides, performing a curve and returning. It is visible that when in the row and with feature readings on both sides, the resulting trajectory is smooth and there is also high map definition. However, the curve on the corner is heavily distorted and the distance between the top lane and bottom lane is shorted. There is also some angular error which can be seen in the returning trajectory, where a slight inclination indicate that the trajectories will cross.

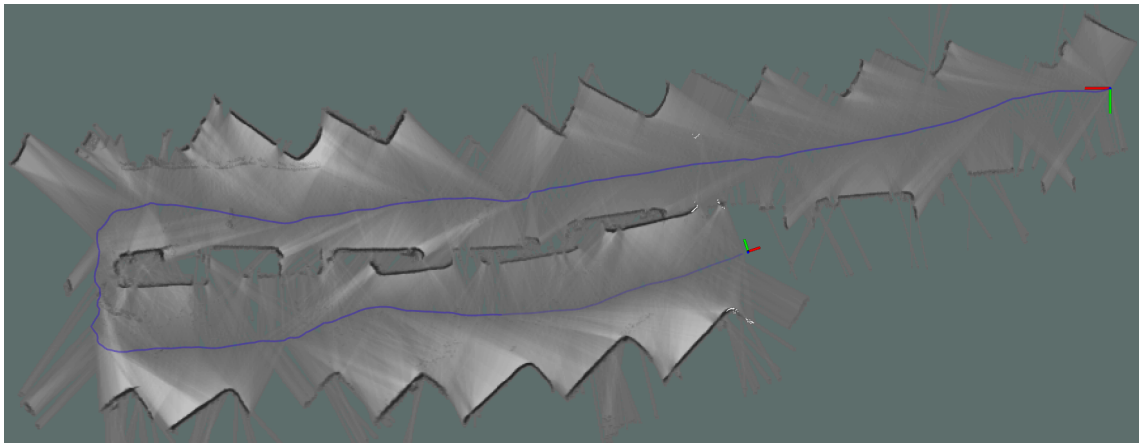

Figure 5.5: Map obtained from the parking lot.

Since the algorithm uses feature matching, the main source of error in this situation is when the

system can't read anything in it's scans or the read information is insufficient for proper estimation. In this case specifically, a moment was identified (Fig. 5.6) where in the start of the curve the scanner only picks up points on one corner bumper of a car, not picking up the one on the opposite side for an unknown reason. This situation constitutes an ambiguity, since it is impossible to determine pose and translation information from one read point uniquely (minimum of 2 being necessary for triangulation). No tuned solution was achieved that could fully solve this situation since the IMU data was not sufficient to correctly perform this curve, especially since IMU data the was noisy due to the vibrations caused from the pushcart's hard plastic wheels and the irregularities of the tar.

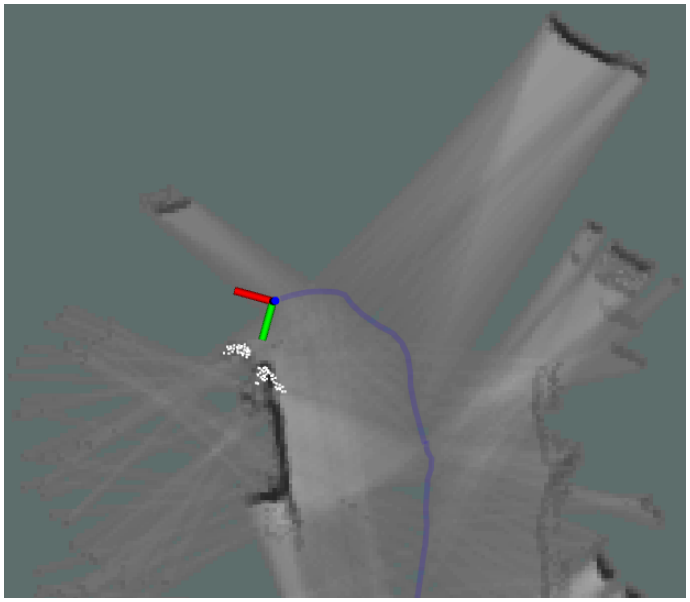

Figure 5.6: Curve point where the scans are insufficient for proper SLAM. The matched points are represented by the white dots.

Although in this situation the curve couldn't be mapped correctly, figure 5.7 shows a row passing and a curve where the quality is noticeably better since it has multiple readings all the way.

To also test the possibility of using the algorithm in a setting approximate to the intended one, a location with vegetation and other naturally introduced noise was chosen to perform experiments. Although traversing harsh terrain without pneumatic wheels, which interferes with the IMU and the consistency of the laser scans, and attempting to localize without odometry makes the task extremely hard, good results were still obtained in some sections. In figure 5.8, the system is able to go back and forth to validate the map and location estimation, use dense vegetation for feature matching, and also able to repeatedly read strong features hidden by vegetation, as can be seen in the black points in front of the rear car bumper.

5.2.2 IMU in the SLAM task

The impact the IMU has in the output varies on the task, it relates to both the factors that affect the IMU only (such as the harsh terrain and subsequent vibrations) and the confidence of the output

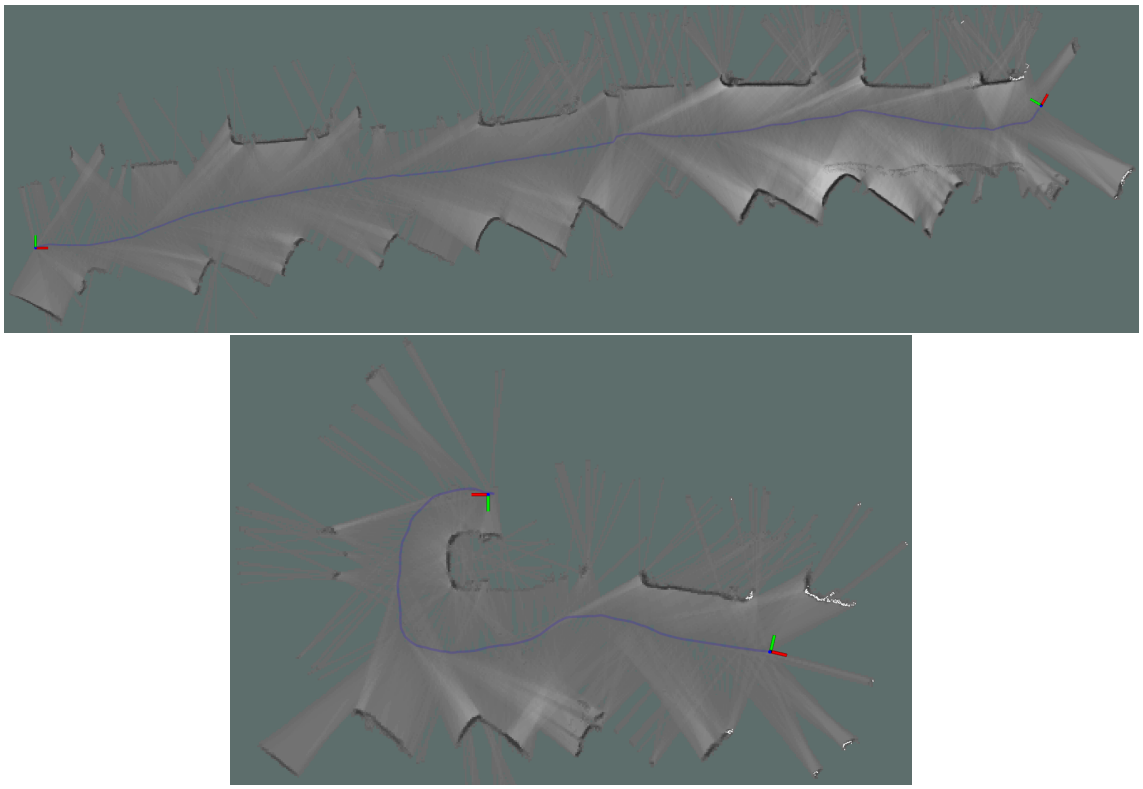

Figure 5.7: Car row passing and curve obtained from the parking lot, with an accurate SLAM output.

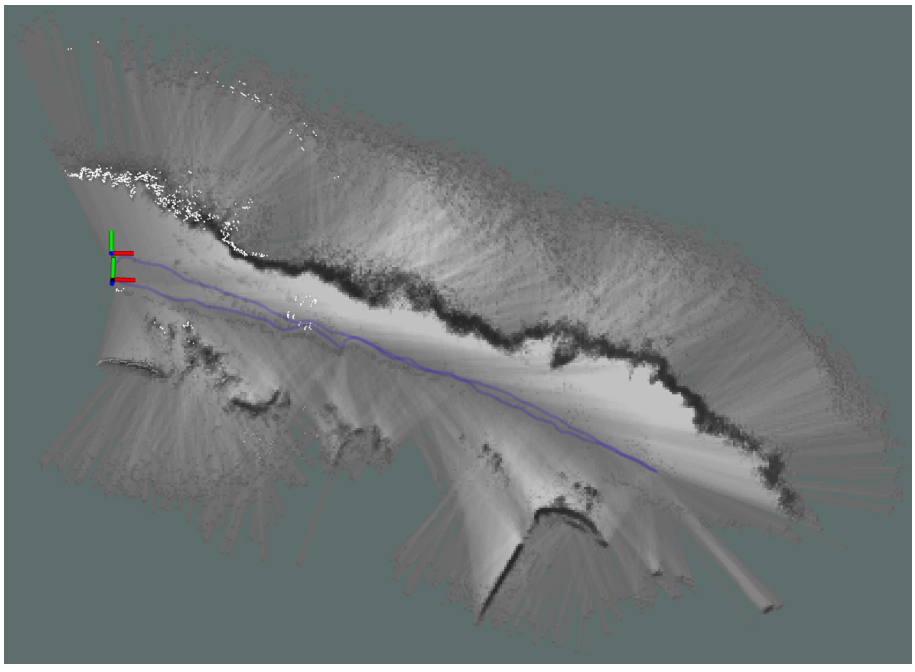

Figure 5.8: Map obtained from a path between natural vegetation and a dirt parking lot.

of the SLAM, this means that if the scan feature matching is repeatable and consistent the IMU plays a lesser role as the baseline of the localization. Two maps are shown below that portray this

effect. The map in figure 5.9 shows the same car row pass as before but this time without fusing the IMU data, in this case the IMU helps smooth out the trajectory and prevent "jumps" in the position estimation, these cases are highlighted below. The second map in figure 5.10 represents the same location as figure 5.8. In this case, the IMU is essential since the SLAM algorithm has to reject most of the noise found in the natural settings and preserve the strongest features in the setting to properly localize, which in this case is impossible without the aid of the IMU.

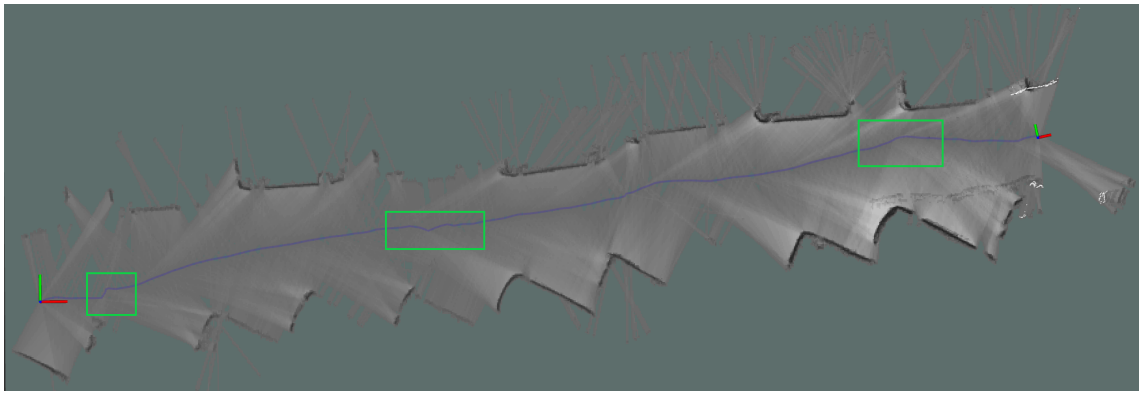

Figure 5.9: Map obtained from row of cars in parking lot, without fusion of IMU data. Green boxes highlight "jumps" in the trajectory estimation.

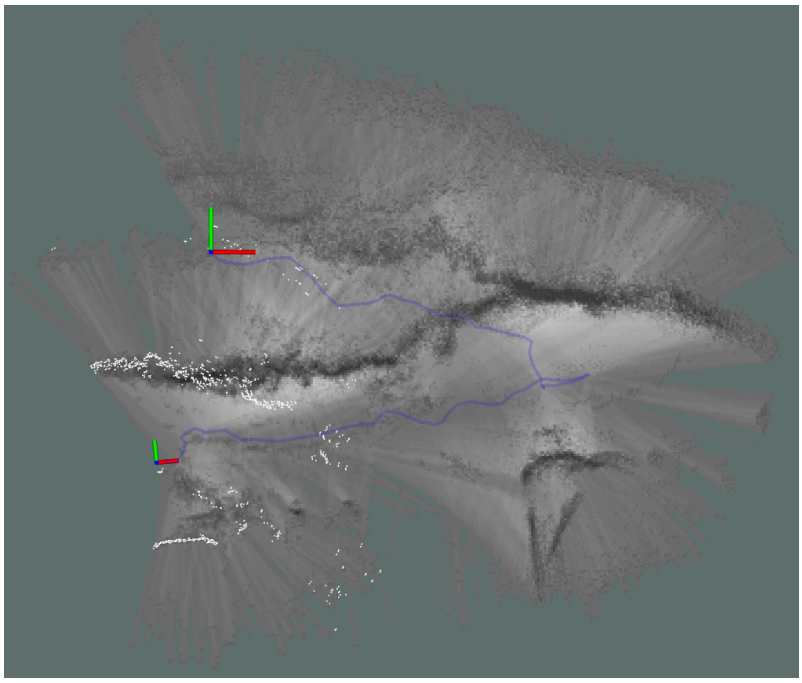

Figure 5.10: Map obtained from a path between vegetation and a parking lot, without fusion of IMU data.

Table 5.2 shows the resulting tuned weights of the algorithm. The lower weights in comparison with what was obtained in table 5.1 reflect the unstructured nature of the chosen testing sites.

Map		Rotation Weight	Translation Weight
Tar Parking Lot	No IMU	0.2	0.8
	Fused IMU	0.3	0.9
Dirt Parking Lot	No IMU	0.15	0.8
	Fused IMU	0.3	0.9

Table 5.2: Weights used in the outdoor testing.

5.2.3 Performance in the Raspberry Pi

No major differences in map quality were observed when computing the maps in a desktop PC or in the Raspberry Pi. The time measurements in table 5.3 also show their are no major differences between the PC and the Raspberry Pi. This indicates that the Pi is sufficient for the SLAM task in different settings, with verification of online performance as well.

	Parking Lot	Dirt Field
PC Map Frequency	1 Hz	1 Hz
PC Localization Frequency	33 Hz	33 Hz
PC Average Latency	67 ms	77 ms
PC Maximum Latency	149 ms	225 ms
RPi Map Frequency	1 Hz	1 Hz
RPi Localization Frequency	33 Hz	33 Hz
RPi Average Latency	68 ms	96 ms
RPi Maximum Latency	188 ms	232 ms

Table 5.3: Localization frequency, mapping frequency, average and maximum latency between scans and pose output.

5.2.4 Comparison between Cartographer SLAM and Hector SLAM

To further verify if Cartographer's choice over Hector was appropriate, tests to the output of the SLAM using only the laser scans and the IMU for orientation were done on the logs collected outside. The side-by-side maps are presented in figures 5.11 and 5.12.

In the first figure, in the parking lot, the issues with Cartographer were minor jumps, but in Hector there are large errors in both rotation and translation, as can be seen in the differences in length and the non-existing curvature of the road. Besides, the map quality of Cartographer is significantly superior, as can be seen in the amount of detail that can be noticed in the cars, especially around the wheels. In the second figure, both Cartographer and Hector perform poorly. Hector's low detail aids the feature extraction om he vegetation, allowing for a smooth trajectory but stopped working midway due to insufficient meaningful data. Cartographer relies more heavily on preserving the strong features found on the cars despite the interfering vegetation, the error is mostly angular, as seen in situations before, and is partially corrected with IMU data and possibly

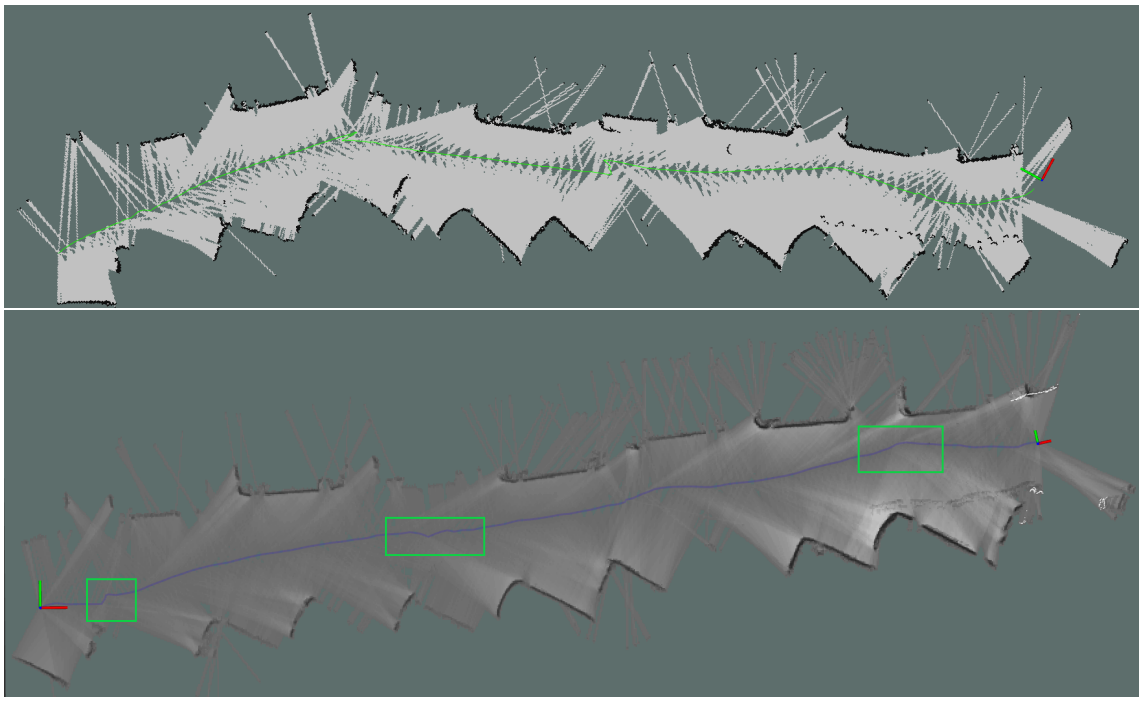

Figure 5.11: Map obtained from row of cars in parking lot, without fusion of IMU data. On the top is the Hector SLAM result and on the bottom Cartographer's. Green boxes highlight "jumps" in the trajectory estimation.

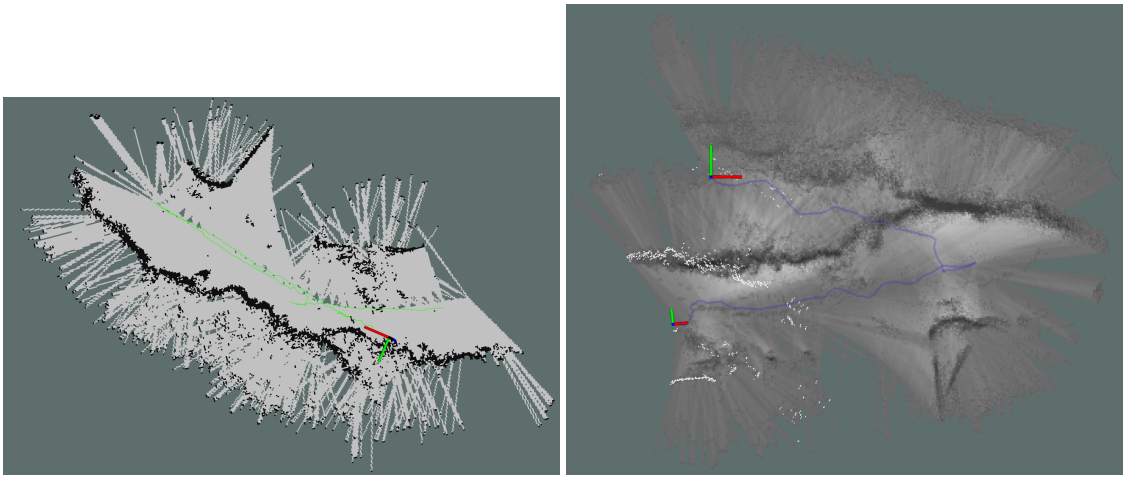

Figure 5.12: Map obtained from a path between vegetation and a parking lot, without fusion of IMU data. On the top is the Hector SLAM result and on the bottom Cartographer's.

corrected with wheel odometry data. Concluding, Cartographer was able to output scan matching results with higher quality, consistency and less error than Hector.

5.2.5 Mission Testing

The goals of mission testing are to validate the trajectory and orientation information passed to the user interface and validate the goal error calculation. In figure 5.13 a trajectory done by the

vehicle is done and is shown simultaneously in the user interface, allowing the user to track the vehicle's location and direction. Figure 5.14 shows the goal error calculation for a goal point in the four quadrants around the vehicle, with recorded robot logs to simulate a real situation. The values below represent the errors between the goal and the vehicle and, as it can be seen, the global goal calculation is accurate.

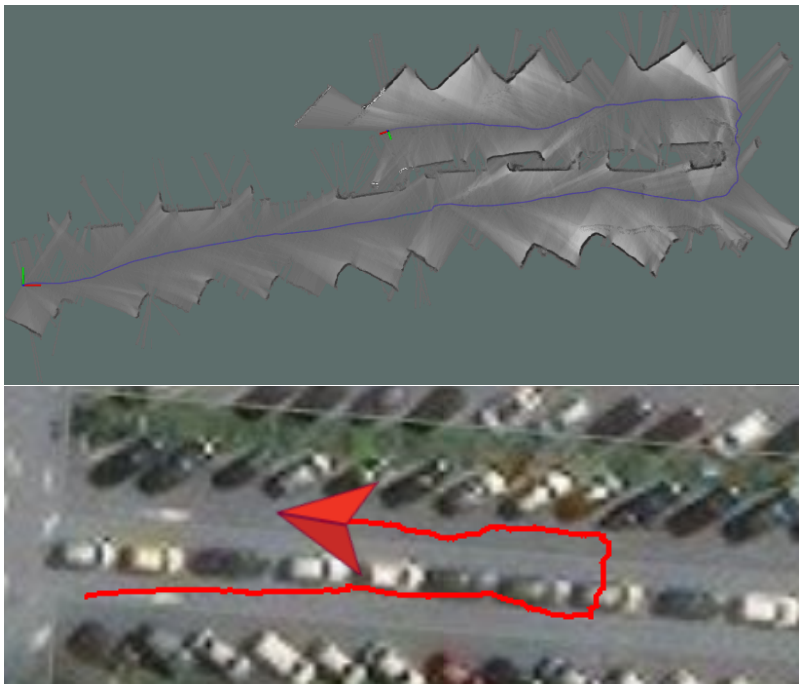

Figure 5.13: Passed trajectory from ROS to QGC.

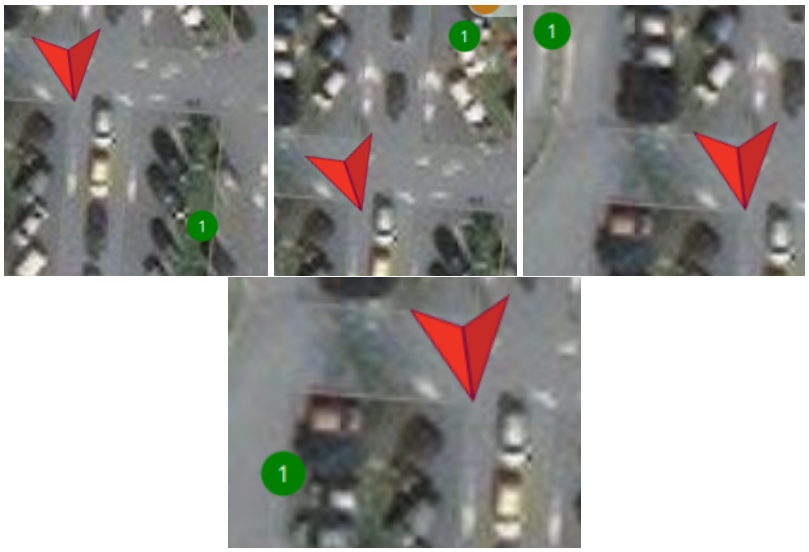

Figure 5.14: Goal points set in QGC. Distance errors (in meters) are, respectively: 19.26; 17.00; 18.09; and 14.3. Angle errors (in degrees) are: 46.75; 159.85; -120.03; and -48.10

5.3 Final Testing

The last section of this chapter describes the testing and results on the final version of the platform. Three goals were accomplished in this section: First, the SLAM's performance was further validated in more settings and while performing in the Raspberry Pi; Second, the navigation was successfully calibrated and finalized; And finally third, the UMBMARK calibration method was performed. This phase also allowed for the analysis of the odometry's impact in the algorithm, albeit it was also discovered that often the achieved odometry is often problematic.

5.3.1 Data collection and SLAM calibration

To obtain a map location that could test different parts of the SLAM function, the -1 floor of the Electrical Engineering department was chosen. To test the local component of SLAM, this location possesses locations with many features which facilitate scan matching, a long corridor which requires the odometry more, and several 90 degree turns to easily see the rotation precision. Globally, the quality of the obtained map is verified by assessing the geometry of the building. The trajectory chosen consists of going from one point of the floor to another, perform a half-turn and return.

This trajectory was performed two times, in order to allow for verification in multiple instances and prevent over-tuning of the algorithm to a specific set of data. The obtained maps and final result of the calibration are shown in figure 5.15. The maps have enough quality and similarity to the location that they can be used for autonomous navigation. Two effects were also noticed during this phase of testing: An error in scan matching rotation output, possibly due to measurement errors in the orientation and position of the laser frame relative to the base link of the robot and the low frequency of the laser for this application; And an error in odometry due to imprecision in the mechanical assembly of the locomotion components.

The resulting parameters from the calibration were tested in the Raspberry Pi in an online performance with the rest of the robot's systems, and the obtained final map is shown in figure 5.16. The quality is good, even without proper odometry, and viable for indoor autonomous navigation. The average pose publishing latency was 107ms and it's maximum 189ms, well within the requirements for the robot's low speed intended uses.

One outdoor large test outside allows the drift error to accumulate, and make the localization useless in a real situation, as seen in figure 5.17. Although the algorithm's performance starts well, as can be seen in the top left corner, it starts to drift midway and after the second turn, it registers an incorrect pose and then drifts permanently. If the odometry's weight would be increased to try and deal with this situation, the drift would be noticeable much sooner and the map quality would become worst, making the calibration of the odometry the next step.

The obtained parameters in table 5.4 reflect the current state of the robotic platform. Due to problems in the mechanical assembly, the algorithm has little trust in the odometry values as can be seen in the low scan matcher translation and rotation weights. This effect is also reflected in the global part, since the odometry cannot be used in order to get the performance, meaning that it is

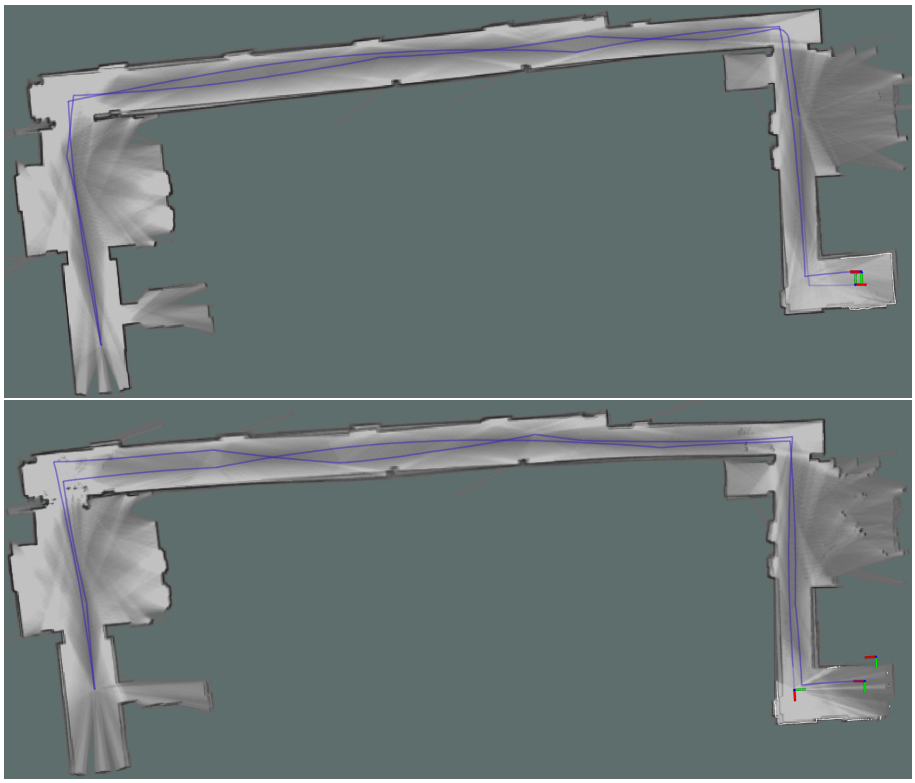

Figure 5.15: Obtained maps, of the -1 floor of the Electrical Engineering department, after calibration of the algorithm in a simulated environment.

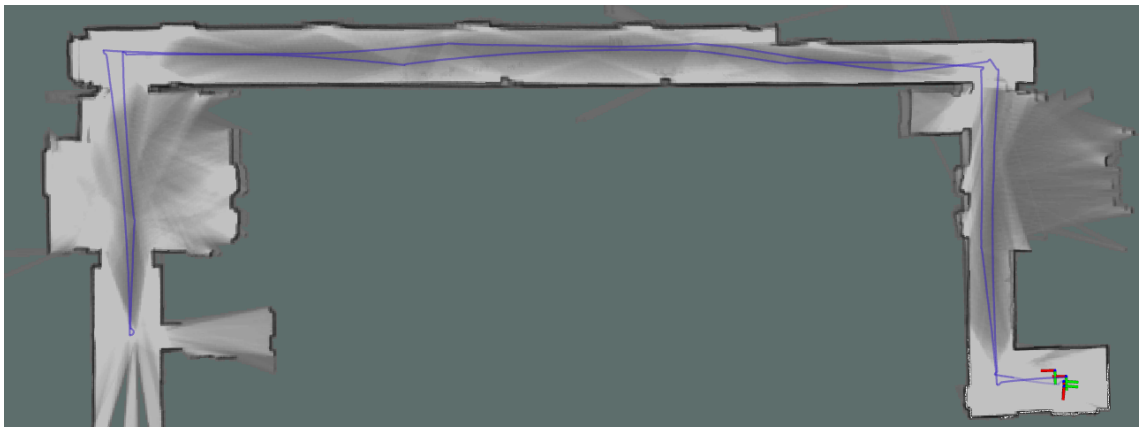

Figure 5.16: Map obtained from the robot in an online situation.

not usable in this case. A calibration is necessary following this situation, so the odometry is usable enough that it can at least fill the gaps when the algorithm has little or no scan readings.

The Raspberry Pi was able to handle the algorithm with a high intensity of background global SLAM calculations, but the program had to be limited to only three threads in the system, since occasionally the algorithm can push the cores to 100% CPU utilization. If it had all the four threads, it would block all the nodes simultaneously, resulting in critical failure. An example of otherwise common utilization of processing resources is shown in figure 5.18, the average utilization varies

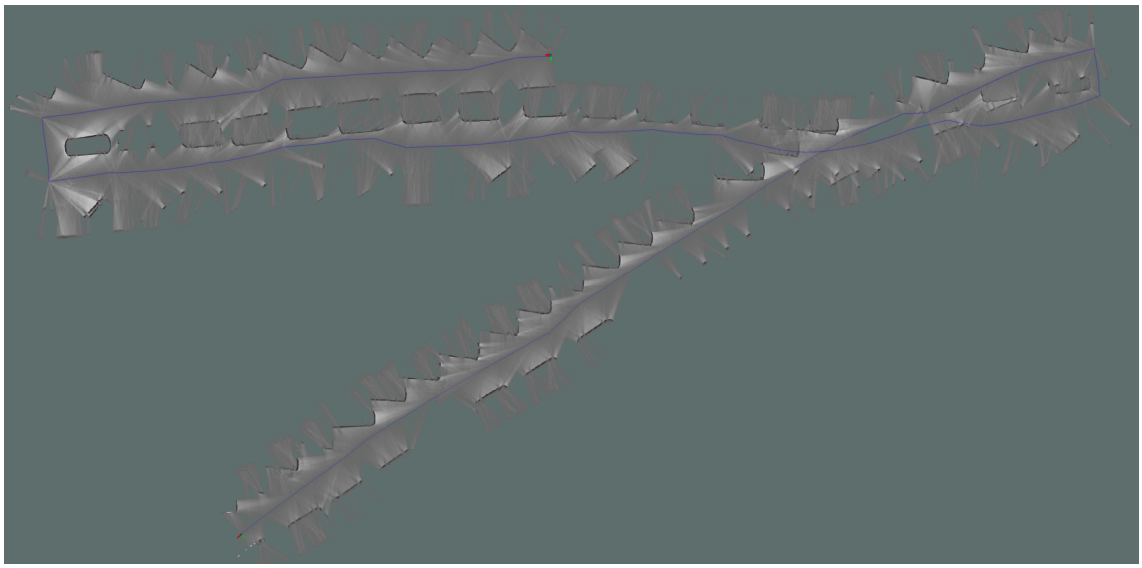

Figure 5.17: Outdoor map captured in the parking lot.

Parameters	Values
Map resolution	0.03 m
Number of accumulated range data	80 scans
Scan matcher translation weight	4
Scan matcher rotation weight	2.5
Voxel filter size	0.02 m
Global: Number of nodes for optimization	70
Global: Local SLAM pose translation weight	100000
Global: Local SLAM pose rotation weight	100000
Global: Odometry translation weight	0
Global: Odometry rotation weight	0
Global: Constraint builder min. score	0.6
Global: Huber scale	10

Table 5.4: Final (resumed) parameter list obtained from Cartographer's calibration.

between 40% and 60%.

Figure 5.18: Capture of Linux *htop* command, with global part active along with the rest of the robot's systems.

From these experiments, it can be concluded that the system is at the moment not prepared for real scenarios without correction of the odometry. Unfortunately the GNSS receiver was

malfunctioning and the signal quality was not enough to get a correct fix. Although these problems arose, the SLAM algorithm is proving to be an effective and robust solution, but it is not sufficient as a standalone with the IMU.

5.3.2 Indoor Navigation

The parameters listed in section 3.5 of chapter 5.5. These parameters were hand calibrated in order for the robot to be able to depart, follow and arrive correctly. The speeds are here represented as factors of the full speed, and the current maximum speed, which coincides with the normal linear velocity, is $0.63m/s$ for safe indoor use. A program was also made to test the navigation where the robot accepts as inputs XY coordinates and navigates towards them

A test was made to exemplify the navigation performance. The robot performs a square in the clockwise direction, while actively using SLAM to estimate position, and navigate to the chosen point with a $10cm$ acceptance radius. The obtained map and trajectory estimation is visible in figure 5.19, where the robot is able to perform the square without glitches, compensating if the error in the trajectory is too large, meaning that the navigation was successfully achieved. Note that, in this case, the final orientation of the robot is not set, since usually the robot will rotate to the next waypoint in the mission.

Parameters	Values
Normal turning velocity (W_NOM)	0.5
De-acceleration turning velocity (W_NOM_DE)	0.075
Maximum initial rotation error (MAX_ETF)	$0.0025rad$
Error hysteresis while moving (HIST_ETF)	$0.05rad$
Normal linear velocity (LIN_VEL_NOM)	0.5
De-acceleration linear velocity (LIN_VEL_DA)	0.3
Angular error controller gain while in GO_FORWARD (GAIN_FWD)	4.25
Angular error controller gain while in DE_ACCELERATE (GAIN_DA)	2.5
Distance to trigger DE_ACCELERATE (DIST_DA)	$1.5m$
Distance to trigger FINAL_DE_ACCELERATE (DIST_FINAL_DA)	$0.5m$

Table 5.5: Parameters obtained from calibrating the navigation.

5.3.3 UMBMARK Method

As mentioned in section 3.2.4, the UMBMARK method was performed as an attempt to minimise the odometry errors. The method consists of performing 5 squared paths in each direction, and measuring the robot's position at the end. With the measured positions an estimation of the error is given and correction factors were applied.

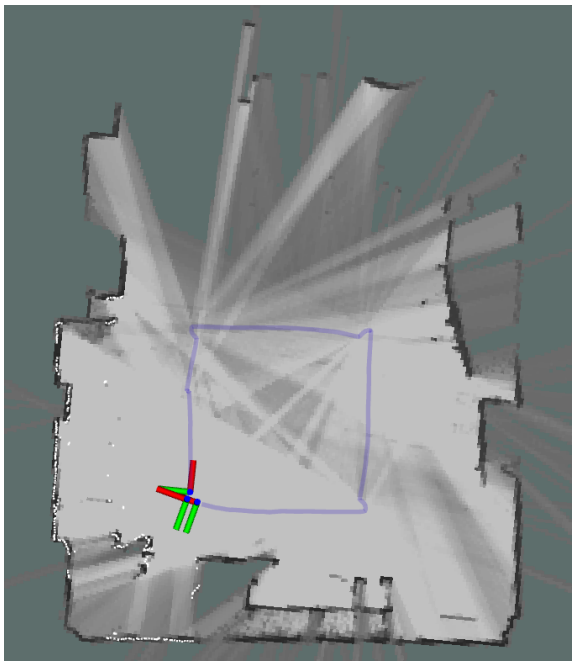

Figure 5.19: Squared trajectory performed using navigation with Cartographer for SLAM.

The obtained values from the procedure are shown in table 5.6, which includes the distance correction factor for both wheels, it can be concluded that the robot was compensating with the left wheel therefore affecting right side turns. The approximate value for the wheel base estimation is also accurate, the hand-measured value being $0.45m$

Obtained estimations	Values
Centre of gravity of the x error	-0.2585333333
Centre of gravity of the y error	0.0723333333
Left wheel compensation factor	1.004946
Right wheel compensation factor	0.995054
Wheelbase Estimation	$0.4535m$

Table 5.6: Values obtained from the UMBMARK method.

5.4 Chapter Summary

Testing in this chapter aimed at first validating the Localization and Mapping algorithm, both the quality of the SLAM output and the performance of the Raspberry Pi in this situation. Afterwards, the algorithm was tested in different settings and finally it was tested in the final prototype. Most robot function's were successfully tested, although the GNSS' impact on the SLAM is unknown due to hardware malfunction, and no useful large scale maps were obtained due to the lack of correct odometry, which was being addressed when the platform became unusable due to mechanical complications.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Following the identified necessity for cost-effective robotic platforms in agricultural settings, by INESC TEC, a robotic platform was developed with cost-effective hardware which is intended to perform robustly in steep-slope vineyards. The robotic platform also possesses planning capabilities, allowing for the performance of various tasks such as monitoring or be equipped with a small robotic manipulator for mechanical jobs. The platform is also small enough and it's localization method sufficient for entering the Field Robot Event competition.

It was designed and developed a small-sized, cost-effective robotic platform with basic sensors and processors which is able to fulfill the mapping, localization and navigation. The proposed objectives and features are available chapter 1, the cost analysis in section 4.6 and the verification of the mechanical capacity in section 4.4.1. Chapter 5 aims to verify the task fulfillment and shows that the platform should be able to perform planned missions.

A comparison of the performance of two SLAM state-of-the-art algorithms is provided in different settings, along as a validation of Cartographer SLAM on agricultural settings and in cost-effective hardware. The chosen algorithm Cartographer further explored throughout the work. A module for fusing GPS data with 2D trajectory estimation was developed and currently waiting for official developer review.

Three software packages were developed for this robot: An application based on the CANopen protocol was implemented to control the motors, along with an interface that receives velocity commands and outputs them to the motors; A navigation module to execute missions was fully developed and calibrated, where the robot preforms linear trajectories from waypoint to waypoint; And an user interface with integration of the application QGroundControl using an implementation of the MAVLINK protocol, allowing robot monitoring and mission planning.

6.2 Limitations and Future Work

Building a robust robotic solution from scratch is an extensive process, and unfortunately the time constraints limited full integration and testing. Although all the independent nodes were developed and tested successfully, the integration phase had to be cut short. As such, the future work of this robotic platform is tied to it's current limitations. The limitations are therefore addressed first below.

The odometry quality in the developed system is poor, mainly due to problems in the mechanical symmetry of the locomotion system. Besides, the locomotion system with a flat swivel wheel is not the most appropriate for harsh terrains. A robust mechanical project is necessary for a cost-effective robot in order to minimise errors that may arise, and are not manageable by the limited solution.

The used SLAM algorithm has a slow monotonic increase in memory usage which grows with map size. No direct measures of this RAM usage are possible, since it heavily depends on the setting and the map it draws, so no estimation is provided. The Raspberry Pi 3 B only has $1GB$ of RAM, with the occupied memory at robot startup around $260MB$, meaning that there is a limit to how long the Raspberry Pi is able to perform before running out of memory. Also, for additional tasks that require heavy processing, an additional dedicated processor is required.

No testing with GNSS data were performed, due to the hardware's malfunction.

The system currently doesn't provide any obstacle detection/avoidance module.

To address these limitations, the following future work is suggested:

Calibration of the systematic errors Although the routine UMBMARK was performed, the robot broke down before verifying the calibration's performance. Odometry calibration is essential to the SLAM process since the IMU and LIDAR are not sufficient for a good enough estimation in agricultural settings. Another possibility is to alter the traction method to a system more appropriate to the settings, such as threads.

Testing with swap memory To address the long duration, one possible workaround is the usage of swap memory from the flash memory card, but this may negatively impact the performance of the algorithm. Additional testing should be done in these conditions to additionally verify if the processor is capable of the indicated tasks.

Integration and testing of GNSS data Although the GNSS receiver has a low precision, the system should be tested with GNSS data to evaluate it's impact on the localization estimation, specifically in the global component since it can significantly aid in tying maps in large locations.

Development of an obstacle detection/avoidance module Due to the unstructured nature of the setting, a module for obstacle avoidance should be developed to deal with any obstacles, like rocks, that can appear in the defined trajectories and should be contoured or the robot should stop.

Testing and calibration in steep-slope vineyards Testing the system in a vineyard is a must for the final requirement verification of the platform. It is estimated that the localization module should be then calibrated to the setting, possibly reducing the odometry impact and increasing the laser scanner's, due to the naturally feature-rich environment.

References

[1] FAO, *World Food and Agricultrure Statistical Pocketbook 2018*. http://www.fao.org/documents/card/en/c/CA1796EN/ [Online; accessed June 11, 2019].

[2] R. Baxter, N. Hastings, a. Law, and E. J. Glass, "A future view of precision farming," *Animal Genetics*, vol. 39, no. 5, pp. 561–563, 2008.

[3] J. J. Roldán, J. del Cerro, D. Garzón-Ramos, P. Garcia-Aunon, M. Garzón, J. de León, and A. Barrientos, "Robots in Agriculture: State of Art and Practical Experiences," *Service Robots*, 2018.

[4] U.-r. White, T. Duckett, S. Pearson, S. Blackmore, B. Grieve, W.-H. Chen, G. Cielniak, J. Cleaversmith, J. Dai, S. Davis, C. Fox, P. From, I. Georgilas, R. Gill, I. Gould, M. Hanheide, A. Hunter, F. Iida, L. Mihalyova, S. Nefti-Meziani, G. Neumann, P. Paoletti, T. Pridmore, D. Ross, M. Smith, M. Stoelen, M. Swainson, S. Wane, P. Wilson, I. Wright, and G.-Z. Yang, "Agricultural Robotics: The Future of Robotic Agriculture," 2018.

[5] R. Barth, J. Hemming, and E. J. van Henten, "Design of an eye-in-hand sensing and servo control framework for harvesting robotics in dense vegetation," *Biosystems Engineering*, vol. 146, pp. 71–84, 2016.

[6] J. M. Mendes, F. N. Dos Santos, N. A. Ferraz, P. M. Do Couto, and R. M. Dos Santos, "Localization Based on Natural Features Detector for Steep Slope Vineyards," *Journal of Intelligent and Robotic Systems: Theory and Applications*, pp. 1–14, 2018.

[7] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Scituate, MA, USA: Bradford Company, 2004.

[8] H. R. Everett, *Sensors for Mobile Robots: Theory and Application*. Natick, MA, USA: A. K. Peters, Ltd., 1995.

[9] K. Ohno, T. Tsubouchi, B. Shigematsu, and S. Yuta, "Differential GPS and odometry-based outdoor navigation of a mobile robot," *Advanced Robotics*, vol. 18, no. 6, pp. 611–635, 2004.

[10] Wikipedia, the free encyclopedia, "A diagram of a pinhole camera," 2008. https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg [Online; accessed June 11, 2019].

[11] J.-p. Tardif, "Monocular Visual Odometry in Urban Environments," *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 22–26, 2008.

[12] J. Courbon, Y. Mezouar, L. Eck, and P. Martinet, "A generic fisheye camera model for robotic applications," *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, no. c, pp. 1683–1688, 2007.

[13] D. M. Little and J., "Using Real-Time Stereo Vision for Mobile Robot Navigation," *Autonomous Robots*, vol. 8, no. 2, pp. 161–171, 2000.

[14] J. Borenstein, L. Feng, P. D. K. Wehe, Y. Koren, G. Students, Z. Fan, B. Holt, and B. Costanza, "UMBmark - A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots," 1995.

[15] G. Antonelli, S. Chiaverini, and G. Fusco, "A calibration method for odometry of mobile robots based on the least-squares technique: Theory and experimental validation," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 994–1004, 2005.

[16] N. Doh, H. Choset, and W. Chung, "Accurate relative localization using odometry," *2003 IEEE International Conference on Robotics and Automation*, pp. 1606–1612, 2003.

[17] N. Roy and S. Thrun, "Online self-calibration for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 3, pp. 2292–2297 vol.3, May 1999.

[18] A. Martinelli, N. Tomatis, and R. Siegwart, "Simultaneous localization and odometry self calibration for mobile robot," *Autonomous Robots*, vol. 22, no. 1, pp. 75–85, 2007.

[19] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, jun 2004.

[20] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 15–22, 2014.

[21] R. Mur-Artal, J. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[22] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 903–910 Vol. 1, Oct 2005.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[25] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[26] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, nov 2004.

[27] K. Lingemann, A. Nüchter, J. Hertzberg, and H. Surmann, "High-speed laser localization for mobile robots," *Robotics and Autonomous Systems*, vol. 51, no. 4, pp. 275–296, 2005.

[28] S. A. Hiremath, G. W. van der Heijden, F. K. van Evert, A. Stein, and C. J. ter Braak, "Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter," *Computers and Electronics in Agriculture*, vol. 100, pp. 41 – 50, 2014.

[29] M. Pinto, H. Sobreira, A. P. Moreira, H. Mendonça, and A. Matos, "Self-localisation of indoor mobile robots using multi-hypotheses and a matching algorithm," *Mechatronics*, vol. 23, no. 6, pp. 727–737, 2013.

[30] Y. Liu and Y. Sun, "Mobile robot instant indoor map building and localization using 2D laser scanning data," *Proceedings 2012 International Conference on System Science and Engineering, ICSSE 2012*, pp. 339–344, 2012.

[31] N. E. Pears, "Feature extraction and tracking for scanning range sensors," *Robotics and Autonomous Systems*, vol. 33, no. 1, pp. 43–58, 2000.

[32] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[33] S. B. J. Pfister, S.T.; Roumeliotis, "[ieee ieee international conference on robotics and automation. ieee icra 2003 - taipei, taiwan (14-19 sept. 2003)] 2003 ieee international conference on robotics and automation (cat. no.03ch37422) - weighted line fitting algorithms for mobile robot map building and efficient data representation," vol. 1, 2003.

[34] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, pp. 381–395, 1981.

[35] T. Pavlidis and S. L. Horowitz, "Segmentation of plane curves," *IEEE Transactions on Computers*, vol. C-23, pp. 860–870, Aug 1974.

[36] Y. Zhao and X. Chen, "Prediction-based geometric feature extraction for 2D laser scanner," *Robotics and Autonomous Systems*, vol. 59, no. 6, pp. 402–409, 2011.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, 1960.

[38] T. Nishizawa, A. Ohya, and S. Yuta, "An implementation of on-board position estimation for a mobile robot-ekf based odometry and laser reflector landmarks detection," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 395–400 vol.1, May 1995.

[39] L. Teslić, I. Škrjanc, and G. Klančar, "Ekf-based localization of a wheeled mobile robot in structured environments," *Journal of Intelligent & Robotic Systems*, vol. 62, no. 2, pp. 187–203, 2011.

[40] W. Burgard, A. Derr, D. Fox, and A. B. Cremers, "Integrating global position estimation and position tracking for mobile robots: the dynamic markov localization approach," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, vol. 2, pp. 730–735 vol.2, Oct 1998.

[41] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[42] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," 11 2002.

[43] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225–234, 2007.

[44] C. Forster, M. Pizzoli, and D. Scaramuzza, "Air-ground localization and map augmentation using monocular dense reconstruction," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3971–3978, Nov 2013.

[45] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collison-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, 1979.

[46] H. Dong, W. Li, J. Zhu, and S. Duan, "The path planning for mobile robot based on voronoi diagram," in *2010 Third International Conference on Intelligent Networks and Intelligent Systems*, pp. 446–449, Nov 2010.

[47] P. K. Gaonkar, A. DelSorbo, and K. S. Rattan, "Fuzzy navigation for an autonomous mobile robot," in *NAFIPS 2005 - 2005 Annual Meeting of the North American Fuzzy Information Processing Society*, pp. 412–417, June 2005.

[48] H. Maaref and C. Barret, "Sensor-based fuzzy navigation of an autonomous mobile robot in an indoor environment," *Control Engineering Practice*, vol. 8, no. 7, pp. 757 – 768, 2000.

[49] F. Lingelbach, "Path planning using probabilistic cell decomposition," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, pp. 467–472 Vol.1, April 2004.

[50] N. Buniyamin, W. W. Ngah, N. Sariff, and Z. Mohamad, "A simple local path planning algorithm for autonomous mobile robots," *International journal of systems applications, Engineering & development*, vol. 5, no. 2, pp. 151–159, 2011.

[51] S. Quinn Marlow and J. Langelaan, "Dynamically sized occupancy grids for obstacle avoidance," 08 2010.

[52] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, March 1985.

[53] H. University, *Field Robot Event Proceedings*. http://www.fieldrobot.com/event/index.php/downloads/previous-proceedings/ [Online; accessed June 11, 2019].

[54] P. Biber, U. Weiss, M. Dorna, and A. Albert, "Navigation system of the autonomous agricultural robot bonirob," in *Workshop on Agricultural Robotics: Enabling Safe, Efficient, and Affordable Robots for Food Production (Collocated with IROS 2012), Vilamoura, Portugal*, 2012.

[55] W. Bangert, A. Kielhorn, F. Rahe, A. Albert, P. Biber, S. Grzonka, S. Haug, A. Michaels, D. Mentrup, M. Hänsel, *et al.*, "Field-robot-based agriculture:"remotefarming. 1" and "bonirob-apps"," *VDI-Berichte*, vol. 2193, no. 439-446, pp. 2–1, 2013.

[56] *VINBOT*. http://vinbot.eu/?lang=pt [Online; accessed June 11, 2019].

[57] *VINEyard ROBOT*. http://www.vinerobot.eu [Online; accessed June 11, 2019].

[58] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, Nov 2011.

[59] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, May 2016.

[60] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, "2d slam quality evaluation methods," 08 2017.

[61] F. Mendes, "Cartographer gps data integration pull request." https://github.com/googlecartographer/cartographer/pull/1591, 2019. [Online, accessed 23 June 2019].

[62] Nanotec, *C5-E CANopen/USB Technical Manual*. Version 2.0.0.

[63] F. Mendes, "twist_can_package," 2019. https://github.com/Conhokis/twist_can_pkg [Online; accessed June 11, 2019].

[64] M. Drwięga, "bosch_imu_driver," 2019. [Online, accessed 11 June 2019.

[65] E. Perko, "nmea_navsat_driver," 2017. https://github.com/ros-drivers/nmea_navsat_driver [Online; accessed June 11, 2019].

[66] C. Rockey, "hokuyo_node," 2014. https://github.com/ros-drivers/hokuyo_node [Online; accessed June 11, 2019].

[67] *QGroundControl*. http://qgroundcontrol.com/ [Online; accessed June 11, 2019].

[68] F. Mendes, "qgc_interface_ros," 2019. https://github.com/Conhokis/qgc_interface_ros [Online; accessed June 11, 2019].

[69] J. O'Quin, "geodesy," 2018. https://github.com/ros-geographic-info/geographic_info [Online; accessed June 11, 2019].

[70] "Mavlink mission protocol," 2007. https://mavlink.io/en/services/mission.html [Online; accessed June 11, 2019].