MSc in Computer Science and Engineering
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

POLITECNICO
MILANO 1863

An Experiment in Autonomous Vineyard Navigation: The GRAPE Project

AI & R Lab

Supervisor: Prof. Matteo MATTEUCCI
Co-supervisor: Ing. Gianluca BARDARO

Master Graduation Thesis by:
Pietro ASTOLFI
student id 841044

Academic Year 2016-2017

Abstract

Field robotics is a fast developing research field, in particular precision agriculture. The GRAPE project is an Echord++ robotic experiment aimed at the use of a mobile robot for automatic pheromone dispenser distribution in vineyards. This thesis illustrates the autonomous navigation system of such robot. For the specific scenario of the vineyard navigation, there not exists a real state of the art, so we adapted techniques that have been designed for different problems, in particular classical methods for navigation in indoor environments. The vineyard environment is challenging because of many variability factors such as weather, soil and vegetation. These factors hinder the indoor methods introducing noise in the robot perceptions. To solve this problem, we propose a specific navigation system that takes advantage of multiple sensors: wheel encoders, IMU and GPS to filter the environment noise and accurately estimate the robot odometry. In addition, the system exploits a LIDAR sensor to localize the robot through AMCL algorithm and to map the vineyard using SLAM techniques. We tested the system in simulation where it obtained very good results which have been confirmed during a field test in a real vineyard.

Estratto

Al giorno d'oggi la robotica outdoor (field robotics) si sta sviluppando sempre di più. In particolare l'agricoltura di precisione rappresenta un'applicazione importante di essa. Il progetto GRAPE tratta l'utilizzo di un robot mobile per la distribuzione automatica di dispensatori di feromoni nei vigneti. Questa tesi illustra lo sviluppo del sistema di navigazione autonoma di tale robot. Per lo scenario specifico della navigazione in un vigneto non esiste un vero e proprio stato dell'arte, perciò abbiamo adattato a questo ambiente tecniche che sono pensate per problemi diversi. In generale, gran parte dei metodi conosciuti in robotica per la navigazione sono ideati per ambienti indoor. Il vigneto presenta molti fattori di variabilità come le condizioni atmosferiche, il suolo o la vegetazione, che rappresentano insidie per i metodi indoor classici, poichè generano molto rumore nelle percezioni del robot. Per risolvere questo problema, in questa tesi, abbiamo proposto uno specifico sistema di navigazione che sfrutta diversi sensori: encoders delle ruote, IMU e GPS per filtrare il rumore dovuto all'ambiente e quindi stimare accuratamente il percorso effettuato dal robot. Inoltre il sistema utilizza un sensore LIDAR per localizzare il robot con l'algoritmo Adaptive Monte Carlo Localization, e per costruire una mappa del vigneto utilizzando metodi di SLAM. Abbiamo testato il sistema in simulazione ottenendo risultati molto buoni che abbiamo successivamente validato in un vero vigneto.

Ringraziamenti

Indice

Elenco delle figure

Elenco delle tabelle

1. Introduction

The application of autonomous robots in rough and unstructured environments has increased exponentially over the last years, so that it has defined a new branch of robotics called field robotics. In particular one of the fields that is most developed recently is precision agriculture. Indeed in agriculture, robots are not only used for crop monitoring, like aerial inspection for growth control, but they are taking a key role in the daily life of farmers and producers. Heavy tasks like pruning, seeding or even precise harvesting are the target topics of these new robots.

The GRAPE project has born to accomplish one of these precision tasks: the distribution of pheromone for integrated pest control in vineyards. It is funded by Echord++ program from European Commission. This thesis carries out a specific part of the GRAPE project: the design of autonomous navigation system of the GRAPE robot. In the title we defined it as "an experiment" since Echord++ uses this terms to define small-scale research projects with a maximum duration of 18 months aimed at verifying the readiness of a technology on the field

Autonomous outdoor navigation in a vineyard is a quite different problem with respect to similar categories of problems, like indoor navigation or autonomous on-road driving. Indeed outdoor scenarios contain some very important destabilizing factors which do not permit to use standard navigation methods as they are. Specifically, we can identify three main factors which are sources of variability: the soil, the weather and the vegetation. Each of these introduces noise in the data perceived from the robot sensors. For example the soil affects the mobility of the robot, and it is responsible for wheels slippage and robot platform wobbling. Such effects influences the perception done by the wheel encoders, by the IMU and by the LIDAR. The

weather, instead, alters the satellite reception and the quality of the LIDAR acquisitions. Finally the vegetation, depending on the season increase or decrease the volume of the vines and the quantity of grass and weeds. This hinder the navigation of the robot in terms of number and size of obstacles.

1.1 Thesis contribution

To solve the variability of the vineyard environment this thesis proposes a multi-sensors navigation system that takes advantage of a robust sensor fusion framework, ROAMFREE, which provides tools to merge different sensors in the estimation of the odometry of a robot. The sensors merged are: wheel encoders, IMU and GPS.

Besides using GPS information the system performs the localization using two different approaches based on the task: if the task is the autonomous navigation the localization is done exploiting a particle filter in the form of Adaptive MonteCarlo Localization (AMCL) algorithm, otherwise if the task is to map the vineyard the localization takes places simultaneously with the mapping of the robot for the autonomous navigation task and exploits a technique, SLAM, able to build a map and simultaneously locate the robot in that map when the task is to map the vineyard. In both cases the robot acquires information about the obstacles and the environment structure using a LIDAR sensor.

The navigation system has been tested in a simulation environment created ad-hoc, which reproduces both the vine plants and the vineyard ground. We have obtained excellent results both in mapping and in autonomous navigation in this simulation environment. Afterwards the system has been installed and configured on a real robot, the GRAPE robot, which is a four wheeled skid-steering mobile robot based on the Husky platform.

The robot has been ran in a real vineyard to validate the performance of the navigation system in real case. During tests this latter has continuously provided the right localization of the robot. Further, the robot has been able to correctly map the vineyard and then to autonomously navigate using that map. For the mapping task the system has been tested with three different laser based SLAM tools: Gmapping, Google's Cartographer and KartoSLAM. Gmapping outperformed the other approaches, since it proved to be more reliable in providing accurate maps.

1.2 Structure of Thesis

The thesis is structured as follows.

- In Chapter 2 we discuss and explain the challenges for autonomous robot navigation due to the vineyard environment. In particular, we first introduce the GRAPE project context, in section 2.1, then we analyze the issues which are present in vineyard robot navigation and for which this thesis propose a solution, section 2.2. Finally we show other examples of vineyard robot in section 2.3

- In Chapter 3 we describe all the tools that we adopt in our solution and the theoretical concepts to understand them. We start introducing the standard approaches for the state estimation in section 3.1. Then we explain the odometry estimation basics and we list the sensor fusion tools that we use to compute it, respectively in section 3.2 and in section 3.3. Finally we outline the most known methods for SLAM and for localization in section 3.4 and in section 3.5.

- In Chapter 4 we present the navigation system architecture we propose. We first depict the final architecture, in section 4.2 and then we analyze singularly the sub-systems composing it from section 4.3 to section 4.7.

- In Chapter 5 we introduce the simulation environment in section 5.2 and then we describe and comment the simulation tests and results in the three tasks of odometry estimation and sensor fusion in section 5.3, mapping in section5.4 and autonomous navigation in section 5.5.

- In Chapter 6 we shows the vineyard used for validation tests in section 6.1 and we discuss the obtained results for mapping in section 6.2 and autonomous navigation in section 6.3.

- In Chapter 7 we summarize the obtained results and we propose some possible futures improvements for the system.

- Appendix A introduces ROS, outlining its structure and its main features.

- Appendix B reports the technical data about the sensor adopted for GRAPE robot during the vineyard tests.

2.
Challenges in Field Robotics: the vineyard case

In the Introduction we have mentioned some of the challenges that we faced during the thesis. In this chapter we first describe the GRAPE project and then we focus on the identification and explanation the problems due to the vineyard environment.

2.1 The GRAPE Project

The application of robotics as a support tool for farmers or automated systems for agricultural tasks can offer the necessary step change in farming production in order to meet the future needs of an increasing world population, 34% by 2050 according to FAO[1]. GRAPE project aims at contributing to that technological breakthrough by developing a mobile robotic system endowed with a robotic arm able to perform precise agricultural tasks at plant level for vineyards.

GRAPE robotic concept is composed by four main technological components designed and integrated to monitor plants health in vineyards and to apply a biocontrol mechanism consisting of pheromone dispensers for plague control. These results will be validated in real scenarios covering a predefined range of scenarios in France, Italy and Spain, the three EU countries with the largest wine production.

The four technological components of the GRAPE projects are:

[1]Food and Agriculture Organization of the United Nations.

- Vineyard navigation module based on advanced localization and mapping capabilities and path planning techniques considering terrain characteristics and kino-dynamic constraints to enhance safety and robustness.
- Plant health monitoring module implementing advanced perception capabilities for plant detection in highly unstructured and geometrically variable scenario and plant health assessment for early detection of problems.
- Precise manipulation and deployment for a targeted pheromone dispenser distribution.
- User friendly operational interface enabling robot teleoperation, data visualization and reporting.

A ground robot for plague control related tasks shall be able to navigate along the rows of vines typically found in this type of crops. The variety of terrains and typology of crops makes the definition of an application scenario extremely relevant for a correct requirements elicitation. This project belongs to Echord++ Program from European Commission. Vineyard protection becomes one of the main issues for the producers. Control of plagues, fungi and other threats are recurrent tasks in winery. This project is focused on the improvement of the plague control tasks, specifically on the installation of pheromone dispensers for matting disruption. The task of dispenser distribution is currently done manually where the operators walk through the vineyard hanging the dispensers in a regular distribution pattern (approximately one dispenser every 5-6 plants). There are different shapes available for the dispensers, depending on the brand, which is something to take into account for our manipulation developments. In our case, we use the dispensers provided by Biogard2, which is a reference company in Europe. Regarding the specific location to deploy the dispenser, it must be placed in one of the main branches. 2.1 shows the shape and the location of the pheromone dispenser. The installation of dispensers should be performed early in the first moth flight (G1), which is to say from the end of March in the earliest areas until early April. In this period, the vineyard is just trimmed. This situation defines a particular scenario for dispenser deployment. The plants have no leaves and only the main branches (two horizontal) remain. The small branches are pruned. This particular situation have to be taken

Figura 2.1: Dispenser shape and location

into account in the use case development. The project is a collaboration between the Politecnico di Milano (Italy), the Eurecat research center (Spain) and Vitirover (France). The three entity should work on different task, in particular the Polimi team have to develop the navigation system and the manipulation part of the robot.

From a robotics perspective, there are two main aspects to take into account when analyzing the traversability challenges for a ground robot operating in vineyards: slope of the ground and terrain morphology. Depending on the region, the vineyard distribution can be very challenging. Although the plant distribution is always in rows, the crop can be located in a plain terrain or in the mountainside. We identify three types of crops:

- Flat crops, located in an almost flat field maybe following the contour of the hill but always in the same plane
- Hillside, with a slope less than 40% where the plants are organized in rows with a horizontal path between them
- Mountainside, where plants are located in a hillside with more than 40% of slope. Usually, in best cases, there is a small path between rows, 60cm at most, which allows the operator walking through the crop, in order to do manual work. If not, there is no horizontal path and robotic deployment becomes almost impossible.

Another important aspect in the performance of a ground robot in a vineyard is the type of terrain. The performance of the robot largely depends on the

type of soil. This can affect not only the navigation but also the battery consumption and motion control, among other aspects.

2.2 Robot navigation in vineyards

Field robotics is concerned with the automation of vehicles and platforms operating in harsh, unstructured environments. Field robotics encompasses the automation of many land, sea and air platforms in applications such as mining, cargo handling, agriculture, underwater exploration and exploitation, planetary exploration, coastal surveillance and rescue.

Field robotics is characterized by the application of the most advanced robotics principles in sensing, control, and reasoning in unstructured and unforgiving environments. The appeal of field robotics is that it is challenging science, involves the latest engineering and systems design principles, and offers the real prospect of robotic principles making a substantial economic and social contribution to many different application areas.

In general, field robots are mobile platforms that work outdoors, often producing forceful interactions with their environments and without human supervision. Many examples and other details about field robotics can be found at [1].

Field robotics is as much about engineering as it is about developing basic technologies. While many of the methods employed derive from other robotics research areas, it is the application of these in large scale and demanding applications which distinguishes what can currently be achieved in field robotics.

The autonomous navigation in a vineyard belongs to the category of the outdoor navigation that is one of the most challenging application of field robotics. In fact it has been described in many works already in the 90s. In particular there exists a work of Amit Singhal [2] that in 1997 identifies three main sub-problems relative to the unmanned outdoor navigation: the unstructured environments, moving obstacles and multiple sensors. Obviously the moving obstacles and the multiple sensors are also problems for the indoor autonomous navigation but in a less accentuated manner. The majority of the known techniques and algorithms in robotics are created for indoor scopes, thus adopting them for outdoor tasks is not simple and requires the usage some tricks since there are a lots of uncertainties and noise due to the environment.

(a)

(b)

Figura 2.2: Examples of a real vineyard and a rough terrain; (a) illustrates a common Italian vineyard (b) shows the irregularity of the terrain

2.2.1 Unstructured environments

For what concerns the unstructured environments, our robot has to work in vineyards that even if they are different in each geographical zone always maintain a similar skeleton: the vines are disposed in straight lines that have a fixed separation distance among them, the vines belonging to the same line are equidistant and often, in a vineyard, there are chunks of lines that share the same length. Some examples of real vineyards can be seen in the 2.2. Thus, we dont have a lack of structure, but we must note that the described structure is very repetitive and this represent a drawback especially during

navigation tasks. In fact, its harder for the robot to localize itself in points of the vineyard which are very similar to others. Furthermore, if on one hand we have an almost fixed structure on the other we have the presence of some instability factors such as the ground, the weather and the vegetative state of the environment. The ground problem is one of the most difficult to solve due to its high variability. Indeed, the terrain is bumpy (see Figure 2.2b) and its consistency changes based on the geolocation and the weather so it affects always differently the movements of the robot. In particular it influences the slippage of the wheels, that increases the unreliability of the wheel encoders acquisitions and it makes the robot platform wobble that increments the noise of the measurements coming from the sensors fixed on it (except for the GNSS). The vegetative state of the environment changes with the seasons and can completely transform the aspect of it. During the late autumn and the winter, the nature is bare, the vine plants havent leaves and the ground has few grass and weeds. Instead during the others months, the surroundings are green, so the vines are full of leaves and in certain periods are also full of grapes while the terrain presents a lot of grass. Obviously in this period many maintenances are done to make the vineyard clean and to permit to work in it, but often the result is approximately clean and so many obstacles remain. The problem created by the vegetation of the vineyard its a visibility problem for the robot. In fact, the presence of leaves and weeds near the plants increases the size of the obstacles and make them less definite (its harder to identify the single vines) to the robots eyes. Furthermore, if there are some high weeds (at least as high as the LIDAR sensor level), they become new obstacles for the robot and makes the autonomous navigation more difficult (they mislead the planner when it draws the path). Finally, the weather impact is the more relative, since obviously the robot is expected to work with acceptable atmospheric conditions, thus no rainfall and not too much wind. Nevertheless, the weather, also when it is acceptable, can influence the satellite reception or the LIDARs acquisitions quality, thus in general the accuracy of the sensors.

2.2.2 Moving obstacles

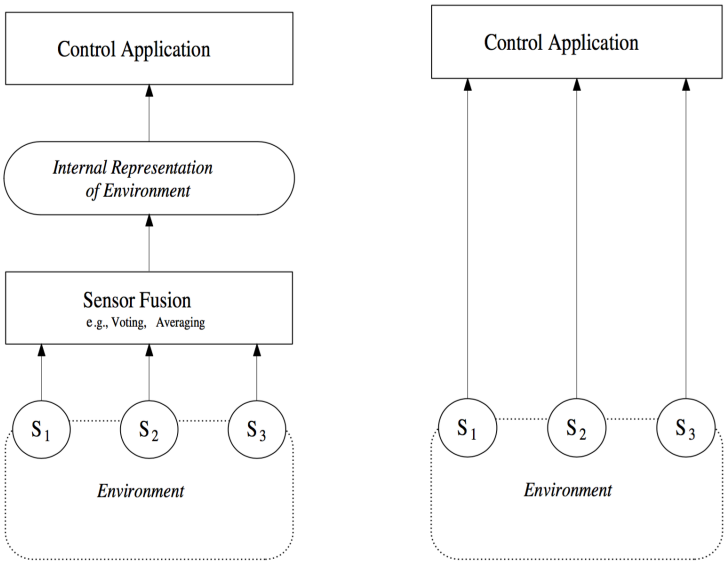
Then we talk about the presence of moving obstacles and it is clear that in our case this problem is the one that is the most similar to indoor navigation. For many unmanned outdoor robots, the scenario is completely different

from our one, i.e. autonomous cars, and the moving obstacles represent a very sensitive issue. However, in all scenarios the problem is faced with robust and, sometimes complex, local planner, thus planning algorithm that are designed to reacts in real time to unexpected changes. Thinking to our scenario there are moving obstacles when it happens that a living being hinds the path that the robot is following. This is a quite rare case in a vineyard since it doesnt guest much animals and humans are not expected to be present in front of the robot while it runs. These kind of events are even more probable in an indoor scenario. What can happen in our case is that the atmospheric conditions create some new unexpected obstacles, i.e. the wind can carry some small objects.

2.2.3 Multiple sensors

Finally, the multiple sensors problem. First of all, we have to make a brief explanation of what we mean when we talk about this problem. In fact, in the literature there exists a clear distinction between multi-sensor integration and sensor fusion. The differences between these two has been well explained in a work of Wilfried Elmenreich [3] and we summarize it here in the 2.3.

In our work we will always refers to sensor fusion and for this reason we report the definition given by Elmenreich. Sensor Fusion is the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually. Another definition of sensor fusion can be found in the article of Kam et al. [4], but its more mathematical and so we prefer to not bring it here. Now that we have a definition, we can start explaining why the sensor fusion problem is adapt to our scenario and what are the issues in it. Looking at the definition of Elemenreich is simple to understand that sensor fusion is needed to improve the accuracy of the state estimation of a robot. In fact, the simultaneous usage of multiple sensors allows the robot to have a more complete knowledge of the environment and so it enriches its internal world representation. Robots differently from humans dont need a global vision of the surrounding environment, but they need the information necessary to complete the tasks for which they are designed. Thus, it is important to choose the right sensors to collect only interesting data (we explain our choices in the chapter 4), in the case of mobile unmanned navigation and especially in our case is fundamental to have more sensors. There



(a) Sensor fusion (b) Multisensor integration

Figura 2.3: Block diagram of sensor fusion and multi-sensor integration

are many reasons of this need and some of them can be retrieved in the problems that we already described. We can say that the multiple sensors problem is, in part, a consequence of the other two, since it belongs to the solutions of them. Thus, solving this problem means also to solve a portion of the others. In general, the advantages that sensor fusion can leads to are the following:

- Robustness and reliability: Multiple sensor suites have an inherent redundancy which enables the system to provide information even in case of partial failure.
- Extended spatial and temporal coverage: One sensor can look where others cannot respectively can perform a measurement while others cannot.
- Increased confidence: A measurement of one sensor is confirmed by measurements of other sensors covering the same domain.
- Reduced ambiguity and uncertainty: Joint information reduces the set of ambiguous interpretations of the measured value.

- Robustness against interference: By increasing the dimensionality of the measurement space (e.g., measuring the desired quantity with optical sensors and ultrasonic sensors) the system becomes less vulnerable against interference.

- Improved resolution: When multiple independent measurements of the same property are fused, the resolution of the resulting value is better than a single sensors measurement.

There exist many methods that face the sensor fusion problem and based on the approach they give a different mathematical definition of the problem. Among all the approaches there are three types that have been mostly used in the last few years, thanks to their performance and reliability. These are two Bayesian approaches: one Gaussian, the Kalman filter and one non-parametric, the Particle filter; and the graph-based approach. In our work we will test all these and in the end we will combine two of them in our proposed solution. The differences between these types of state estimation techniques and their description is explained in chapter 3 3. After this brief digression on the possible solutions to the sensor fusion problem, we can resume explaining what are the intrinsic issues of using more sensors. A first and basic consideration is that every real sensor is affected by some noise, so it provides measurements containing an error. Usually this error is composed by two parts: one expected and one unexpected or unpredictable, using more technical words they are, respectively, the statistical bias and the standard deviation. For all commercial sensors there exists a descriptive data-sheet in which the value of the expected error is specified, since its an error that is intrinsic of the sensor for construction reasons. Thus, the knowledge of the bias can be exploited, with a simple post processing, to have better measurements in the state estimation. Instead, it remains unknown the unexpected part of the error since its dependent from external factors, in our case the ones coming from the outdoor environment, and thus it has to be identified and reduced in real-time. In order to notice the entity of the unexpected error is fundamental to have more sources of data which provide an inherent measurement, thus more sensors collecting information on the same domain, but from a different point of view. Its consequent that the redundancy is a very important feature for sensor fusion, obviously in the convenience limits. In fact, Its implicit that it is not meaningful to have too much sensors measuring the same variable both for the economic side and for

the computational complexity side. Integrate and merge acquisitions coming from different sources is a very hard work since all the differences have to be uniformed. What changes between the various sensor is: the frequency with which they acquire data, the unit of measure and their uncertainty. The amount of uncertainty is the most influent factor for the greatness of a sensor fusion system, indeed there exists a well-known work of Fowler [5] in which he comments the military systems and in particular he spends some words about the usage of multiple sensors. He says: Massaging a lot of crummy data doesnt produce good data; it just requires a lot of extra equipment and may even reduce the quality of the output by introducing time delays and/or unwarranted confidence. Thus, it is fundamental to have sensors that provide good quality data and for this aim it is often necessary to calibrate them before start executing the desired tasks. The importance of the calibration phase should not be underestimated, especially for certain type of sensors which otherwise produce harmful data. In case where its not possible to successfully complete this phase it is better to discard at all the information coming from the interested sensors.

In outdoor environments the sensors used for the planar navigation must be able to perceive the robot movements and the obstacles. The clearer is the data acquired the simpler is the localization and thus the navigation of the robot. For these reasons the sensors, usually, adopted are: GPS, wheel encoders, IMU (gyroscope, magnetometer, accelerometer) and laser range finders. The GPS sensor is very important for outdoor robot, it allows to have a real geolocation and thanks to the the last year improvement it is also able to provide precise measure through the RTK adjustments. The curse of the GPS is always the unreliability since it depends from the satellite reception. So, building systems that are heavily based on the GPS work is never a good choice, even in case like our one in which there is no signal reflection problem and there are not very high barriers. Furthermore, the GPS (without RTK correction) also suffers of multiple paths problem, thus in the vineyard it can leads to jump among different lines. The wheel encoders are really precise in the measurements, since they are internal sensor, but the problem here is, as we stated above, the wheels slippage. This latter has a different influence depending on the kinematic model of the robot (wheels disposition). The gyroscope and the accelerometer are highly influenced by the robot stresses due to the irregularity in the terrain. The magnetometer is a very delicate sensors, in the sense that it is really sensitive to more factors:

the geolocation, the presence of magnetic distortion due to metals or cables around it. Finally the LIDAR is often quite stable, but it is measurement depends on the robot platform inclination and thus it is influenced by the rough ground. Further it has also some influence by the weather that we already cited.

2.2.4 Simulation problems

Simulation is the imitation of the operation of a real-world process or system over time; in robotics it corresponds to a software able to reproduce as real as possible the robot and its working environment. This means to reproduce all the physical behaviors of each component of the desired world and in particular of the robot model. Furthermore, thanks to the growing technologies in computer graphics, the simulators are also able to show graphically what is happening during the tests.

On the other hand, reproducing graphically and physically an entire environment is a very complex operation and thus the simulator results to be a very heavy software respect to the computational power of common PCs. The weight of a simulation depends on the quantity and the quality of the objects that have to be reproduced and based on these values the simulation is more or less flowing.

In our thesis we need to simulate both the vineyard and the GRAPE robot. In particular the simulation of the vineyard is not a simple task due to its high variability. Indeed the terrain and the plants have not a constant shape, thus reproducing them with a 3D model introduces a relevant approximation error. In addition, it is not possible to reproduce the weather condition which another variability factor of the environment. We can assert that, the simulated environment is always a simpler approximation of the reality and in this scenario the approximation is even higher with respect to other scenarios. Despite these approximations the models representing the outdoor environment are very complex, both physically and graphically, thus they can lead to some fluency problem during the simulation or to inconsistency problem (if the physical model has some lacks).

Figura 2.4: the "Dassie" prototype

2.3 Other similar projects

In the last years in the agricultural robotics is rapidly increasing the number of unmanned robots. We report here some examples of other projects that like our one have to deal with the field robotics problem. In particular we describe robots that has been developed for the vineyard work. Almost all are not already in commerce and in general we don't have access to much information about their navigation systems.

The first example of a similar robot is the one developed in SouthAfrica by a collaboration between the CSIR (Council for Scientific and Industrial Research) and the Stellenbosch University. The aim of this project is to create a cost-effective platform to inspect and monitor horticultural crops on local farms. The robot prototype is called Dassie 2.4, is quite agile and can easily move around in the vineyard. A few sensors have been fixed to the platform and include a laser (LIDAR) scanner as well as high definition cameras facing to the front and sideways. The robot can also pull an electromagnetic induction sensor behind it to be able to map soil differences. All measurements that it acquire are streamed to an online computer able to process the information. After the prototype development it has been tested in a vineyard and thus the sensors have been configured to estimate grape yield

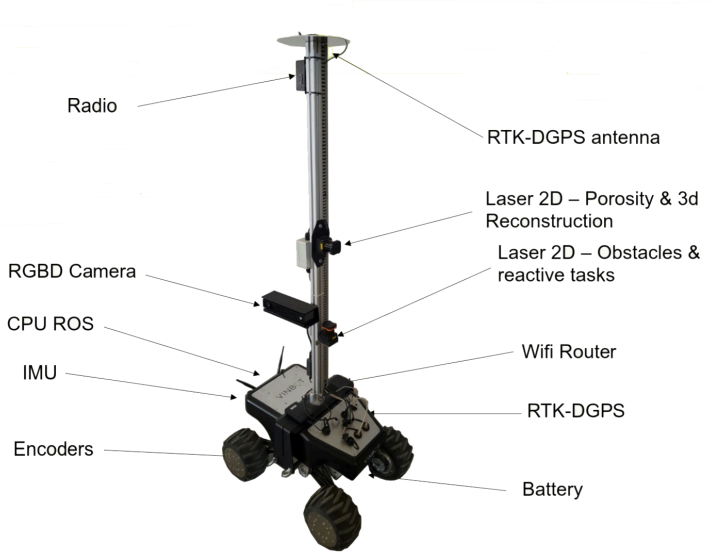


Figura 2.5: Vinbot robot prototype

and to monitor plant growth and canopy health. Currently, it is able to navigate autonomously through the vineyard, using CSIR-developed data-sensor fusion techniques to combine the data from the different sensors. Using the CSIR-designed data-fusion algorithm, it is also able to perform path planning, obstacle avoidance and lane following. Unfortunately we don't have information about the sensor fusion algorithm and the navigation system structure, thus we cannot compare our intuitions with their ones.

The second example that we report is another vineyard robot built under an European project: VinBot.

The VinBot project is a European project: "Autonomous cloud-computing vineyard robot to optimise yield management and wine quality" in which there is an all-terrain robot that monitors vineyard criteria essential to successful yield management by accurately estimating, by means of computer vision: the quantity of leaves, the quantity of fruit and the grapes exact location on the vine.

The VinBot consists of an autonomous wheeled mobile robot based on the Robotnik platform Summit XL HL. See Figure 2.5.

It mounts:

- a Kinect V2 RGB-D device for image and 3D scan

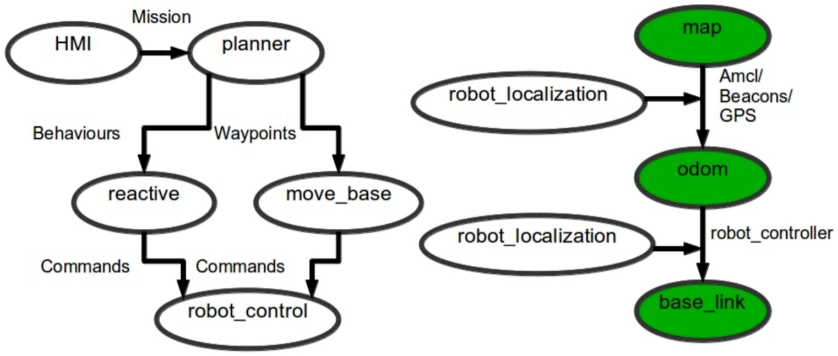


Figura 2.6: Vinbot navigation architecture

- a RTK-DGPS antenna for high accuracy geo-localization
- two lasers Hokuyo 30m outdoor sensors (one for navigation purposes, the second for plant 3d scanning)
- a small computer for basic computational functions running ROS
- a dual interface communication system using Wi-Fi and Radio

Looking at the navigation system it proposes an hybrid reactive/waypoint based navigation architecture. It makes use of a laser range finder and RGBD device to perform reactive row following and obstacle avoidance, while it can make use of other reactive behaviors or GPS waypoint navigation for changing from row to row or field to field, thus supporting different levels of automation. In particular the hybrid architecture implements a high level planner able to receive and store a sequence of waypoints or reactive actions from an external HMI (Human Machine Interface). The reactive actions are preset and are: follow the right/left row, follow the 2 rows (center), turn to the right/left row, turn in-site 180o. The planner is implemented through ROS actionlib stack and provides the waypoints and the reactive actions through two different action servers. The structure is described in 2.5. Then, the VinBot proposal implements the localization of the robot through the ROS package robot_localization, that we will explain the next chapter. Thanks to this package they obtain a localization by merging the

wheel odometry the IMU measurements and the GPS data. They show and discuss their navigation results in this paper [6].

In the chapter 4 we illustrate our navigation system, there it can be observed the different choice respect to the VinBot solution.

3.
Relevant background and tools

In this chapter we describe as clearer as possible the methods that we will exploit in our solution and the theoretical background necessary to understand them. Since this thesis belongs to an applicative field such as the field robotics in which there is an infinite quantity of different scenarios, we don't have a pure state of the art, but we have a set of tools that are currently adopted in the majority of the cases (similar to our one) and that for the specific task that each of them solve, they are one of the most known solutions. Thus, we will present them through their real implementation in order to be specific in preparation to the next chapters. Going in a logic sequence, we analyze as first the techniques that face the odometry estimation with the sensor fusion, describing the robot_localization implementation and the ROAMFREE framework. Then, we illustrate the methods able to solve the SLAM problem referring in particular to Gmapping implementation, but also mentioning different approaches such as KartoSLAM and Google's Cartographer. Finally, we describe the localization approach explaining the most used algorithm and tool, AMCL.

3.1 Background

Since the tools that we report have shown great performance in solving different and difficult applied problems, the background that stays behind them is almost a state of the art for mobile robot navigation. In fact, we explain the concept of state estimation and consequently the various approaches which try to solve it that have emerged in the recent years, such as the Kalman filters, the particle filters and the graph-based filters.

3.1.1 State estimation

In the previous chapter we analyzed the general problems of the outdoor unmanned navigation adapting some of them to our specific case. In particular we showed how the problems relative to multiple sensors are very important in our case and influence the state estimation. Now we go one step further by describing with more details the whole state estimation since it is fundamental in order to make the robot navigate autonomously. We describe the state estimation as the ensemble of two parts: the first regards the odometry estimation while the second depends on the aim of the robot, that in our case can be the mapping of the environment or the localization for the navigation in a known map. These two parts are strictly linked, in particular the second needs the first. Technically speaking the sensor fusion is present in both the parts, but in two different ways. In fact, in the first part all the sensors that give information about the movement of the robot are taken into account, thus wheel encoders, gyroscope, gps etc., instead in the second part the merging is between the estimated odometry and the laser acquisitions. If the estimated odometry is erroneous the second part has a more difficult work and very often, if not always, this means to have worst results. Thus, the odometry estimation is crucial in order to obtain good results, in our case this means accurate maps and a precise localization. Furthermore, the complex is the scenario the higher is the importance of the estimated odometry.

In the previous chapter we outlined the existence of three types of approaches for the state estimation, these are the one representing the basis for the methods that solve the two parts that we just mentioned. Thus, before explaining these methods we illustrate the basic concepts of the three approaches.

3.1.2 Bayesian approaches

The Bayesian approaches are the first important category for the theory of the state estimation in robotics. Obviously, they make usage of the Bayes statistic principles that can be found in chapter 2 of the this book [7]. In general all the approaches belonging to this category start their reasoning from the belief definition. The belief is the posterior of the state and thus is defined by the formula

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{3.1}$$

This posterior is the probability distribution over the state x_t at time t, conditioned on all past measurements $z_{1:t}$ (sensor acquisitions) and all past controls $u_{1:t}$ (robot control inputs). This definition include also the measurement z_t, but it is not always convenient to wait for this measurement before estimate the state, thus there exists another definition of the belief

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{3.2}$$

In the probabilistic filtering this distribution is called prediction, due to the fact that $\overline{bel}(x_t)$ predicts the state at time t based on the previous state posterior before incorporating the measurement at time t. Starting from this definition we can obtain the $bel(x_t)$ by applying a step, known as correction or measurement update.

Bayes filter

The first type of filter we analyze is the most simple, the Bayes filter. It computes the belief in a recursive way, thus the $bel(x_t)$ is computed using $bel(x_{t-1})$ along with the most recent control u_t and the most recent measurement z_t,

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \tag{3.3}$$

where η is a normalization term. This equation can be obtained starting from (3.1) thanks to the Bayes rule and two assumptions: (i)the states follow a first-order Markov process, $p(x_t|x_{0:t-1}) = p(x_t|x_{n-1})$; (ii)the observations are independent of the given states, i.e. $p(z_t|x_{0:t}, z_{1:t}, u_{1:t}) = p(z_t|x_t)$.

The filter can be described more clearly through the pseudo-algorithm reported below. It can be observed that it is composed of two steps, the prediction and the correction. In the prediction $\overline{bel}(x_t)$ is computed integrating the prior belief over state x_{t-1}, and the probability that control u_t induces a transition from x_{t-1} to x_t. Instead in the correction or update step the $\overline{bel}(x_t)$ just computed is multiplied by the probability the measurement z_t may have been observed and since this product can result greater than 1, it is normalized with the parameter η. The initialization of the belief, $bel(x_0)$, is necessary to make the algorithm start and it can be done with certainty if it that value is known or it can be generated with a uniform distribution (or a different one) if it is unknown (or partially known).

The problem of this algorithm is the complexity, in fact the presence of an integral is a bottleneck and further the product in the update state can

Algorithm 1 Bayesian filtering

function BAYESFILTER($bel(x_{t-1}), u_t, z_t$)
  for all x **do**
    $\overline{bel}(x_t) \leftarrow \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx$
    $bel(x_t) \leftarrow \eta p(z_t|x_t)\overline{bel}(x_t)$
  end for
  return $bel(x_t)$
end function

Algorithm 2 Kalman filtering

function KALMANFILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
  $\overline{\mu}_t \leftarrow A_t\mu_t + B_tu_t$
  $\overline{\Sigma}_t \leftarrow A_t\Sigma_{t-1}A_t^T + R_t$
  $K_t \leftarrow \overline{\Sigma}_tC_t^T\left(C_t\overline{\Sigma}_tC_t^T + Q_t\right)^{-1}$
  $\mu_t \leftarrow \overline{\mu}_t + K_t\left(z_t - C_t\overline{\mu}_t\right)$
  $\Sigma_t \leftarrow \left(I - K_tH_t\right)\overline{\Sigma}_t$
  return μ_t, Σ_t
end function

result very complex. Thus, a classical Bayes filter can be used only for very simple problems in which the state space is very limited, since in these cases the integral can be reduced to a finite sum.

Kalman filter

The Kalman filter (KF) is the first practical implementation of the Bayes filter. It was invented by Rudolph E. Kalman in the 1960. It try to transform the bayesian filter formulation in a efficient one, deleting the integral and putting it in a closed form. To do this, It assumes the beliefs are Gaussian: At time t, the belief is represented by the the mean μ_t and the covariance Σ_t. In addition to the Markov assumptions there are other three properties that are needed to treat the beliefs as Gaussian: (i) the states are linear, $x_t = A_tx_{t-1} + B_tu_t + \epsilon_t$ where A_t and B_t are matrices having dimension according to the ones of the state vector x_t and control vector u_t, while ϵ is a Gaussian noise; (ii) the measurement probability is linear in its arguments, $z_t = C_tx_t + \delta_t$, where C_t is matrix and δ a Gaussian noise; (iii) $bel(x_0)$ is normally distributed.

Algorithm 3 Extended Kalman filtering

function EXTENDEDKALMANFILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
$\overline{\mu}_t \leftarrow g(u_t, \mu_{t-1})$
$\overline{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_t^T + R_t$
$K_t \leftarrow \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1}$
$\mu_t \leftarrow \overline{\mu}_t + K_t \left(z_t - h(\overline{\mu}_t)\right.$
$\Sigma_t \leftarrow (I - K_t C_t) \overline{\Sigma}_t$
return μ_t, Σ_t
end function

Given all the assumptions the Kalman filter can be formulated in a closed form that we report with a pseudo algorithm. The demonstration of the listed formulae can be found at [7]. Again we have a two steps filter. The differences respect to the purely Bayes filter are that the beliefs are substituted by the mean, μ_t, and the covariance Σ_t that are predicted in the first two lines by including the new control u_t and updated in the remaining ones by considering the new measurement z_t. Another news is represented by the variable K_t that is known as Kalman gain and specifies the degree to which the measurement is incorporated into the new state estimate.

Thanks to its matrix formulation the Kalman filter can be solved in a closed form and thus results efficient. Instead, it suffers for what concern the generality, in the sense that very often the assumptions (i) and (ii) are too strong, thus far from the reality and so it cannot be applied in many real cases. For this reason it has been evolved to an Extended version, so EKF.

Extended Kalman Filter

The EKF solve the problems of the KF by replacing the linear predictions with their nonlinear generalizations. Moreover, EKFs use Jacobians G_t and H_t instead of the corresponding linear system matrices A_t, B_t, and C_t in Kalman filters. The Jacobian G_t corresponds to the matrices A_t and B_t, and the Jacobian H_t corresponds to C_t. Thus it maintain the computational efficiency and the simplicity of the KF while it adds the applicability to many cases, it results robust also in for unexpected problems (problems that violate the underlying assumptions). We report here the pseudo algorithm, just for simplify the comparison with the standard KLF, but we then explain all the passages in section 3.3.1 where it has a real implementation.

The EKF main limit is that it approximates state transitions and measurements using linear Taylor expansions [8] . In virtually all robotics problems, these functions are nonlinear. The goodness of this approximation depends on two main factors. First, it depends on the degree of nonlinearity of the functions that are being approximated. If these functions are approximately linear, the EKF approximation may generally be a good one, and EKFs may approximate the posterior belief with sufficient accuracy. However, sometimes, the functions are not only nonlinear, but are also multi-modal, in which case the linearization may be a poor approximation. The goodness of the linearization also depends on the degree of uncertainty. The less certain the robot, the wider its Gaussian belief, and the more it is affected by nonlinearities in the state transition and measurement functions. In practice, when applying EKFs it is therefore important to keep the uncertainty of the state estimate small. We also note that Taylor series expansion is only one way to linearize. In fact, there exist other two approaches that have demonstrated to lead to better results. One is the Unscented Kalman filter (UKF), which probes the function to be linearized at selected points and calculates a linearized approximation based on the outcomes of these probes. Another is known as moments matching, in which the linearization is calculated in a way that preserves the true mean and the true covariance of the posterior distribution (which is not the case for EKFs).

Particle filter

An alternative to the Kalman filters (in general to Gaussian filters) are nonparametric filters. These filters do not rely on a fixed functional form of the posterior, such as Gaussians. Instead, they approximate posteriors by a finite number of values that are samples of the real distribution (state space). The number of parameters used to approximate the posterior can be varied. The quality of the approximation depends on the number of parameters used to represent the posterior. As the number of parameters goes to infinity, nonparametric techniques tend to converge uniformly to the correct posterior (under specific smoothness assumptions). Some nonparametric Bayes filters rely on a decomposition of the state space, in which each such value corresponds to the cumulative probability of the posterior density in a compact subregion of the state space. Others approximate the state space by random samples drawn from the posterior distribution. In particular, we explain a

method belonging to this latter category, called particle filter.

The particle filter is a nonparametric implementation of the Bayes filter, thus it start from the same belief definition, but it approximate the posterior distribution by using a finite number of parameters, the particles. The key idea of the particle filter is to represent the posterior $bel(x_t)$ by a set of random state samples drawn from this posterior. 3.1 illustrates this idea. Obviously, this representation is an approximation of the real distribution, but being drawn from the posterior it is able to represent a much broader space of distributions.

The particles are denoted as

$$\mathcal{X}_t = x_t^{(1)}, x_t^{(2)}, ..., x_t^{(M)}$$

where M is the finite number of particles (usually large). Each particle $x_t^{(m)}$ (with $1 \leq m \leq M$) is a concrete instantiation of the state at time t, thus it is a hypothesis about what the true world state may be at time t. In some implementations M is a function of t or of other quantities related to the belief. Thus the belief $bel(x_t)$ is approximated by the set of particles $\mathcal{X}_t$. Ideally, the likelihood for a state hypothesis x_t to be included in the particle set $\mathcal{X}_t$ shall be proportional to its Bayes filter posterior $bel(x_t)$:

$$x_t^{(m)} \sim p(x_t | z_{1:t}, u_{1:t}) \tag{3.4}$$

As a consequence of (4.23), the denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region. This property should hold only for $M \to \infty$, but in practice this can approximated drawing the particles from a slightly different distribution and using a not too small M (i.e. $M \geq 100$).

Just like all other Bayes filter algorithms, the particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that particle filters construct the particle set $\mathcal{X}_t$ recursively from the set $\mathcal{X}_{t-1}$. The most basic variant of the particle filter algorithm is shown in the Algorithm 4. The input of the algorithm are the particles $\mathcal{X}_{t-1}$, the most recent control u_t and measurement z_t. In the first cycle the algorithm constructs the temporary set $\overline{\mathcal{X}}$ which is reminiscent (but not equivalent) to $\overline{bel}_t$. In this cycle the first step generates a hypothetical state $x_t^{(}m)$ for time t based on the particle $x_{t-1}^{(}m)$ and the control u_t. The m apex of the sample indicates that it is generated from the m-th particle in $\mathcal{X}_{t-1}$. The

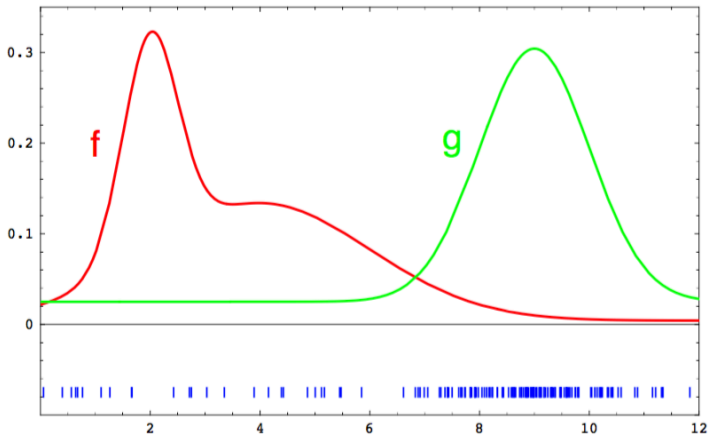
Algorithm 4 Particle filtering

function PARTICLEFILTER($\mathcal{X}_{t-1}, u_t, z_t$)
  $\overline{\mathcal{X}}_t \leftarrow \mathcal{X}_t \leftarrow \emptyset$
  for all $m := 1$ to M **do**
    sample $x_t^{(m)} \sim p(x_t|u_t, x_{t-1}^{(m)})$
    $w_t^{(m)} \leftarrow p(z_t|x_t^{(m)})$
    $\overline{\mathcal{X}}_t \leftarrow \overline{\mathcal{X}}_t + \langle x_t^{(m)}, w_t^{(m)} \rangle$
  end for
  for all $m := 1$ to M **do**
    draw i with probability $\propto w_t^{(i)}$
    add $x_t^{(i)}$ to $\mathcal{X}_t$
  end for
  return $\mathcal{X}_t$
end function

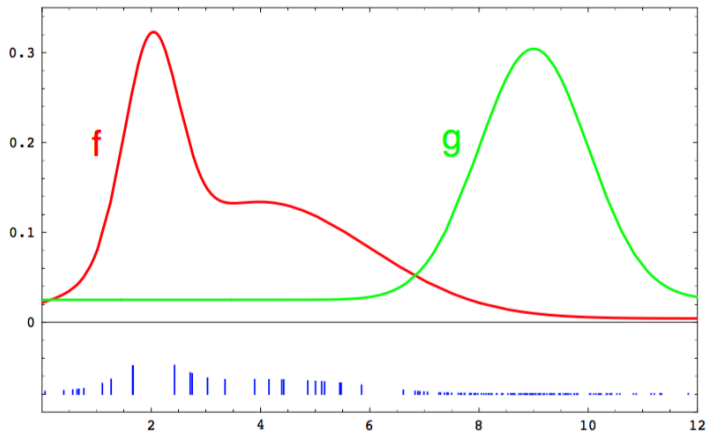
sampling is done from the distribution $p(x_t|u_t, x_{t-1}^{(m)})$. Then the second step calculates for each particle $x_t^{(m)}$ its importance factor (weight) $w_t^{(m)}$, which is the probability of z_t under the particle $x_t^{(m)}$. Finally the third step adds iteratively the new samples with the correspondent weights to the set $\overline{\mathcal{X}}$.

In the second cycle the algorithm implements the resampling or importance resampling. Thus it draws with replacement M particles from the temporary set $\overline{\mathcal{X}}_t$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the importance weights into the resampling process, the distribution of the particles change: whereas before the resampling step, they were distributed according to $\overline{bel}(x_t)$, after the resampling they are distributed (approximately) according to the posterior $bel(x_t) = \eta p(z_t|x_t^{(m)})\overline{bel}(x_t)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement and the particles that are not contained in it tend to be the ones with lower importance weights.

The importance resampling is the main feature of the particle filters, in fact it allows to draw particles from a known distribution, called proposal distribution, and iteratively update them according to their relative weights until the set of sample approximate enough correctly the target distribution, $bel(x_t)$. The weight of a particle represent the similarity between the proposal



(a)



(b)

Figura 3.1: Illustration of importance resampling in particle filters: (a) Instead of sampling from f directly, we can only generate samples from a different density, g. Samples drawn from g are shown at the bottom of this diagram. (b) A sample of f is obtained by attaching the weight $f(x)/g(x)$ to each sample x.

and the target function computed in it. Thus

$$w_t^{(m)} = \frac{f(x_t^{(m)})}{g(x_t^{(m)})} \tag{3.5}$$

where f is the target distribution and g is the proposal one. The resampling step is easily synthesized in the Figure (3.1).

The approximation errors in particle filters are unavoidable since the set of sample is a discrete distribution while the target distribution is a continuos one. In particular, there are four complimentary sources of approximation error, each of which gives rise to improved versions of the particle filter.

1. The first approximation error relates to the fact that only a finite number of particles are used. This introduces a systematic bias in the posterior estimate. Thus, it is fundamental to use an high number of particles M in order to limit the degree of approximation.

2. A second source of error in the particle filter relates to the randomness introduced in the resampling phase. In particular, the resampling process induces a loss of diversity in the particle population, which in fact manifests itself as approximation error. Such error is called variance of the estimator: Even though the variance of the particle set itself decreases, the variance of the particle set as an estimator of the true belief increases. Controlling this variance, or error, of the particle filter is essential for any practical implementation.

3. A third source of error pertains to the divergence of the proposal and target distribution. In fact, the more they are different the more the algorithm must iterate to approximate the target. Thus, the efficiency of the particle filter relies crucially on the match between the proposal and the target distribution. If, at one extreme, the sensors of the robot are highly inaccurate but its motion is very accurate, the target distribution will be similar to the proposal distribution and the particle filter will be efficient. If, on the other hand, the sensors are highly accurate but the motion is not, these distributions can deviate substantially and the resulting particle filter can become arbitrarily inefficient.

4. A fourth and final disadvantage of the particle filter is known as the particle deprivation problem. When performing estimation in a high-dimensional space, there may be no particles in the vicinity to the correct state. This might happen because the number of particles is too small to cover all relevant regions with high likelihood, but it is not the only reason. In fact, also for M enough high during the resampling there is a probability greater than zero (due to the random nature) that all the particles near the true state are wiped out.

In conclusion the particle filters performance strongly depends on the number of particles used, that obviously is very difficult to be pre-estimated. Thus, it is usually tuned during the experiments. In particular, the complex is the state space the higher should be the number of particles.

3.1.3 Graph-based approaches

The development of graph-based approach in robotics starts in 1997 when Lu and Milios [9] propose a graph-based solution to the SLAM problem. In fact, in the recent years graph-based has been heavily applied to the SLAM problem since it allows to formulate the problem in a very intuitive way. As we will see better in the SLAM section, 3.4, the basic formulation using a graph represent the robot poses as nodes while the landmark parametrization builds the edge. Thus, if a landmark is visible from a certain pose, then an edge is added between the two poses. The state estimation problem is then formulated as a max-likelihood optimization over the built graph in which the goal is to find the configuration of robot poses and landmarks such that the joint likelihood of all the observations is maximum. The graph structure allows to exploits many statistical tricks, especially regarding the joint probabilities since the nodes connections impose the dependency and independency of the variables. Common statistical graph based approaches are the Bayesian networks (belief networks), the Markov networks and many others [10]. The graph state estimation to be solved requires a large non-linear, least-squares, optimization problem. For this reason the greatness of the optimization method is very important for the efficiency and the application of a graph-based algorithm. In fact, Graph-based approaches are nowadays considered superior to conventional Bayesian solutions [11] , but they needed some major advancements in sparse linear algebra [12] to become competitive from the point of view of computational complexity. For a modern synthesis of these optimization methods see [13].

The advent of graph techniques in SLAM slightly changed the perspective from the state estimation point of view: while filters typically model state estimation as a recursive process performed measurement-per-measurement and the state consists of the latest robot pose and all the landmarks, graph-based approaches attempt to estimate the full robot trajectory, and thus a (long) sequence of robot poses together with the landmark positions form the whole set of measurements. This notion was already present in other

approaches [14] which never found a robotics application due to the counter intuitive structure. From the SLAM formulation the graph approach has been generalized even further, considering hyper-graphs, called factor graphs [15] that allows a more flexible graph building. We illustrate them in section 3.3.2. Thanks to this very generic graph many powerful tool for multi-sensor fusion has been developed [16]. In particular we will explain in details a tool called ROAMFREE that will solve for us the sensor fusion problem.

3.2 Odometry estimation

We just wrote about the importance of odometry estimation in complex scenarios, thus also in our one. The odometryăis the use of data fromămotion sensorsăto estimate change in position over time. It follows that the goal of the odometry estimation is to estimate as well as possible the pose (position and orientation) of the robot given the sensor acquisitions and the initial pose of the robot in a certain time interval.

3.2.1 Odometry

The simplest method to compute the odometry is only numeric. The standard odometry estimates the distance travelled by measuring how much the wheels have turned, thus it integrates the velocity measurements over the time. The computation differs according to the number and the shape of wheels mounted on the robot. The simplest case is a single freely rotating wheel that, if considered with ideal conditions, implies to cover a distance on the ground of $2\pi r$ for each rotation, where r is the radius of the wheel. In practice, even the behavior of a single wheel is substantially more complicated than this. In fact, there are many variable factors that can create error, such as the wheels that are not necessarily mounted so as to be perpendicular, nor are they aligned with the normal straight-ahead direction of the vehicle. In addition to these issues related to precise wheel orientation and lateral slip, there can also be insufficient traction that leads to slipping or sliding in the direction of the wheels motion, which makes the estimate of the distance travelled imprecise. Then other factors arise due to compaction of the terrain and cohesion between the surface and the wheel (as we stated in Chapter 2).

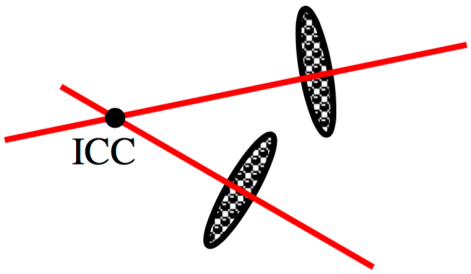


Figura 3.2: ICC identification

Each wheeled mobile robot (WMR) to be able to move must have a point around which all wheels follow a circular course. This point is known as the instantaneous center of curvature (ICC) or the instantaneous center of rotation (ICR). In practice it is quite simple to identify the ICC because it must lie on a line coincident with the roll axis of each wheel that is in contact with the ground, see Figure 3.2. Thus, when a robot turns the orientation of the wheels must be consistent and a ICC must be present otherwise the robot cannot move.

A WMR can only moves on a plane and thus it has three degrees of freedom represented by the three component of the pose (x, y, θ) where (x, y) is the position and θ is the heading or orientation. The ability to have complete independent control over all these three parameters depends on the disposition and the number of wheels and usually mobile robots dont have a complete control. Thus, they are obliged to perform complex maneuvers in order to reach a desired pose (i.e. car parking). Referring to our case, we now describe the type of kinematic of our robot, in order to understand what are the motion that it can or it cannot do. Our robot moves with a skid-steering kinematic that is a derivative of the differential drive kinematic.

3.2.2 Differential drive odometry

Differential drive is probably the simplest possible drive mechanism for a ground contact mobile robot. Often used on small, low-cost, indoor robots such as the TurtleBot [17] or Khepera [18], larger commercial bases such as the Robuter [19] utilize this technology as well. As depicted in Figure 3.3 , a differential drive robot consists of two wheels mounted on a common axis, thus parallel, controlled by separate motors.

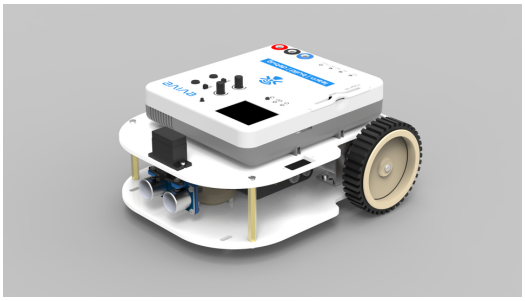


Figura 3.3: Example of differential drive robot

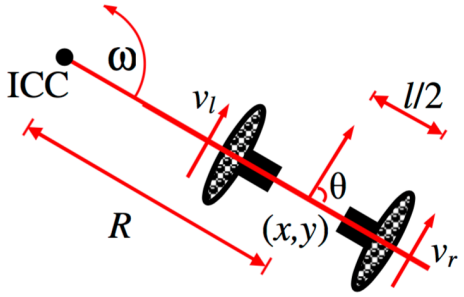


Figura 3.4: Differential drive robot

Consider how the controlling wheel velocities determine the vehicles motion. For each of the two drive wheels to exhibit rolling motion, the robot must rotate around a point that lies on the common axis of the two drive wheels. By varying the relative velocity of the two wheels, the point of this rotation can be varied and different vehicle trajectories chosen.

At each instant in time, the point at which the robot rotates must have the property that the left and right wheels follow a path that moves around the ICC at the same angular rate ω, and thus

$$\omega(R + l/2) = v_r$$
$$\omega(R - l/2) = v_l,$$

where l is the distance along the axle between the centers of the two wheels, the left wheel moves with velocity v_l along the ground and the right with velocity v_r , and R is the signed distance from the ICC to the midpoint between the two wheels. Note that v_l , v_r , ω, and R are all functions of

time. At any instant in time, solving for R and ω results in

$$R = \frac{l}{2}\frac{(v_l + v_r)}{(v_r - v_l)}, \qquad \omega = \frac{v_r - v_l}{l}.$$

A number of special cases are of interest. If $v_l = v_r$, then the radius R is infinite, and the robot moves in a straight line. If $v_l = -v_r$, then the radius is zero, and the robot rotates about a point midway between the two wheels, that is, it rotates in place. This makes differential drive attractive for robots that must navigate in narrow environments.

For other values of v_l and v_r , the robot does not move in a straight line but rather follows a curved trajectory about a point a distance R away from the center of the robot, changing both the robots position and orientation.

The kinematic structure of the vehicle prohibits certain vehicle motions. For example, there is no combination of v_l and v_r such that the vehicle can move directly along the wheels common axis.

A differential drive vehicle is very sensitive to the relative velocity of the two wheels. Small errors in the velocity provided to each wheel result in different trajectories, not just a slower or faster robot. Differential drive vehicles typically use castor wheels for balance. Thus, differential drive vehicles are sensitive to slight variations in the ground plane. This limits their applicability in non-laboratory environments.

To compute a robots position $\mathbf{x}$ in the idealized error-free case with a velocity vector dx/dt, we use

$$\mathbf{x} = \int_{t_0}^{t_f} \frac{dx}{dt} dt,$$

where the motion takes place over a time interval t_0 through t_f . More generally, for motion information from higher-order derivatives (such as acceleration) we can integrate repeatedly to recover position. This also implies, however, that errors in the sensing or integration process are manifested as higher-order polynomials of the time interval over which we are integrating. For discrete motions, where positional change is expressed by a difference vector δ_i , we can compute the absolute position as

$$x = \sum \delta_i.$$

Let us explain with some details the odometry computation (Forward kinematics) for differential drive robots.

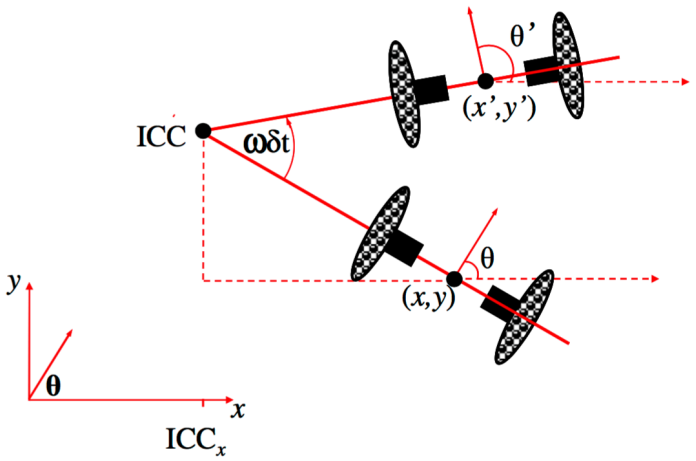


Figura 3.5: Differential drive robot motion from pose (x, y, θ) *to* (x', y', θ')

Suppose that the robot is at some position (x, y) and facing along a line making an angle θ with the x axis 3.4. Through manipulation of the control parameters v_l and v_r , the robot can be moved and take on different poses. Since v_l and v_r and hence R and ω are functions of time, it is straightforward to show that if the robot has pose (x, y, θ) at some time t, and if the left and right wheels have contact velocities v_l and v_r, respectively, during the period $t \to t + \delta t$, then the ICC is given by

$$\text{ICC} = (x - Rsin(\theta), y + Rcos(\theta)),$$

and at time $t + \delta t$ the pose of the robot is given by

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}. \tag{3.6}$$

Equation (3.6) describes the motion of a robot rotating a distance R about its ICC with an angular velocity given by ω. See Figure 3.5.

By integrating (3.6) from some initial condition (x_0, y_0, θ_0), it is possible to compute where the robot will be at any time t based on the control parameters $v_l(t)$ and $v_r(t)$, that is, to solve the forward kinematics problem for the vehicle. In general, for a robot capable of moving in a particular

direction $\theta(t)$ at a given velocity $V(t)$,

$$\begin{aligned} x(t) &= \int_0^t V(t)cos(\theta(t))dt \\ y(t) &= \int_0^t V(t)sin(\theta(t))dt \\ \theta(t) &= \int_0^t \omega(t)dt, \end{aligned} \tag{3.7}$$

and for the special case of a differential drive vehicle,

$$\begin{aligned} x(t) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))cos(\theta(t))dt \\ y(t) &= \frac{1}{2}\int_0^t (v_r(t) + v_l(t))sin(\theta(t))dt \\ \theta(t) &= \frac{1}{l}\int_0^t (v_r(t) - v_l(t))(t)dt, \end{aligned} \tag{3.8}$$

3.2.3 Skid-Steering odometry

The skid-steering kinematic (see Figure 3.6) is an evolution of the differential drive, in the sense that it tries to maintain its simplicity while it is physically a more robust model. With this motion model the robot requires slippage of the wheels while it turns. Like differential drive, skid-steering leads to high maneuverability [20][21] , and has a simple and robust mechanical structure, leaving more room in the vehicle for the mission equipment [22][23]. In addition, it has good mobility on a variety of terrains, which makes it suitable for all-terrain missions. However, this locomotion scheme makes it difficult to develop kinematic and dynamic models that can accurately describe the motion. It is very difficult for the skid-steering kinematics to predict the exact motion of the vehicle only from its control inputs. As a result, the kinematics models with pure rolling and no-slip assumptions for non-holonomic wheeled vehicles cannot apply in this case. Furthermore, other disadvantages are that the motion tends to be energy inefficient, difficult to control, and the tires tend to wear out faster [24]. Thanks to the work of Wang et al. [25] we can give a coarse explanation of which are the mathematical differences with to the differential drive model and how the skid-steering can be approximated to it.

First of all we make three assumptions: (i) the mass center of the robot is located at the geometric center of the body frame; (ii) the two wheels of

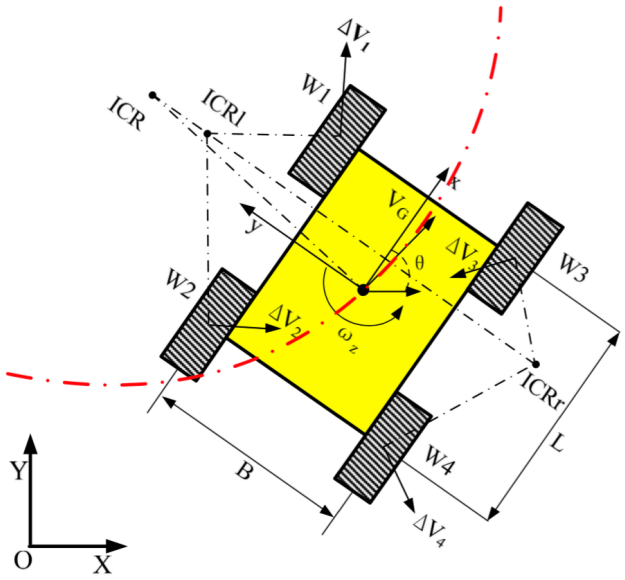


Figura 3.6: Skid-Steering robot

each side rotate at the same speed ($w_l = w_1 = w_2$ and $w_r = w_3 = w_4$); (iii) the robot is running on a firm ground surface, and four wheels are always in contact with the ground surface.

Then consider the Figure 3.6 and the Equation (3.8). By deriving this latter we obtain

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = f \begin{bmatrix} w_l r \\ w_r r \end{bmatrix}, \tag{3.9}$$

where $v = (v_x, v_y)$ is the vehicle's translational velocity with respect to its local frame, w_z is its angular velocity and r is the radius of the wheels.

When the robot moves we observe the presence of more ICRs: ICR_l, ICR_r, ICR_G that are respectively of the left-side tread, right-side tread, and the robot body. We define the coordinates of the ICRs respect to the local frame as (x_l, y_l), (x_r, y_r) and (x_G, y_G). All the treads share the same angular velocity w_z. Thus,

$$y_G = \frac{v_x}{w_z} \tag{3.10}$$

$$y_l = \frac{v_x - w_l r}{w_z} \tag{3.11}$$

$$y_r = \frac{v_x - w_r r}{w_z} \tag{3.12}$$

$$x_G = x_l = x_r = -\frac{v_y}{w_z} \tag{3.13}$$

From Equations (3.8) to (3.11) the kinematics relation (3.7) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} = J_w \begin{bmatrix} w_l r \\ w_r r \end{bmatrix}, \tag{3.14}$$

Where the elements of matrix J_w depend on the tread ICR coordinates:

$$J_w = \frac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix} \tag{3.15}$$

If the mobile robot is symmetrical, we can get a symmetrical kinematics model (i.e., the ICRs lie symmetrically on the x-axis and $x_G = 0$), so matrix J_w can be written as the following form:

$$J_w = \frac{1}{2y_0} \begin{bmatrix} y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \tag{3.16}$$

where $y_0 = y_l = -y_r$ is the instantaneous tread ICR value. Noted that $v_l = w_l r$, $v_l = w_r r$ for the symmetrical model, the following equations can be obtained:

$$\begin{cases} v_x = \dfrac{v_l + v_r}{2} \\ v_y = 0 \\ w_z = \dfrac{-v_l + v_r}{2y_0} \end{cases} \tag{3.17}$$

Noted $v_y = 0$, so that $v_G = v_x$. We can get the instantaneous radius of the path curvature:

$$R = \frac{v_G}{w_z} = \frac{v_l + v_r}{-v_l + v_r} y_0 \tag{3.18}$$

A non-dimensional path curvature variable λ is introduced as the ratio of sum and difference of left- and right-sides wheel linear velocities [26] , namely:

$$\lambda = \frac{v_l + v_r}{-v_l + v_r} \tag{3.19}$$

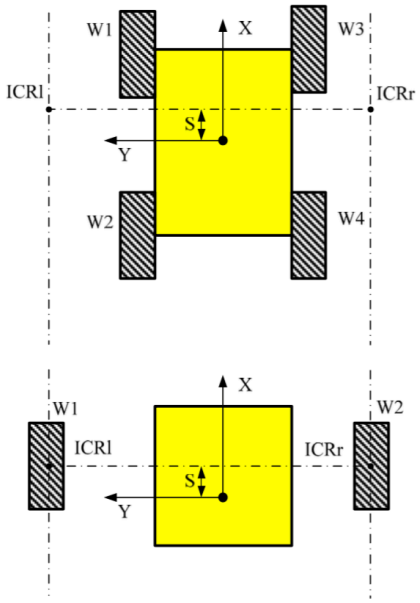


Figura 3.7: Skid-Steering and differential drive equivalence

and we can rewrite Equation (3.16) as:

$$R = \lambda y_0 \tag{3.20}$$

We use a similar index as in Mandows work [27], then an ICR coefficient χ can be defined as:

$$\chi = \frac{y_l - y_r}{B} = \frac{2y_0}{B}, \qquad \chi \geq 1 \tag{3.21}$$

where denotes the lateral wheel bases, as illustrated in Figure 3.6.

χ indicates the approximation from the differential drive ideal kinematic. In fact, χ is equal to 1 when there is no slippage and thus when the model corresponds to the ideal differential drive. Therefore, for instantaneous motion, there is kinematic equivalences between skid-steering and ideal wheel vehicles. This means that the skid-steering can be approximated to the differential drive and that the approximation can be done increasing the wheel separation in the differential drive formulae as shown in Figure 3.7. This approximation is very important, and we will exploit it in order to have a simple but correct motion model in our final solution.

3.3 Sensor fusion frameworks

In this section we illustrate two real tools that solve the sensor fusion problem producing as output an filtered odometry.

3.3.1 Robot_Localization

The standard odometry estimation as already written, presents many problem and its simplicity it is not enough especially for real complex scenario. The first alternative that we report can be found in the robot_localization package [28]. This package is one of the most adopted tool for the sensor fusion odometry estimation for its excellent tradeoff among performance, complexity and reliability. it is an efficient solution that is built for the ROS framework with a generic structure that can be configured for many different cases. For these reasons it is widely adopted in the robotic community. It performs state estimation in 3D space, allows for an unlimited number of sensors, supports multiple standard ROS message types, and allows per-sensor control of which message fields are fused with the state estimate. Another important feature is the continuos estimation, thus each state estimation node begins estimating the vehicle state as soon as it receives a single measurement, if there is a long period in which no data is received the filter continues to estimate the robot state via an internal motion model. The type of sensors that it supports are the odometry sources (encoders), the IMUs (gyroscope, accelerometer, heading) and GPS sensors. Currently the package is composed of more ROS nodes (the ones we tried are better described in Chapter 4): two state estimation nodes, one using exploiting UKF (Unscented Kalman Filter) and the other EKF, plus another node for the GPS conversion. The core node is the one implementing the Extended Kalman Filter (EKF) and so we describe its logic.

Extended Kalman Filter oppure EKF node

The EKF formulation and algorithm is very known and can be found in the theoretical background of many fields different from robotics. Here some articles explaining it [29][30][31]. The final goal of the EKF is to estimate the odometry. Thus, the process can be described as a nonlinear dynamic system, with

$$x_t = g(u_t, x_{t-1}) + \epsilon_t, \tag{3.22}$$

where x_t is the robots system state (i.e., 3D pose) at time t, g is a nonlinear state transition function, and ϵ_t is the process noise, which is assumed to be normally distributed. The 15-dimensional state vector, x, comprises the vehicles 3D pose, 3D orientation, and their respective velocities and accelerations. Rotational values are expressed as Euler angles.

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

Additionally, the sensors measurements have the form

$$z_t = h(x_t) + \delta_t, \tag{3.23}$$

where z_t is the measurement at time t, h is a nonlinear sensor model that maps the state into measurement space, and δ_t is the normally distributed measurement noise.

The first stage in the algorithm, shown as equations (3.24) and (3.25), is to carry out a prediction step that projects the current state estimate and error covariance forward in time:

$$\overline{\mu}_t = g(u_t, \mu_{t-1}). \tag{3.24}$$

$$\overline{\Sigma}_t = G\Sigma_{t-1}G^T + R. \tag{3.25}$$

In this interpretation, g is a standard 3D kinematic model derived from Newtonian mechanics. The estimate error covariance, Σ, is projected via G, the Jacobian of g, and then perturbed by Q, the process noise covariance.

Then the filter correction step is the following:

$$K = \overline{\Sigma}_t H^T \left(H\overline{\Sigma}_t H^T + Q\right)^{-1}. \tag{3.26}$$

$$\mu_t = \overline{\mu}_t + K\left(z_t - H\overline{\mu}_t\right). \tag{3.27}$$

$$\Sigma_t = \left(I - KH\right)\overline{\Sigma}_t\left(I - KH\right)^T + KQK^T. \tag{3.28}$$

The Kalman gain is calculated using the observation matrix, H, the measurement covariance, R, and $\overline{\Sigma}_t$. Then the gain is used to update the state vector and covariance matrix. To promote filter stability by ensuring that Σ_t remains positive semi-definite the Joseph form covariance update equation (3.27) is exploited.

The standard EKF formulation specifies that H should be a Jacobian matrix of the observation model function h. To support a broad array of

sensors, Moore makes the assumption that each 15sensor produces measurements of the state variables we are estimating. As such, H is simply the identity matrix. A core feature of EKF node is that it allows for partial updates of the state vector. This is critical for taking in sensor data that does not measure every variable in the state vector, which is nearly always the case. In practice, this can be accomplished through H. Specifically, when measuring only m variables, H becomes an m by 15 matrix of rank m, with its only nonzero values existing in the columns of the measured variables.

3.3.2 ROAMFREE

The name ROAMFREE stays for Robust Odometry Applying Multi-sensor Fusion to Reduce Estimation Errors. Thus, as can be deducted, it is a tool able to solve the sensor fusion problem and the odometry estimation, but it could be used also for more extended scopes.

Framework Overview

It is a flexible and modular framework designed to deliver to mobile robots and unmanned vehicles developers (i) off-the-shelf position and attitude tracking, (ii) intrinsic, extrinsic, and kinematic parameters self-calibration capabilities. In ROAMFREE the information fusion problem is formulated as a fixed-lag smoother whose goal is to track not only the most recent pose, but all the positions and attitudes of the mobile robot in a fixed time window: short lags allow for real time pose tracking, still enhancing robustness with respect to measurement outliers; longer lags allow for online calibration, where the goal is to refine the available estimate of sensor parameters.

The core of ROAMFREE lies in a software module that keeps and updates the probabilistic representation of the sensor fusion problem in terms of a factor graph, composed of pose, sensor parameter nodes and sensor error models connecting them. Other modules, such as the outlier rejection module, which detects and exclude incoherent sensor readings from the estimation, and inference algorithms performing state estimation, operate upon this representation. The factor graph allows to deal with an arbitrary number of multi-rate sensors, i.e., different sensors producing readings at different rates, having non-constant frequencies of operation, and possibly producing out-of-sequence data. The framework is completely independent from the hardware of the robot, in fact it abstracts the real sensors with some general

logic sensors which can be configured in order to correctly represent the real ones. Each logic sensor is characterized by a parametric error model specific for its domain. ROAMFREE is very modular and can be used with ROS. Unfortunately, currently the public implementation of RF as a ROS node is yet primitive and very unstable. For this reason in our solution we have implemented the ROS node from scratch.

Factor graph filter

As anticipated the core of RF stays in the factor graph filter. We have written about the basic concepts of graph-based approaches in 3.1.3 and we show these approaches are really good for high-dimensional problem, such sensor fusion. The graph represent the full joint probability of sensor readings given the current estimate of the state variables, representing its factorization in terms of single measurement likelihoods. The nodes in the factor graph contain all the robot poses in a given time window, and the sensor calibration parameters, such as gains, biases, displacements or misalignments. The factors represent measurement constraints: each factor is an hyper-edge and connects multiple pose and calibration parameters nodes. Whenever a new sensor reading is available a new factor instantiated according to the type of the information source, and inserted into the graph and it is connected to the pose and sensor parameter nodes required to evaluate the measurement likelihood. The number of nodes to which the factor is linked depends on its domain. Since the sensor fusion problem is formulated as a non-linear optimization, an initial guess is needed for the new state variable. If the sensor reading provides enough information, a prediction for the new robot pose can be obtained based on the latest available estimate and on the sensor reading itself. Otherwise, the measurement handling is delayed until such information becomes available. Moreover, as new pose nodes are inserted into the graph, old ones have to be removed so that the length of the fixed-lag window remains constant. This causes old factors to be removed as well. To avoid the loss of information, old nodes are marginalized and an new factor is inserted over their Markov blanket [32] in a way such that it is equivalent to the lost edges, in the neighborhood of the current node estimates. The filter usually estimates the pose by means of max-likelihood estimation over the joint distribution, which can be efficiently done by means of non-linear optimization algorithms. An example of a factor graph is shown in Figure

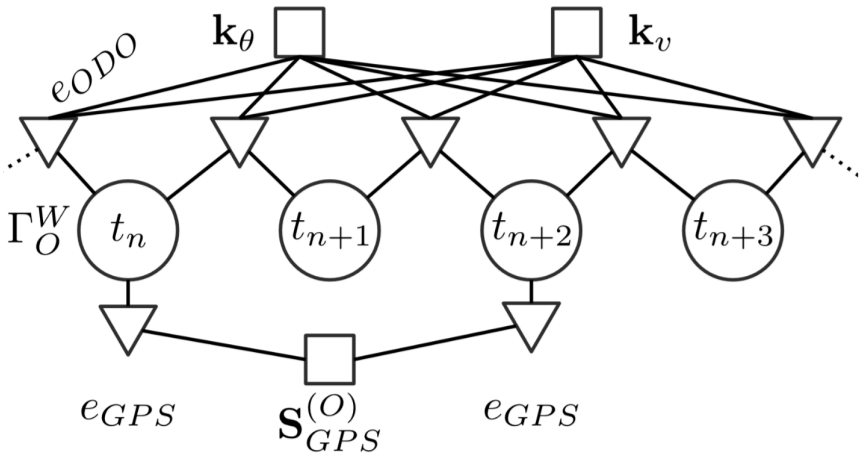


Figura 3.8: An instance of the pose tracking factor graph with four pose vertices $\Gamma_O^W(t)$ (circles), odometry edges e_{ODO} (triangles), two shared calibration parameters vertices k_θ and k_v (squares), two GPS edges e_{GPS} and the GPS displacement parameter $\mathbf{S}_{GPS}^{(O)}$.

3.8.

The advantages of the factor-graph formulation are manifold. First of all, it is general and flexible with respect to the nature, and multiplicity, of information sources; indeed, if we need to add a new sensor to the architecture, this reduces to insert further edges into the pose-graph, once a proper likelihood function has been defined for such measurement domain. Second, it allows to apply non-linear optimization algorithms that are aware of the manifold state variables belong to. Moreover, it allows to solve both the offline parameter calibration and the real-time pose tracking problems; the latter case indeed simply restricts the max-likelihood estimation, i.e., the non-linear optimization, to a subset of the robot poses. Finally, out-of-sequence and delayed measurements do not constitute an issue, as it might be the case in Bayesian approaches; they are simply associated to the proper pose nodes, according to their timestamps.

Error models

As stated before RF is based on some generic logical sensors, each of which is represented by its specific error model. All the error models starts from a

common definition,

$$e_i(t) = \hat{z}(t; \hat{x}_{S_i}(t), \xi) - z + \eta \tag{3.29}$$

where $\hat{z}(x)$ is a measurement predictor computed as a function of the incident nodes, in particular $\hat{x}_{S_i}(t)$ is the extended state for the sensor frame S_i in which are indicated the motion (velocity and acceleration), the position and the orientation of the sensor respect to the world frame and ξ is the vector of parameters relative to the sensor. Then, z is the actual sensor reading and it is directly associated to the corresponding factor, finally η is a zero-mean, Gaussian noise encoding measurement uncertainty. Equation (3.29) yields the difference between the expected sensor reading given the robot state and calibration parameters at time t, and the actual measurement produced by the sensor. It can be observed that zero-mean error $e(t)$ is obtained when the prediction of the sensor reading matches the actual one.

The specific form of the measurement predictor $\hat{z}(x)$ depends on the type of measurement we are considering. The available measurement domains are: (i) absolute position and/or orientation, (ii) linear and angular velocity in sensor frame, (iii) acceleration in sensor frame, (iv) vector field in sensor frame, (v) landmark pose with respect to sensor.

Each logical sensor implementation introduces a different set of parameters ξ to model domain specific sources of distortion, bias or other quantities which have to be known to evaluate the predictor. In addition, there is a fundamental parameter for the logical sensors, the covariance matrix. In fact, this parameter is the most incisive for the correct computation of the state estimation. It indicate how much confidence the filter must put on a certain measure, thus how much is expected to be good the sensor acquisition. Tuning this parameter we can decide to give more importance to data coming from certain sensors. Thus, based on the scenario and on the greatness of the hardware you need to find the correct values in order to rely more on great sensor and give lower importance to the bad ones. Sometimes, it can happen that there is a sensor which is working perfect except for some instants in which it provide very worst measurements. RF, as said before, implements also an outlier rejection, in particular it implements a method called robust kernel. This techniques allows to set a parameter which indicates what is the threshold of acceptable values for a certain sensor (a kind of restrict domain). When the threshold is overpassed the error model of the sensor pass from quadratic to linear, thus the measurement is considered less respect the other sensors data. Thanks to this method, when it happens a case like the one

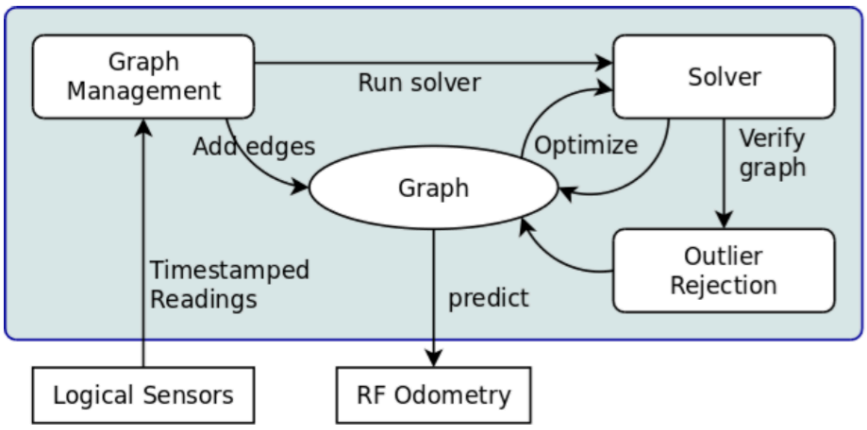


Figura 3.9: ROAMFREE estimation schema.

just mentioned, the outliers are less influent on state estimation and so they do not cause big errors.

Optimizations

RF implements two optimization algorithms: Gauss-Newton and Levenberg-Marquardt. In order to use them the sensor fusion problem must be reformulated as a non-linear, weighted, least-squares optimization. A complete schema of the RF functioning is shown in Figure 3.9

Consider the error function $e_i(x_i, \eta)$ associated to the i-th edge of the hyper-graph and defined as (3.29). Thus, e_i is a random vector and its expected value is approximated as $\overline{e}_i(x_i, \eta) = e_i(x_i, \eta)|_{\eta=0}$. Since e_i can involve non-linear dependencies with respect to the noise, its covariance Σ_η is computed by means of linearization,

$$\Sigma_{e_i} = J_{i,\eta} \Sigma_\eta J_{i,\eta}^T |_{x_i=\tilde{x}_i, \eta=0} \tag{3.30}$$

where $J_{i,\eta}$ is the Jacobian of e_i with respect to η evaluated in $\eta = 0$ and in the current state estimate $\tilde{x}_i$. Equation (3.30) shows the mathematical impact of the sensors covariance Σ_η. In fact, being multiplied with the squared Jacobian and it directly influences the error estimation. Then, A negative log-likelihood function can be associated to each edge in the graph, which stems from the assumption that zero-mean, Gaussian, noise corrupts the sensor readings. Omitting the terms which does not depend on x_i, for the

i-th edge this function reads as,

$$\mathcal{L}(x_i) = \overline{e}_i(x_i)\Omega_{e_i}\overline{e}_i(x_i) \tag{3.31}$$

where $\Omega_{e_i} = \Sigma_{e_i}^{-1}$ is the information matrix of the i-th edge. The solution for the information fusion problem is given by the assignment for the state variables such that the likelihood of the observations is maximum,

$$\mathcal{P} = \arg\min_{x} \sum_{i=1}^{N} \mathcal{L}(x_i) \tag{3.32}$$

It is possible to observe that this is a non-linear least-squares problem where the weights are the information matrices associated to each factor. If a reasonable initial guess for x is known, a numerical solution for $\mathcal{P}$ can be found by means of the popular Gauss-Newton (GN) or Levenberg-Marquardt (LM). Due to the mathematical complexity we do not report here the details of the two algorithms.

3.4 Simultaneous Localization And Mapping (SLAM)

The objective of simultaneous localization and mapping (SLAM) is to build a map and to locate the robot in that map at the same time. We should clarify that for the SLAM problem, it does not matter if the robot moves autonomously or is controlled by a human. The important thing is to build the map and locate the robot correctly. The most basic way to locate a robot is using odometry, but we have written in 3.2 the problem is that odometer error accumulates as the robot moves. Eventually, the error is so large that odometry no longer gives a good estimate of the state of the robot. Then the robot must make observations of some references in the environment to correct the odometer error, assuming the references are static. When the robot returns to observe, these references can reduce the accumulated error. The main work of the SLAM method is to correct the estimation of the robot state and the map. On the other hand, the SLAM is a chicken-and-egg problem because the robot needs a map to locate itself and the map needs the localization of the robot to build a consistent map. Thus SLAM methods must be recursive. Thus, the reason why the SLAM problem is so difficult is because errors in robot and map states affect each other, producing inconsistent maps. In the following section, we will give an overview of several SLAM methods.

From a probabilistic perspective, there are two main forms of the SLAM problem, which are both of equal practical importance. One is known as the online SLAM problem: It involves estimating the posterior over the momentary pose along with the map:

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{3.33}$$

Here x_t is the pose at time t, m is the map, and $z_{1:t}$ and $u_{1:t}$ are the measurements and controls, respectively. This problem is called the online SLAM problem since it only involves the estimation of variables that persist at time t. Many algorithms for the online SLAM problem are incremental: they discard past measurements and controls once they have been processed. The second SLAM problem is called the full SLAM problem. In full SLAM, we seek to calculate a posterior over the entire path $x_{1:t}$ along with the map, instead of just the current pose x_t:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.34}$$

The difference between the two is that full SLAM computes the state only one time while online SLAM compute it incrementally.

There exist many methods implementing the full or the online SLAM, but in general every SLAM method must do three tasks: (i) Landmark detection. The robot must recognize some specific objects in the environment; they are called landmarks. It is common to use a laser range finder or cameras to recognize landmarks, such as corners, lines, trees, etc. (ii) Data association. Detected landmarks should be associated with the landmarks on the map. Because landmarks are not distinguishable, the association may be wrong, causing large errors on the map. Besides, the number of possible associations can grow exponentially over time; therefore, data association is a difficult task. (iii) State estimation. It takes observations and odometry to reduce errors. The convergence, accuracy, and consistency of the state estimation are the most important properties. Thus, the SLAM method must maintain the robot path and use the landmarks to extract metric constraints to compensate the odometer error.

The major difficulties of SLAM are the following:

- High dimensionality. Since the map dimension always grows when the robot explores the environment, the memory requirements and time processing of the state estimation increase. Some submapping techniques can be used to solve it [33][34][35].

- Loop closure. When the robot revisits a past place, the accumulated odometry error might be large. Then the data association and landmark detection must be effective to correct the odometry. Place recognition techniques are used to cope with the loop closure problem [36][37].

- Dynamics in environment. State estimation and data association can be confused by the inconsistent measurements in the dynamic environment. There are some methods that try to deal with these environments [38][39].

In particular we concentrate on laser-based SLAMs. These employ a laser range finder which scans generate the occupancy grid maps of the environment. The laser scans relevant objects and tries to associate them to the ones in the map by comparing different scans (scan matching) in order to extract the constraints fundamental for the mapping and the localization of the robot. This problem is also called front-end problem and is typically hard due to potential ambiguities or symmetries in the environment.

3.4.1 Gmapping

Gmapping is a laser-based SLAM algorithm as described by Grisetti et al in 2007 [40]. It is the most widely used SLAM package in robots worldwide. This algorithm has been proposed by Grisetti et al. and it is a Rao-Blackwellized Particle Filter (RBPF) SLAM approach.

Rao-Blackwellized particle filter

In 3.1.2 we have written the basic versione of the PF, now we describe the optimized version RBPF for SLAM problems.

Let us start from Equation (3.34) and factorize it as

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{3.35}$$

This factorization allows to first estimate only the trajectory of the robot and then to compute the map given that trajectory. In particular $p(m|x_{1:t}, z_{1:t})$ can be easily computed using mapping with known poses since $x_{1:t}$ and $z_{1:t}$ are known.

The RBPF occupancy grid SLAM works as follows, if new control data u_t from the odometry and a new measurement z_t from the laser scanner is available:

1. Determine the initial guess $x_t^{'(i)}$, based on u_t and the pose, since the last filter t update x_{t-1} has been estimated.

2. Perform a scan matching algorithm based on the map $m_{t-1}^{(i)}$ and $x_t^{'(i)}$. If the scan matching fails, the pose $x_t^{(i)}$ of particle i will be determined according to a motion model, otherwise the next two steps will be performed.

3. If the scan matching is successfully done, a set of sampling points around the estimated pose $\hat{x}_t^{(i)}$ of the scan matching will be selected. Based on this set of t poses, the proposal distribution will be estimated.

4. Draw pose $x_t^{(i)}$ of particle i from the approximated Gaussian distribution of the improved proposal distribution.

5. Perform update of the importance weights.

6. Update map $m^{(i)}$ of particle i according to $x^{(i)}$ and z_t.

A more detailed introduction to learn occupancy grid maps and the Rao-Blackwellized particle filter can be found in the Algorithm 5 reported and in Algorithm 5. The RBPF solves the depletion problem, described in 3.1.2 by implementing an adaptive resampling technique, which is performed only when it is needed. The authors also proposed a way to compute an accurate distribution by taking into account not only the movement of the robotic platform, but also the most recent observations. This decreases the uncertainty about the robots pose in the prediction step of the PF. As a consequence, the number of particles required decreased since the uncertainty is lower, due to the scan matching process.

3.4.2 Graph-based methods

These methods use optimization techniques to transform the SLAM problem into a quadratic programming problem. The historical development of this paradigm has been focused on pose-only approaches and using the landmark positions to obtain constraints for the robot path. The objective function to

Algorithm 5 improved RBPF for map learning

Require:
$\mathcal{S}_{t-1}$, the sample set of the previous time step
z_t, the most recent laser scan
u_{t-1}, the most recent odometry measurement
Ensure:
$\mathcal{S}_t$, the new sample set
$\mathcal{S}_t = \{\}$
for all $s_{t-1}^{(i)} \in \mathcal{S}_{t-1}$ **do**
$\langle x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} \rangle = s_{t-1}^{(i)}$
// scan-matching
$x_t^{'(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$
$\hat{x}_t^{(i)} = \arg\max_x p(x|m_{t-1}^{(i)}, z_t, x_t^{'(i)})$
if $\hat{x}_t^{(i)} =$ failure **then**
$x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$
$w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t|m_{t-1}^{(i)}, x_t^{(i)})$
else
// sample around the mode
for all $k = 1$ to K **do**
$x_k \sim \{x_j \mid |x_j - \hat{x}^{(i)}| < \Delta\}$
end for
// compute Gaussian proposal
$\mu_t^{(i)} = (0,0,0)^T$
$\eta^{(i)} = 0$
for all $x_j \in \{x_1, ..., x_K\}$ **do**
$\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
$\eta^{(i)} = \eta^{(i)} + p(z_t|m_{t-1}^{(i)}) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
end for
$\mu_t^{(i)} = \mu_t^{(i)}/\eta^{(i)}$
$\Sigma_t^{(i)} = \mathbf{0}$
for all $x_j \in \{x_1, ..., x_K\}$ **do**
$\Sigma_t^{(i)} = \Sigma_t^{(i)} + (x_j - \mu_t^{(i)})(x_j - \mu_t^{(i)})^T \cdot p(z_t|m_{t-1}^{(i)}) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
end for
$\Sigma_t^{(i)} = \Sigma_t^{(i)}/\eta^{(i)}$
// sample new pose
$x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)})$
// update importance weights
$w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
end if
// update map
$m_t^{(i)} = \text{integrateScan}(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
// update sample set
$\mathcal{S}_t = \mathcal{S}_t \cup \{\langle x_t^{(i)}, w_t^{(i)}, m_t^{(i)} \rangle\}$
end for
$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
if $N_{\text{eff}} < T$ **then**
$\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
end if

optimize is obtained assuming Gaussianity. Since this methods are based on a factor graph, they are able to remember better the old sub-maps and the old localization and thus results more accurate respect to other approaches. Their main disadvantage is the high computational time they take to solve the problem. So they are suitable to build maps off-line. Some methods are: GraphSLAM [41], Square Root SLAM [42], and Sliding Window Filter [43]. pagine 15-20 di "State Estimation and Optimization for Mobile Robot Navigation"

KartoSLAM

KartoSLAM is a graph-based SLAM approach developed by SRI Internationals Karto Robotics, which has been extended for ROS by using a highly-optimized and noniterative Cholesky matrix decomposition for sparse linear systems as solver. A graph-based SLAM algorithm represents the map by means of a graph. In this case, each node represents a pose of the robot

along its trajectory and a set of sensor measurements associated to that pose. These nodes are connected by arcs which represent the robot motion between successive poses. At each new node, the global map is computed by finding the spatial configuration of nodes which is consistent with the constraints deriving from the arcs. In the KartoSLAM version available for ROS, the Sparse Pose Adjustment (SPA) algorithm is responsible for both scan matching and loop-closure procedures . The higher the number of landmarks, the more amount of memory is required, but in case of large-scale environments, graph-based SLAM algorithms are usually more efficient than other approaches.

Google's Cartographer

Googles Cartographer provides a real-time solution for indoor and outdoor mapping in the form of a sensor equipped backpack that generates 2D grid maps with a r = 5 cm resolution. Laser scans are inserted into a submap at the best estimated position, which is assumed to be sufficiently accurate for short periods of time. Scan matching happens against a recent submap, so it only depends on recent scans, and the error of pose estimates in the world frame accumulates. To achieve good performance with modest hardware requirements, this SLAM approach does not employ a particle filter. To cope with the accumulation of error, it regularly run a pose optimization. When a submap is finished, that is no new scans will be inserted into it anymore, it takes part in scan matching for loop closure. All finished submaps and scans are automatically considered for loop closure. If they are close enough based on current pose estimates, a scan matcher tries to find the scan in the submap. If a sufficiently good match is found in a search window around the currently estimated pose, it is added as a loop closing constraint to the optimization problem. By completing the optimization every few seconds, the experience is that loops are closed immediately when a location is revisited. This leads to the soft real-time constraint that the loop closure scan matching has to happen quicker than new scans are added, otherwise it falls behind noticeably. This has been achieved by using a branch-and-bound approach and several precomputed grids per finished submap.

As it can be expected the algorithm implemented by Googles Cartographer is significantly more heavy with respect to gmapping and Karto SLAM, but it is more accurate as it is expected by its complex formulation.

3.5 Localization

Robot localization is the problem of estimating a robots pose relative to a map of its environment. This problem has been recognized as one of the most fundamental problems in mobile robotics [44]. The mobile robot localization problem comes in different flavors. The simplest localization problem is position tracking. Here the initial robot pose is known, and localization seeks to correct small, incremental errors in a robots odometry. More challenging is the global localization problem, where a robot is not told its initial pose, but instead has to determine it from scratch. Even more difficult is the kidnapped robot problem [45], in which a well-localized robot is tele-ported to some other place without being told. This problem differs from the global localization problem in that the robot might firmly believe itself to be somewhere else at the time of the kidnapping. The kidnapped robot problem is often used to test a robots ability to recover from catastrophic localization failures.

Localization can be seen as a problem of coordinate transformation. Maps are described in a global coordinate system, which is independent of a robots pose. Localization is the process of establishing correspondence between the map coordinate system and the robots local coordinate system. Knowing this coordinate transformation enables the robot to express the location of objects of interests within its own coordinate frame (knowing the pose of the robot is sufficient to determine this coordinate transformation, assuming that the pose is expressed in the same coordinate frame as the map). Unfortunately the pose can usually not be sensed directly. It has therefore to be inferred from data. A key difficulty arises from the fact that a single sensor measurement is usually insufficient to determine the pose. Instead, the robot has to integrate data over time to determine its pose.

In the context of robot localization, the state x_t of the system is the robots position, which is typically represented in a two-dimensional Cartesian space and the robots heading direction, $x_t = (xy\theta)$. The state transition probability $p(x_t|x_{t-1}, u_{t-1})$ describes how the position of the robot changes using information u_t collected by the robots wheel encoders. The perceptual model $p(z_t|x_t)$ describes the likelihood of making the observation z_t given that the robot is at location x_t.

3.5.1 AMCL

The adaptive Monte Carlo localization (AMCL) is a method to localize a robot in a given map. It is a concrete implementation of the particle filter with some modifications. Also, it is an adaptive algorithm of the Monte Carlo localization, because the count of used particles is not fixed. The decision of how many particles are used is based on KLD-Sampling (Kulback-Leibler-Divergence). KLD-Sampling is described in [46] or [47]. Algorithm 6 describes the AMCL. It requires the set of particles of the last known state $\mathcal{X}_{t-1}$, the control data u_t for prediction and the measurement data z_t and the map m for the update. The algorithm returns the new state, represented by a set of particles.

An advantage of the AMCL is, that this filter implementation is able to solve the global localization problem, the kidnapped robot problem, as well as localization tracking. Notice, the AMCL can use an arbitrary resampling technique. A second advantage of the AMCL is, that it is able to recover from localization errors by adding random particles to the set $\mathcal{X}_t$ after a specified decade (see: lines 13 and 14).

AMCL is adopted by many robotics since it is implemented in a ROS package. In this latter it allows to set many configuration parameter (see [48]), we will see the most important in the next chapters.

Algorithm 6 Adaptive variant of Monte Carlo Localization

procedure AMCL($\mathcal{X}_{t-1}, u_t, z_t, m$)
  static w_{slow}, w_{fast}
  $\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
  for all $m := 1$ to M **do**
    $x_t^{(m)} = \text{SampleMotionModelOdometry}(u_t, x_{t-1}^{(m)})$
    $w_t^{(m)} = \text{MeasurementModel}(z_t, x_t^{(m)}, m)$
    $\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle x_t^{(m)}, w_t^{(m)} \rangle$
    $w_{avg} = w_{avg} + \frac{1}{M} w_t^{(m)}$
  end for
  $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
  $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
  for all $m := 1$ to M **do**
    with probability $\max(0.0, 1.0 - w_{fast}/w_{slow})$ **do**
      add random pose to $\mathcal{X}_t$
    else
      draw $i \in \{1, ..., N\}$ with probability $\propto w_t^{(m)}$
      add $x_t^{(i)}$ to $\mathcal{X}_t$
    endwith
  end for
  return $\mathcal{X}_t$
end procedure

4.
Navigation system for the GRAPE robot

In this chapter we describe the navigation system that we designed for the GRAPE robot. We start justifying the choice of the robot platform and the sensors that we decided to adopt. Then we provide an overview of the structure of the solution we propose. In particular, we take advantage of the ROS visualization tools, in order to show the entire architecture through a set of graphs. Then, in a top-down fashion, we detail the different subsystems that compose the navigation system: the simulation module in section 4.4, the GRAPE robot module in section 4.3, the odometry estimation and sensor fusion module in section 4.5, the mapping module in section 4.6 and the autonomous navigation module in section 4.7.

4.1 GRAPE robot

The GRAPE robot (Figure 4.1) is, in the current version, a skid-steering four wheeled mobile robot based on the Husky platform [49]; It sports a frontal Hokuyo laser range finder and a Velodyne Puck on the extern, and inside it hosts a common PC with Wifi module and an IMU sensor. In addition, the robot mounts a robotic arm, the Kinova Jaco^2 [50], plus a vertical sensor support to which a GPS antenna and a Zed stereo camera are fixed. We provide a brief description of the listed in sensors in Table 4.1 and we report their technical specification in Appendix B.

Type	**Model**	**Note**

GNSS	OEMstar + gps-701-gg antenna	This sensor will provide two types of data sources for localization. A standalone GNSS + EGNOS positioning solution and raw data to be fused with external sources in a differential solution to reduce the expected error of a low-cost GPS received. Galileo compliant receivers will be evaluated in a later stage of the project
Laser 2D	Hokuyo UTM-30LX-EW	An outdoors tested, 30m range, 270o FOV scanning laser mounted horizontally in the front of the vehicle in order to detect plants (i.e. trunks) and potential obstacles. This laser will also be used to build a map of the static elements of the environment and use such map in order to refine robot position estimate through an Adaptive Monte Carlo Localization (AMCL) algorithm.
IMU	Xsens Mti -10	Inertial data are used to complement at higher frequency the position information provided by the GNSS. Their estimate of robot motion drifts over time, but this can be compensated by sing sensor fusion techniques such as Kalman Filtering or windowed pose graph optimization, e.g., through the ROAMFREE sensor fusion library.
Stereo camera	Stereolabs Zed HD	An outdoors tested, 30m range, 270o FOV scanning laser m
Lidar 3D	Velodyne puck lite	

Tabella 4.1

The choice of the Husky platform has been enforced by the GRAPE project specifications. Indeed, the project partner Eurecat (Spain) has been in charge of providing the physical platform. The choice has fallen on Husky for many reasons; First, it is designed to work outdoor and it is quite resistant

Figura 4.1: GRAPE robot prototype

to the different weather condition, but especially, it mounts four knobby wheels with a skid-steering behavior that, as written in section 3.2.3, are suited for rough terrain navigation. The second reason is its shape, indeed it is composed by a body which can host the wiring, the onboard PC and some sensors and in general it is highly customizable while being easy to be transported given its weight and size. The third reason is that the Husky's company provides an updated open-source repository in which are stored the ROS packages implementing all the basic function of the robot.

4.2 Navigation system overview

Being in the contest of field robotics, the aim of our navigation system is to be innovative but also reliable since it has to guarantee to work in a difficult scenario like the vineyard one. Usually, the terms innovation and reliability generate a kind of duality, an example is the beta release of a software. Thus, it is very difficult, in general, to obtain an innovative and reliable system. For this reason we set up the navigation system construction in an incremental

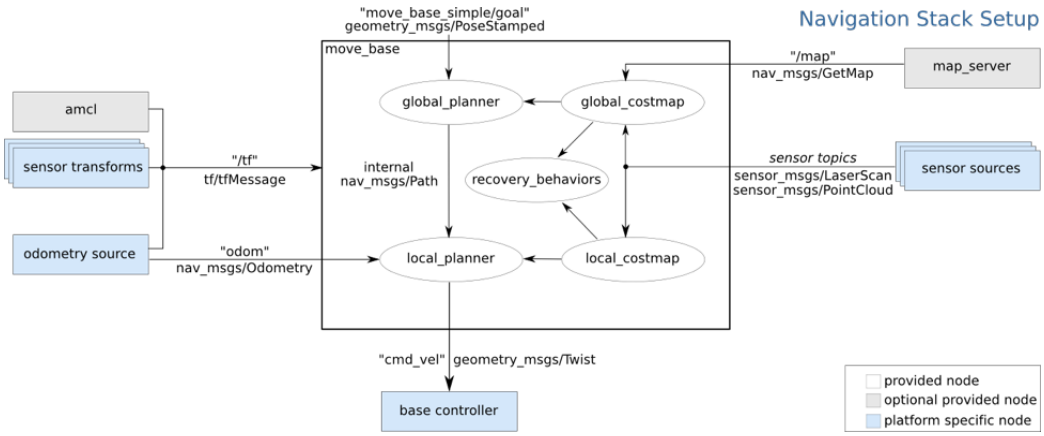


Figura 4.2: ROS standard navigation stack

way. We started from a simple but very stable solution, and then we added one piece at time until we reached the final architecture we are presenting in this section.

The implementation of such approach has been possible thanks to the ROS structure (see Appendix A), in fact, as illustrated in the appendix, it allows to create very modular solutions in which the different components can be easily added or deleted. In principle, we attempted to build a general solution, which could be adapted to other scenarios without much effort. At the end the proposed solution is highly configurable, and we tuned all the parts to reach the best configuration in our specific case.

The basic solution from which we started is the standard ROS navigation stack which is reported in Figure 4.2. Our navigation system follows this schema, especially it maintains its modularity. Based on it, we can explain our navigation system logic by analyzing the logical blocks in the figure:

Odometry source The odometry source provides the estimated odometry (explained in section 3.2), thus the robot position with respect to its starting pose. The source can be a single sensor, usually the wheel encoders, or it can be a sensor fusion system which integrates more sensors data to generate a more accurate odometry. During the project we investigated and compared three different source: (i) simple wheels encoder node; (ii) EKF sensor fusion node; (iii) Factor graph filter sensor fusion node. The sensors that we merged in (ii) and (iii) are the wheels encoder, the IMU and the GNSS. Thanks to the ROS modularity these three methods can be easily

switched, in particular we are interested in comparing the (ii) and (iii) which are more accurate and reliable with respect to (i). We show the filters comparison in the next chapters. We made these methods live together; the odometry is published both as a value in a topic and as a frame in the TF tree. In particular the odometry frame links the global frame (usually map frame) and the robot frame.

Sensor sources This block refers to the sensors able to perceive the robot external environment: the lasers (Hokuyo, Velodyne). Laser scan data or point cloud data are sent to the Navigation stack for mapping the environment or to localize the robot in it. Furthermore they are used to create the local costmap which is an occupancy grid sub-map which describes the obstacles surrounding the robot. In the final navigation architecture only data coming from the 2D frontal laser are used, indeed the cost of a 3D LIDAR might be too high for a final product so we were interested in validating if a 2D source is sufficient to allow for robust navigation on rough terrains.

TF (transformation frames) TF defines the relationships between coordinate frames (either between sensor frames and robot frame or robot frame and odometry frame etc). The geometry behind this roto-traslations is implemented by the `tf` ROS package. We report the complete explanation of the TF library in Appendix C.

Amcl and map server These two blocks are optional in the standard navigation stack, but they are necessary for the autonomous navigation. They are strictly connected, in fact the map server is responsible to provide a static map of the environment, while amcl is the localization method, thus it localize the robot in the map. It tries to adjust the odometry frame (to which robot pose is referenced) with respect to the map frame exploiting the laser scans.

Local and Global Costmaps The local and global 2D costmaps contain the information representing the projection of the obstacles in a 2D plane (the ground), as well as a security inflation radius, i.e., an area around the obstacles to guarantee that the robot will not collide with any objects, no matter what is its orientation. These projections are associated to a cost, and the robot objective is to achieve the navigation goal by creating a path

with the least possible cost. While the global costmap represents the whole environment (or a huge portion of it), the local costmap is, in general, a scrolling window that moves in the global costmap in relation to the robot current position and it is updated online with the incoming laser scans.

Local and Global Planners Local and global planners do not work the same way. The global planner takes the current robot position and the goal and traces the trajectory of lower cost with respect to the global costmap. The local planner is responsible for updating the global trajectory in case that obstacles not present in the costmap are detected by the robot. It generates a new trajectory which avoids these obstacles and that deviates the least possible from the global trajectory.

Base controller The base controller has the function to convert the velocity commands, which indicate the linear and angular velocity of the robot into the corresponding motor velocities of the robot (e.g., motor of the single wheel). It is the output block of the navigation stack since it receives velocity commands generated by the planners and it produces a command that makes the robot actually move.

The final navigation system that we built is shown in Figure 4.4. There, we can observe how the logical blocks described above are present also in our solution. We split the architecture in three macro-modules each of which solve a different problem. In the next sections we explain the detailed functioning of these three modules (GRAPE robot module, Sensor fusion and odometry estimation module and autonomous navigation module), in addition we explain other two modules: the simulation module and the SLAM module which can be substituted, respectively to the GRAPE robot module and to the autonomous navigation module.

Looking at the Figure 4.4 we can also note the presence of some nodes that are shared between the modules, some of them are crucial while some others have a secondary importance respect to the navigation but they are needed to make possible the connection between more important nodes.

- `teleop_twist_joy`: The purpose of this node is to provide a generic facility for tele-operating Twist-based ROS robots with a standard joystick. This node provides no rate limiting or autorepeat functionality. It subscribes to the Joystick messages (not reported in the figure) and

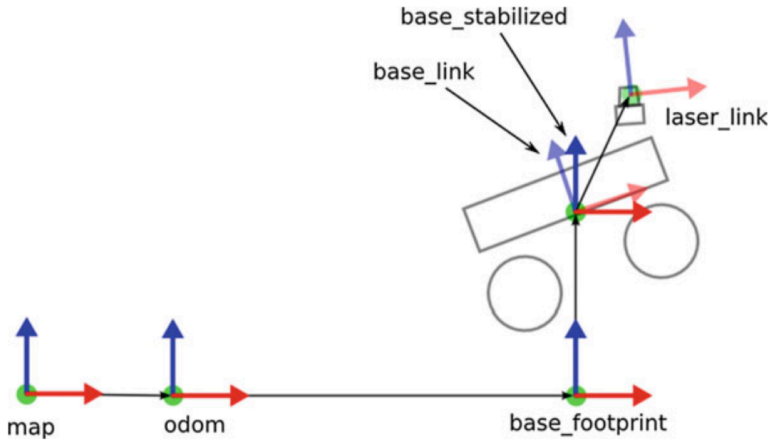


Figura 4.3: Example of the transformation frames during the navigation of a robot

translates them into velocity commands. Then it publishes the velocity commands in a topic (`/joy_teleop/cmd_vel` in figure). The presence of this node is important since the Joystick is fundamental for emergency cases. It allows to manually intervene when the robot is stuck or when something is not working properly.

- `/twist_mux`: This node subscribes to a list of topics publishing geometry_msgs/Twist messages and multiplex them using a priority-based scheme. It also supports timeouts for each input and locking by means of topics that publish std_msgs/Bool messages. Priority is currently given to manual operation which overrides the velocity command sent by the autonomous navigation system.

- `/tf`: It publishes the transformation between frames that are broadcasted by other nodes. Then it create the TF tree using the received TF (the tf tree of our architecture is shown in the Figure 4.5).

4.3 GRAPE robot module

In this section we illustrate the nodes and the most important topics which allow ROS to interface with the robot hardware: sensors and actuators. The Figure 4.6 shows a simplified version of the rosgraph of such module.

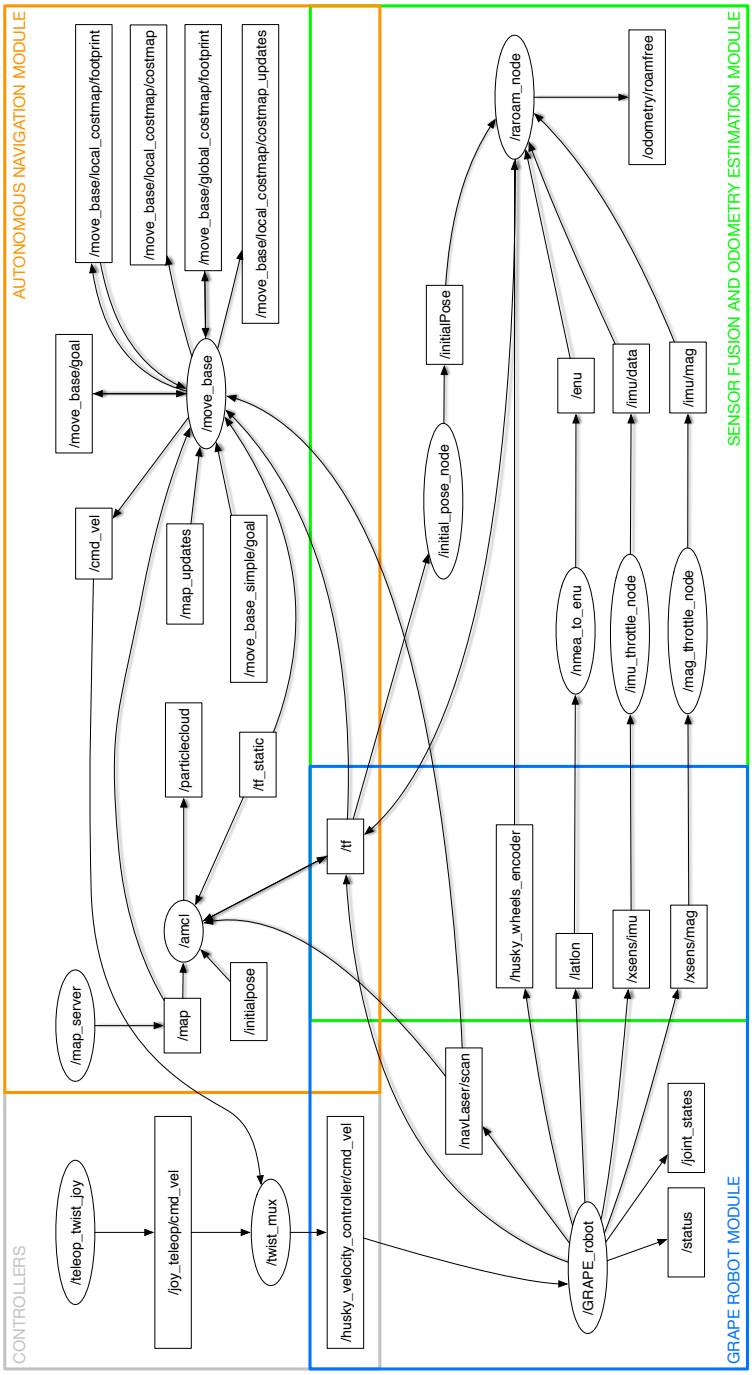


Figura 4.4: Final navigation system architecture. Circles are nodes while rectangle are topics. The graph is simplified of some nodes or topics for clearness reason

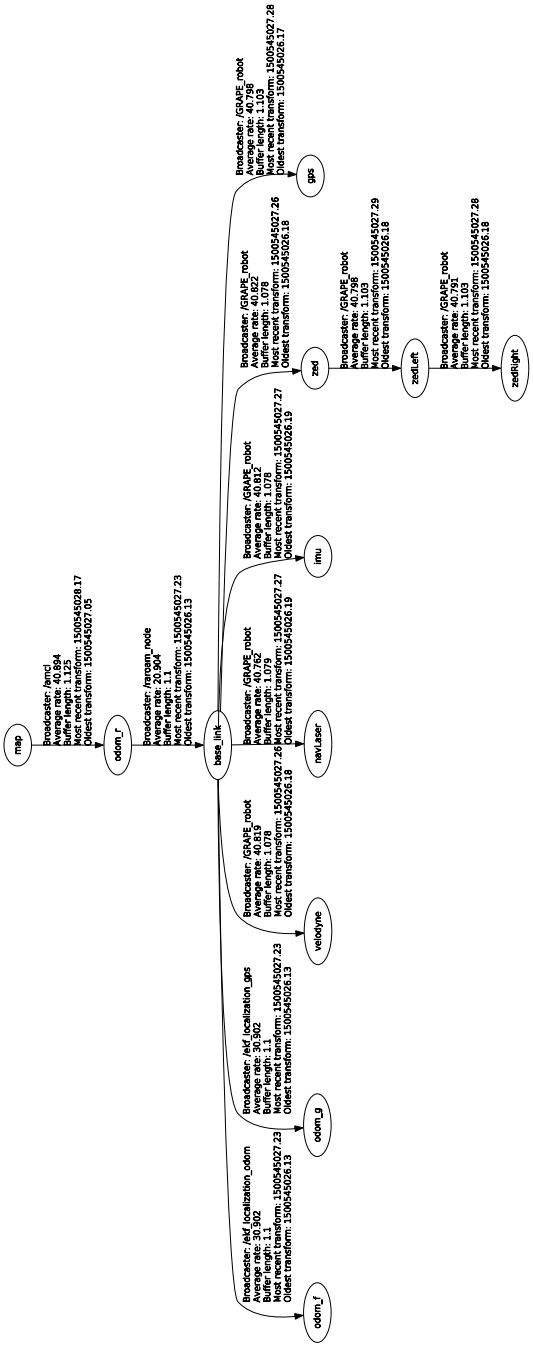


Figura 4.5: TF tree relative to the final architecture

- `/Husky_velocity_controller/cmd_vel` : Its a `ros_controller` topic which publishes the linear and the angular velocities that are listened

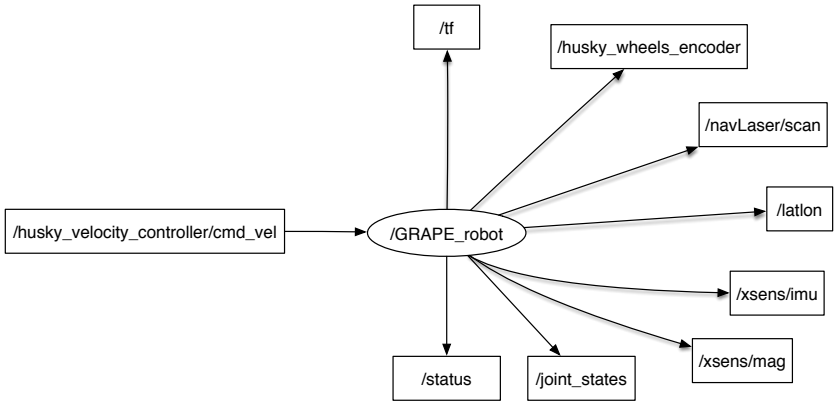


Figura 4.6: The hardware interface rosgraph. For clarity reasons some driver nodes has been omitted

from `twist_mux` node. It is the input for the robot wheel actuators.

- `/joint_states` : This topic publishes the current states of the robot joints, thus it give information about the current configuration of the moving components of the robot. This topic can be used to update the some TFs.

- `/xsense/imu` and `/xsense/mag`: The first topic corresponds to the gyroscope and accelerometer sensor while the second represents the magnetometer. Both topics publish the readings of the sensors with a fixed rate. In particular the topic relative to the imu publishes messages in which it specifies the gyroscope data the accelerometer data, but also an heading(orientation) estimate which is intrinsically computed in the sensor driver. We do not use this value in our proposed solution, since being an estimation we do not know its error model, and thus we cannot optimize it correctly using ROAMFREE.

- `/latlon`: it publishes the GPS readings in LLA (Latitude, Longitude, Altitude) coordinates.

- `/Husky_wheels_encoder`: It publishes the rotational velocity of the right and left side wheels. The rotational velocity is internally computed in the encoder driver starting from the number of tick measured directly by the encoder. Being a skid-steering robot, it publishes only one velocity for each side. It suppose that same side wheels have same

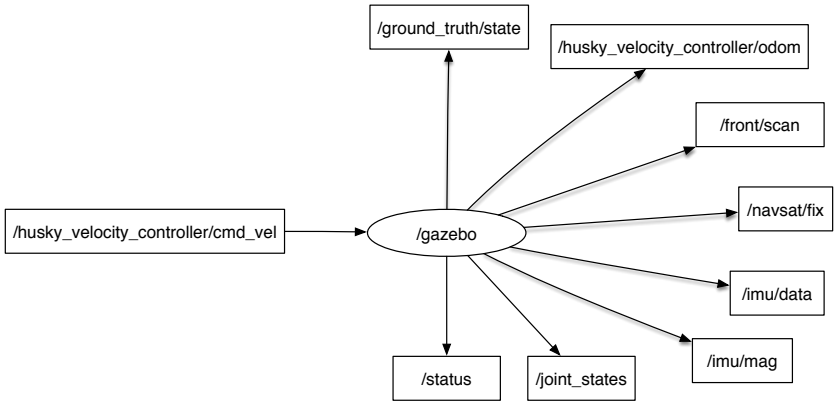


Figura 4.7: Gazebo simulation node and its topics

rotational velocity. In practice each wheel has its own actuator (motor), thus the velocity could be slightly different even between same side wheels.

- `navLaser/scan`: This topic is created by the Hokuyo laser node (not in figure). It publishes messages of the type `sensor_msgs/LaserScan`. This scans should represent 270° of the environment, in practice we reduced this range by few degrees because it intercepts the robot body.

4.4 Simulation module

In this section we show (see Figure 4.7) which are the differences between a simulated robot and a real one from the system architecture point of view. It can be observed that the two modules can be swapped without compromise other part of the system. This allows to work with the same methods both in simulation and in real tests. In field robotics the importance of the simulation is even more accentuated since it is difficult and expensive to test the system performance directly on the field.

- `/gazebo`: It represents the integration of Gazebo simulation in ROS environment. It communicates with other nodes through Gazebo plugins topics (see Figure 4.7).

- `/ground_truth/state`: This topic publishes the correct (ground truth) pose of the robot in the simulation environment. It specifies the robot position with respect to the `map` frame which in this case corre-

sponds with the `odom` frame (the ground truth does not have error in localization).

- `/Husky_velocity_controller/odom`: It has the same scope of `/Husky_wheels_encoder` described above, thus provide wheels odometry, but in this case instead of publishing the wheels rotational velocity it publishes directly the robot linear and angular velocities. It is a kind of pre-odometry estimation computed only with the wheel encoders acquisitions. We use it only in the simulation solution since here it is implemented directly in the official Gazebo plugin for the Husky robot or in our real solution when the `robot_localization` package is adopted.

-

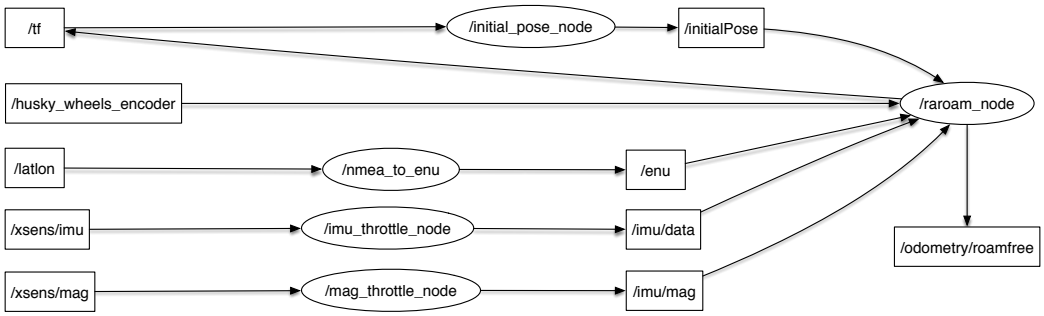
- `/imu/data` and `/imu/mag`: These are the topics relative to the simulated IMU sensors. We exploited the `hector_gazebo_plugins` package which provides common sensors gazebo plugins that are highly configurable. In fact they allow to set the noise, the rate and may other parameters.

- `/navsat/fix`: This topic is relative to the GPS sensor plugin. Thus it publishes the LLA coordinate of the robot. Again it is taken from the `hector_gazebo_plugins` package.

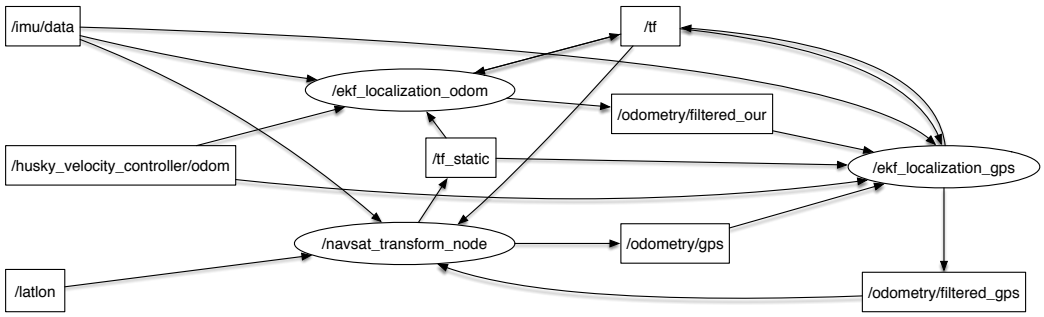
4.5 Sensor fusion and odometry estimation module

This module contains the main logic of our navigation system. In Chapter 3 we explained the theory behind the odometry estimation and the sensor fusion, in particular we highlighted how a correct estimation of the odometry is fundamental for the navigation or mapping module in order to wok properly. For this reason we dedicated to the implementation, configuration and tests the majority of the time. We reasoned about which should be the best method to integrate our sensors, differing in accuracy, reliability, acquisition rate and type of domain measured.

As mentioned in 4.1 the GPS is a really important source when the satellite signal is strong enough, thus we tried to emphasize the GPS advantages while limiting its drawbacks. We exploited this intuition both in the



(a)



(b)

Figura 4.8: The sensor fusion and odometry estimation node in the two variants. (a) shows our ROS implementation of ROAMFREE (the `raroam_node`), (b) represents the nodes of `robot_localization`.

`robot_localization` configuration and in the ROAMFREE implementation. In this latter, thanks to the graph-based approach and to the outlier rejection module intrinsic of ROAMFREE we are able to limit the influence of the GPS when it provides bad measurements. Then, about the IMU exploitation we have reasoned on its importance for the orientation estimation. In fact, the gyroscope and the magnetometer, if correctly configured, gives the right orientation of the robot. This is very important since the most difficult part of the navigation in the vineyard is turning between two vine lines. During this movement there are few references in terms of landmarks and the robot data coming from the wheels encoder are not accurate due to the variability of the slippage of the different side wheels.

The wheel encoders are important during the straight navigation especially supposing an acceptable terrain. Indeed, they become a bad information source if the robot is stuck or if it stuck against an obstacle. In these case the third not mentioned component of IMU, the accelerometer performs a

fundamental role. It is able to give information about the robot acceleration, that is zero when the robot is blocked.

As in the previous module we illustrate the tasks of the nodes shown in Figure 4.8. We start from the ROAMFREE implementation depicted in Figure 4.8a:

- `/raroam_node` : This node implements a factor graph based filter exploiting ROAMFREE (hereafter abbreviated as RF). `/raroam_node` builds the factor graph filter combining wheel encoders, accelerometer, magnetometer, gyroscope, and GPS sensor (in ENU coordinates). The result is an estimated odometry that considers all the cited sensors and which is expected to be robust to sensor noise or missing measurements. The published odometry topic is `/odometry/roamfree` and refers to the frame `odom_r`. This node represents the core of our sensor fusion and odometry estimation system, as explained in section 3.3.2 it optimizes the error models in the factor graph to obtain an accurate odometry. Thus, the grade of accuracy depends also on the number of optimization iterations that is arbitrarily decided and on the length of the integration window (fixed lag).

- `/nmea_to_enu` : This node translates a global position expressed in terms of LLA (Longitude, Latitude, Altitude) to one expressed in ENU (East, North, Up) relative to a new frame arbitrarily decided (usually the frame is fixed geographically near to the robot operation position). Thus the GPS absolute position becomes a relative position. This helps the math and the measurements fusion.

- `/imu_throttle_node` and `/mag_throttle_node`: These nodes are two ROS `throttle` nodes. A `throttle` node reduces the frequency of a desired topics, thus it subscribes the desired topic and it re-publish the messages that arrives nearest to its frequency rate. It ignores all the other messages. So `/imu/data` and `/imu/mag` correspond to the limited rate version of `/xsens/imu` and `/xsens/mag` respectively. This node is required in our solution, because the RF factor graph has only a master sensor which publishes the pose nodes and all the other factors nodes have to be added with a lower or equal rate with respect to the master sensor frequency.

- `/initial_pose_node` : This node computes the initial pose of the robot given the sensors measurements. To do this, it requires an absolute position (x, y, θ), thus it extract the (x, y) position from the GPS sensor and the orientation θ from the magnetometer. It listen these sensors for an arbitrary time interval (usually 30sec or 1min), in order to have a more reliable and smooth initial pose. The initialization is required from RF, which use it to create the first pose node in the factor graph filter. If the initialization is completely wrong then RF, since it uses an optimization algorithm, struggles to reach fast the correct estimation. If there is not this node, the initialization must be done manually through a ROS parameter.

Considering the `robot_localization` variant in Figure 4.8b:

- `/ekf_localization_GPS`: This node is the core odometry estimation node in the considered architecture. As in `/raroam_node` it takes sensors measurements in input and publishes a final odometry through the topic `/odometry/filtered_GPS` which refers to the broadcasted frame `odom_g`. It implements an EKF filter (see section 3.3.1) with which it merge the different sources data. In particular the sources are the GPS, `/odometry/GPS`, the imu `/imu/data`, the wheels odometry `/Husky_velocity_controller/odom` and another filtered odometry `/odometry/filtered_our`. The reason of the latter is its reliability, in fact it can happen to loose of the GPS signal, in such cases it is useful to robust the estimation with an odometry that is already estimated from multiple sensors but not from the GPS.

- `/navsat_transform_node`: This node is also part of the robot_localization package, and its scope is similar to nmea_to_enu, except for the fact that it produces a GPS position encapsulated in a `nav_msgs/Odometry` message that refers to the robots world frame.

- `/ekf_localization_odom`: This node is very similar to `/ekf_localization_GPS`, the only difference is that it doesn't exploit the GPS acquisitions. The final odometry computed is published through the topic `/odometry/filtered_our` and refers to the broadcasted frame `odom_f`.

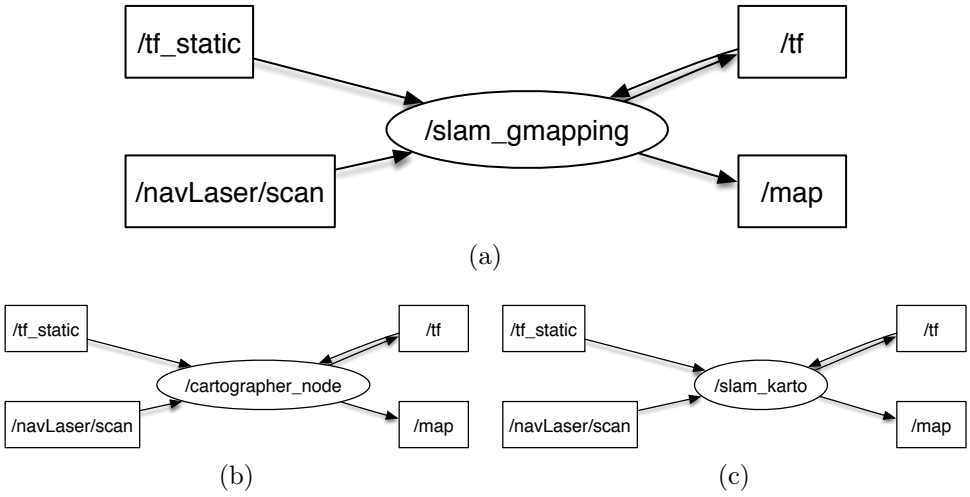


Figura 4.9: The `SLAM_node` in the three variants. (a) Gmapping, (b) Google's Cartographer, (c) KartoSLAM.

4.6 Mapping module

The SLAM module is needed when the robot goal is to map the environment. In our project we expect to map the vineyard once and then let the robot navigate autonomously with the created map. During the mapping the robot is manually driven along the vineyard lines both in simulation and in real tests. As mentioned in the previous chapter we tried three different SLAM tools: gmapping, Google's cartographer and kartoSLAM. In our definitive architecture the tool chosen is gmapping, but we now show how these three tools can be easily swapped. In fact, it is enough to observe Figure 4.9 to confirm this statement.

Since they have the same kind of subscribed and published topics we describe the logic functioning of a generic `SLAM_node` representing all of them.

- `SLAM_node`: This node is able to create a map and simultaneously localize the robot in it, exploiting the robot estimated odometry and the laser sensor. As it can be observed in Figure 4.9 there is no trace of any estimated odometry topic, and the only subscribed topics are `/tf` and (`/navLaser_scan`) relatives to the laser scans. This is because SLAM algorithms take advantages of the estimated odometry in terms of TF, thus their aim (as explained deeply in 3.4) is to adjust the

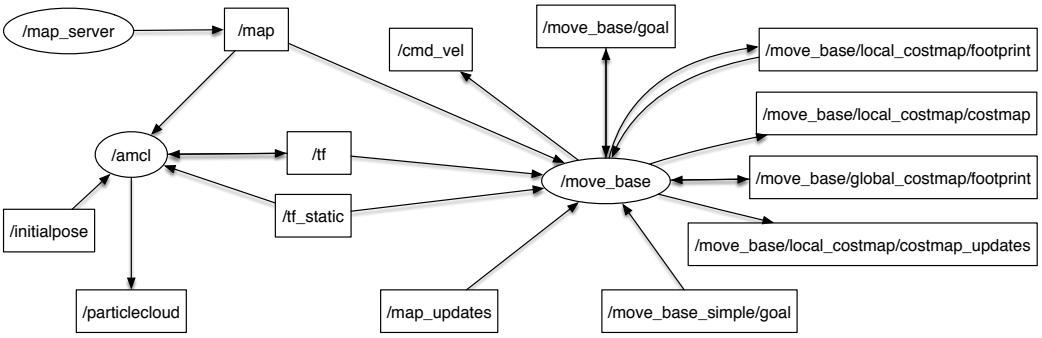


Figura 4.10: The autonomous navigation module

odometry frame respect to the map frame in order to match the laser readings with the position in the map.

4.7 Autonomous navigation module

This module is is able to make the robot navigate autonomously. It respects the standard ROS navigation stack (Figure 4.2) structure. In fact, it is composed by the three nodes `/map_server`, `/amcl` and `/move_base`. We decide to adopt this standard structure since it is highly used in the ROS community and thus, it has shown extensively its great performance and reliability. Furthermore, both the `/amcl` and the `/move_base` package allow for an infinite number of possible configuration since they have many configuration parameters. We tuned only some of them since the performance with most of the default values resulted to be acceptable (we show the parameters values we used in the next chapters). The fact that standard values perform quite well is not casual since, especially for `/amcl`, in the documentation it is specified that the default parameters values have been chosen after a very long tuning phase.

As for the other modules we analyze the nodes in Figure 4.10.

- `/map_server`: It loads the map we indicate and it publishes it in the topic `/map`.
- `/amcl`: This node is able to localize the robot in a map exploiting data from the laser and the estimated odometry (in terms of TF). Thus it broadcasts the transformation between the map frame and the odometry frame.

- `/move_base`: This node performs all the tasks needed to make the robot navigate autonomously. In fact it publishes the local and the global costmaps which are used by the planner to create a valid path to reach the desired goal. This node publishes also the plans created by the local and the global planner.

5.
Simulation Experiments

Building robot navigation system needs a lot of experiments, thus is very important to have a method to make tests without working directly on the real robot. This method is the simulation.

The first thing that we need to virtually reproduce is the working environment of the robot, the vineyard. In addition, since the aim of the simulation is to make trusted tests, we have to build a world that correspond to a worst case scenario. This means to build a simulated vineyard in which the vines a similar to the real ones for dimension and shape and the ground have to be rough.

After the environment is fundamental to create an appropriate model of the robot. Usually, for commercial robots the 3D model is provided by the seller. Further, especially for research, are often used the same robots (also to have a valid comparison with other research works), thus their simulation models are diffused in the robotics community. When this does not happen a new robot model have to be created from scratch (there exist some specific editors). The simulation model of the robot must contain also all the desired sensors. For this latter as for the commercial robots, is often present a ready-to-use package in which is possible to specify the amount of measurements error. Usually the simulated sensors noise is generated with a Gaussian distribution.

5.1 Gazebo simulator

For the simulation we used Gazebo simulator. Gazebo can simulate multi-body dynamics, including interactions between bodies such as impact, using several existing physics engines. In Gazebo, both the robot model and en-

vironment model must be defined. Thus it allows to create 3D models of arbitrary environments and robots and their sensors. Figure 5.1a shows an example of a 3D model of the environment and a 3D model of the GRAPE robot.

To create a 3D model of a robot, the Unified Robot Description Format (`URDF`) file format can be used. `URDF` is a `XML` based description format. In general, the root link for the description is the base of the robot. All the leaves of this tree are sensors, wheels and other components. Using the `XML` macro language `xarco`, the creation of such `URDF` is simplified. For example, if all wheels of a robot are equals in physical properties and looking, a macro for the them can be created and you has only to specify where each wheel have to be set by calling it with the `xacro`.

In addition, 3D models of an arbitrary shapes can be included via the use of `COLLADA` files. `COLLADA` is also a `XML` based description language, like URDF, but it is used for 3D models of building and other objects. `COLLADA` files can be imported into the Simulation Description Format `SDF` files of Gazebo, which is also `XML` based. `SDF` is the internal format of Gazebo to store the different models and worlds.

Simulating open kinematic chains has been fairly well established in Gazebo. However, simulating closed chain mechanisms presents some technical challenges. Closed kinematic chains consist of a series of rigid bodies connected by joints, where a child link connects back to the parent link. They are typically used to generate a desired output motion or force at one link from the input at another link, and are the basis of many mechanisms.

The main advantage of Gazebo is that it easily interfaces with ROS. Indeed Gazebo is treated as a ROS node describing the hardware layer (as illustrated in section 4.4). ROS then communicates with these simulated hardware nodes. This approach allows for a physical system to be simulated in conjunction with the robot control algorithms. The sensors are simulated in Gazebo through the plugins which specify their technical features and their behavior. In the ROS structure, these plugins result to be topics that are published by Gazebo node, as shown in the Figure 4.7 in the previous chapter.

Finally a consideration about the Gazebo performance; in fact, when it is used with the GUI and a complex model is simulated in it, it becomes a very heavy process for the PC and often this reduces the simulation rate. If this decrease too much the simulation results unusable for activity in which

human interaction is required.

5.2 Simulation models

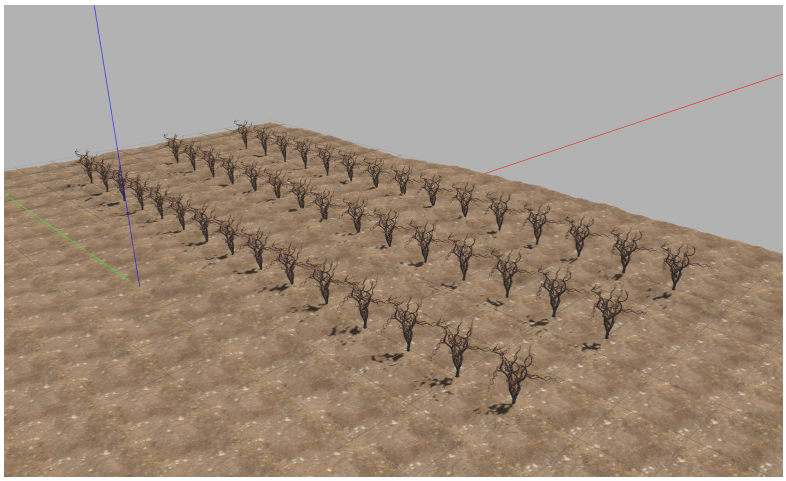
As mentioned in the previous section, an object is added to the simulation through a descriptive model which indicates its parameters. It follows that objects which have a not well-defined shape in the reality are more difficult to be modeled and often this leads to the creation of a simpler model that acceptably approximate the real one.

We can summarize the problems linked to the simulation phase as problems of this thesis due to the difficult reproduction of the real world, accentuated in some scenario (like our) and problems due to the not complete reliability of the simulated results (common problem in the whole research world).

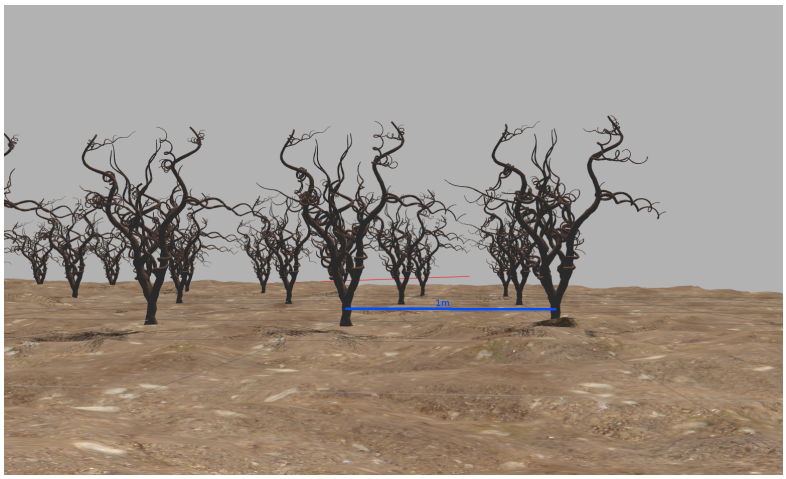
5.2.1 Environment and robot models

The simulation environment consists of an irregular terrain that simulates the rough vineyard soil, and three rows of vines, Figure 5.1a. We used a 3D vine model without leaves since the dispenser deployment is done after pruning tasks. The irregular terrain is a rectangular surface created using an `heightmap` file; it consists of the repetitions of a 129x129px image which stores, in each pixel, the height value of the corresponding point in the real terrain map. This produces small variations of the height in certain parts of the surface, resulting in an irregular surface (see Figure 5.1b). For rendering purposes the `heightmap` is sided by a texture that reproduces the appearance of a real vineyard terrain.

To simulate the vineyard plants, we used a 3D vine model. Our scenario has 3 rows of vines (16 plants per row), with 1m of separation between each plant, see Figure 5.1b. We followed the standard distribution that we usually find in real vineyards. This environment can challenge the fluidity of the simulation execution (it becomes too heavy), so we built also a downsampled version of it in order to manage the daily simulations with a standard computer. We simplified the irregular terrain to be composed by 33x33px patches instead of 129x129 (or 17x17 for some specific simulation) and the vine model has been downsampled in terms of number of faces. The result is a lightweight environment having the appearance in Figure 5.2.



(a)



(b)

Figura 5.1: The environment model simulated in Gazebo

The robot model we used in simulation is really similar to the real prototype except for some differences due to the fact that we changed the position of the sensors in the GRAPE robot during the last field tests. The robot model is based on the Husky URDF model with the additions of our sensors.

5.3 Odometry filter and sensor fusion evaluation

In this section we show the tests we have done to determine the best sensor fusion odometry estimator and the results we obtained by these tests.

(a) (b)

Figura 5.2: The difference between the accurate vine model (a) and downsampled model (b)

The tools we compared are robot_localization and ROAMFREE, which implement two different state estimation approaches (EKF and factor-graph filtering respectively).

5.3.1 Evaluation procedure

Before comparing the two methods we describe here the method by which evaluate their performance. This is not a simple operation; there exist many articles [51] in which navigation or mapping tools comparison is done but each of them adopts a different evaluation metric. In general, there is not a standard way to compare two odometry estimators.

We decided to adopt a very practical method. To understand it, we have to remember the definition that we gave for the localization problem in section 3.5. We stated that localization can be seen as a problem of coordinate transformation. Reasoning in this direction we can evaluate an odometry estimator based on the distance between the odometry frame and the global frame (map frame). Actually if you think about this evaluation metric, it corresponds to measure how much the pose estimated in the odometry estimation is far from the real pose of the robot in the map. Thus, we must have a ground truth localization which indicates the real position of the robot. In simulation this can be obtained by publishing through Gazebo the actual robot pose in the simulated environment and its reference frame.

We have another problem in evaluating the distances between multiple odometry frames. This problem regards the TF tree structure, in fact inserting into it many odometry frames all connected to the robot frame is not possible. The tree, to remain a tree, cannot allow multiple parent nodes. Thus the only way to have multiple odometry sensors living simultaneously is to have one of them as parent of the robot frame and the others as sons of the robot frame. To do this we have to invert the transformation published by odometry estimators except for the one we want to have as parent for the robot frame.

To clarify the reasoning which can seems difficult if explained with words, we take advantage of Figure 5.3. There, the `odom` frame is published by the node `/odom_bl_tf` which is a node that takes in input the ground truth pose and broadcast its frame. The frames relative to the three methods we are comparing (two of them are different configurations of robot_localization) are published in the inverse form and are the sons of the robot frame `base_link`.

5.3.2 Tests and Results

During the development of the architecture we made many trials to find the best configuration for the sensor fusion odometry estimators. First we tuned the robot_localization estimator in order to find the most stable configuration. In this estimator there are not many parameters to be tuned, since even if it has a lot of parameters only few of them really influence the method performance.

The most influential parameters are relatives to the EKF sensor fusion algorithm, in fact they can specify which dimensions of the state vector are affected by each sensor, and the type of integration and the admitted delay of the measurements etc.. To have a complete view of such parameters, consult the official documentation of the package at [52].

After the inclusion of robot_localization in our system, we started experimenting with ROAMFREE. Here the configuration resulted significantly more difficult since all the parameters in RF strongly influence its performance. In particular we struggled with the tuning of the sensors covariances; indeed from section 3.3.2 we know the sensors covariance importance in the error computation, thus even a small variation of the covariance of a sensor causes a relevant impact to the model performance. If the covariance is high

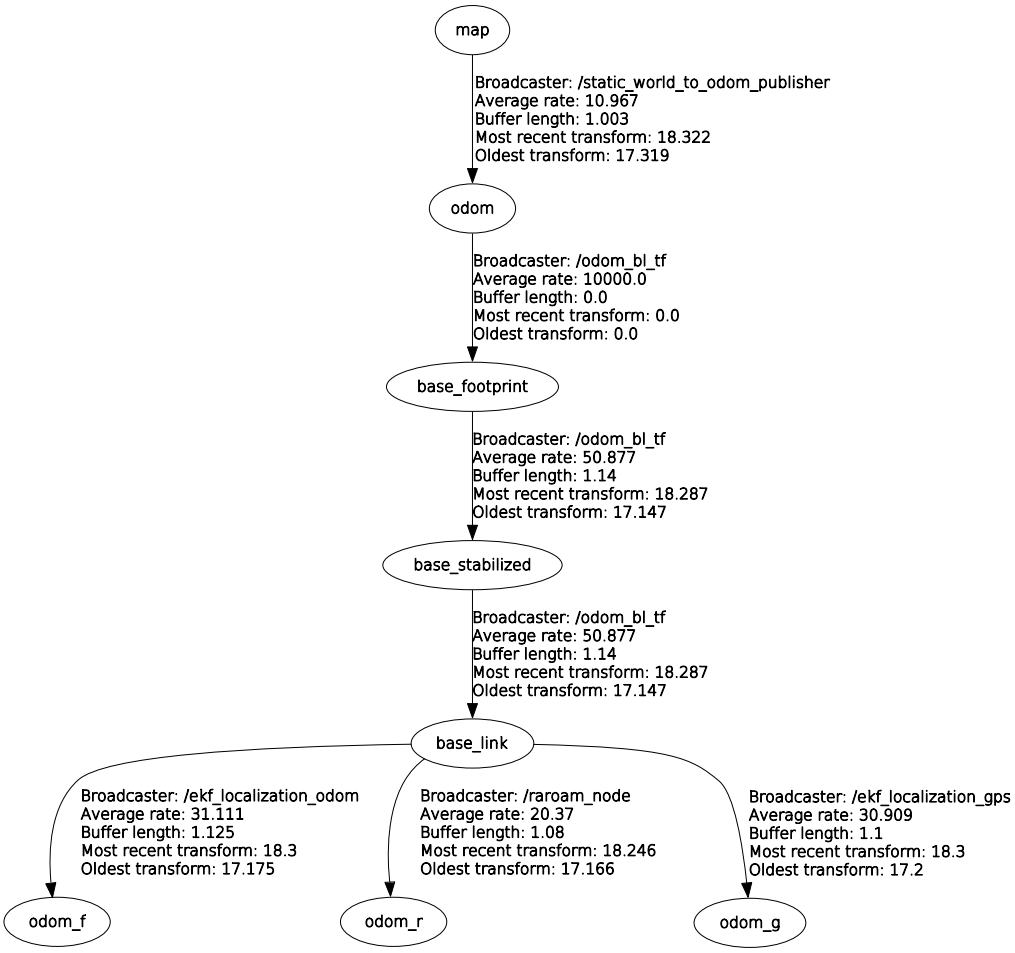


Figura 5.3: TF tree of the simulation evaluation architecture

the sensor has little consideration, viceversa if it is small the sensor highly influences the state estimation. So, to reach good performance we have to give high importance to accurate sensors and low importance to others. Besides the covariance another important parameter is relative to the outlier rejection (see 3.3.2). This parameter is called `robust_kernel`. After extensive testing we found a very good configuration of ROAMFREE for the simulation case.

In the tests we made we have adopted the evaluation procedure explained above, in Figure 5.4 we reports the results obtained from the comparison between `/ekf_localization_odom`, `/ekf_localization_gps` and `/raroam_node`. Observing the figure it is clear the supremacy of the

ROAMFREE respect to robot_localization. In addition the final result is the one expected, because we have `odom_r` that is the nearest frame to `map`, then the second nearest is `odom_g` and the third is `odom_f`. This shows the importance of the GPS that helps especially during the emergency cases.

5.4 Mapping

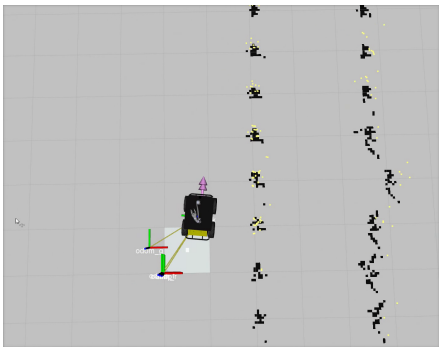
Once we ascertained the best method for the odometry estimation, we started the analysis to find the best way to map the environment (the one in Figure 5.1). To this aim, we tried in the architecture composed by the Gazebo nodes and odometry estimation nodes, different SLAM methods.

The simulated environment, even if it is as real as possible, represents a very optimistic scenario respect to the real one. In fact, mapping it correctly resulted to be very easy, so much that we used a primitive version of the architecture in which there were robot_localization in the variant without GPS. We used this architecture since this test has been one of the first we have done and thus at that time we did not have developed the ROAMFREE node or the robot_localization with GPS.

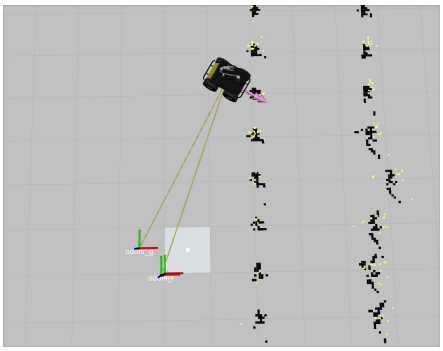
However the results we obtained excellent. We show the map built with the mentioned architecture in Figure 5.5.

5.5 Autonomous navigation

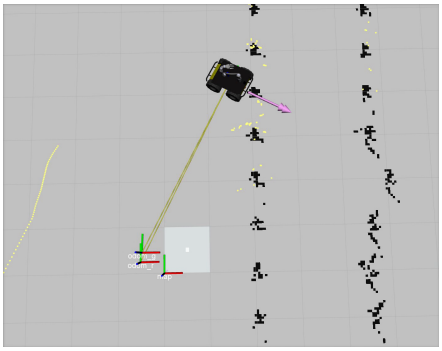
The last task we tested in simulation is the autonomous navigation. As for the mapping task in simulation, also here we have not encountered much problems. In fact, we made the robot autonomously navigate in the simulated vineyard, only by introducing the standard `amcl` and `move_base` to the architecture described in the previous section. We do not report the results obtained with this simple structure because even if they are quite good, we get better ones by trying the final architecture.



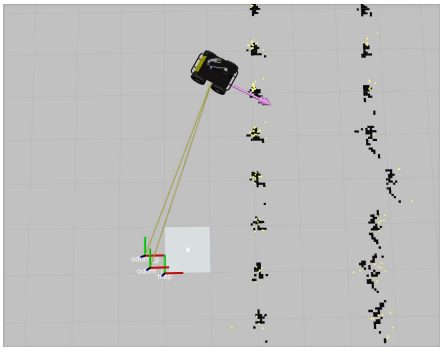
Initial state



start of emergency

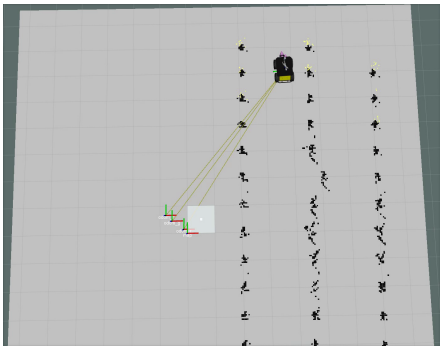


mid of emergency, about 10sec after the start



End of the emergency

Before turning



End state, after the turn and another emergency case

Figura 5.4: Odometry estimator comparison. `odom_r` *is the frame relative to* `/raroam_node`*,* `odom_g` *is relative to* `/ekf_localization_gps` *and* `odom_f` *is relative to* `/ekf_localization_odom`

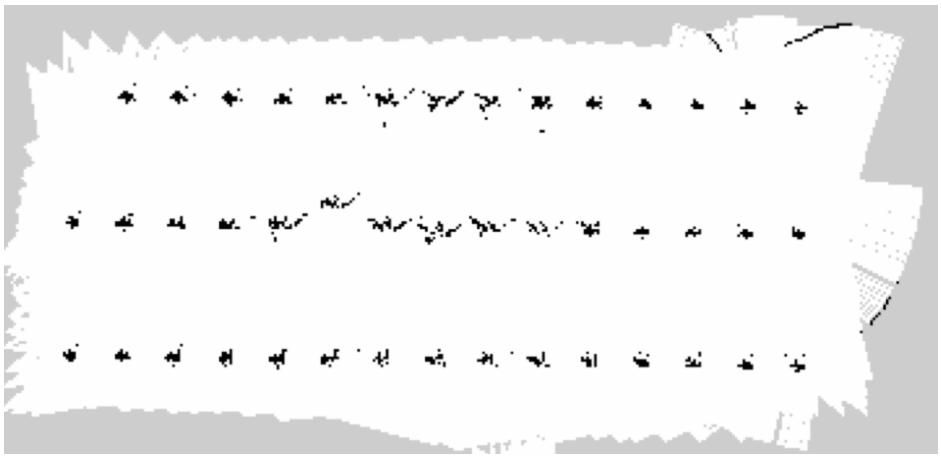


Figura 5.5: Map of the simulated vineyard, made with gmapping and ekf (wheels+imu) odometry estimatot

6.
Field test validation

In this chapter we explain how we tested our navigation system with the real environment data and the results we obtained.

6.0.1 Field test methods

In general two ways to test a ROS architecture with real data exist, the first is to go directly on the field and run the robot there, the second is to go only one time on the field acquire and save the data about the environment and then use it offline. This latter can be done thanks to the `rosbag` package with which we can save in files called bags the messages published by the topics we are interested to. The two ways have different pro and cons. In particular referring to our case, testing the system directly on the field has the advantages that we can try different hardware version of the robot (i.e., different sensors dispositions) and we can put the robot autonomous navigation in practice, on the other hand transporting the robot and all the technical stuffs to the field is usually very expensive and uncomfortable.

In our case the testing location and the GRAPE robot are situated in a different country (Spain), thus going on the field means to organize an expedition; moreover since the testing location corresponds to a vineyard it is not easy to work with PCs and other devices which require electric power. On the other hand, making tests using the bagged data allows to repeat the test as much as desired and so it allows to try different system structures or configuration without an extra effort. The main limitation of offline testing is that the data is always the same, thus there exists the risk to find a system configuration which overfit the data (it works good with the bag data and

bad with new data), and further there is not the possibility to change some hardware configuration if needed.

In our case we used both the ways, in fact we first exploited a bag to develop our final system, adding a module at the time and testing its impact with the bag data, until we reached the final solution we describe in chapter 4, then we have organized an expedition and went to the vineyard with the GRAPE robot where we validated (with additional parameters tuning) our navigation system.

In the bag we used during the first tests the GRAPE robot runs continuously along four vine lines. In particular it goes back and forth along two consecutive lines, then at the end of the second line it bypass one line and goes in the next line, from here it runs again a back and forth path between two consecutive lines. The GRAPE robot used to acquire this bag does not correspond to the current prototype, specifically in the hardware module of the ROS architecture there are some differences. In fact, in the bag we do not have the `/husky_wheels_encoder`, but we only have the `/husky_velocity_controller/odom`, and there is not any topic publishing the magnetometer readings.

This bag has been acquired during the winter (February 2017) when the vegetation is bare, that is also the period in which the robot is expected to work by the GRAPE project requirement. Afterwards we have done the vineyard validation tests during the summer (July 2017). Thus the testing environment in the two periods was quite different.

Since in real scenario is almost impossible to have a ground truth of the robot position, we did not repeated the evaluation procedure (5.3.1) adopted in simulation. Instead we decided to compare the two sensor fusion odometry estimators using the mapping tests, i.e., by comparing the accuracy of the maps generated with the same SLAM method. We show the test details and results in section 6.2.

6.1 Vineyard environment

The vineyard we used for tests has been granted by a Spanish winery called MasLlunes. It is situated in Catalunya, north-east of Spain. We show the satellite photo of the vineyard in Figure 6.1a. In this picture it can be observed that the MasLlunes vineyard respects the standard vineyard structure, in fact

(a) (b)

Figura 6.1: The MasLlunes vineyard. (a) is a satellite view, (b) shows the vine lines and the ground

it is composed by many long straight parallel lines of vines with a constant inter-line distance (2,50 meters). Each line is long about 100 meters.

The terrain of the vineyard is very irregular, due to the repeated passage of agricultural vehicles. The irregularity is even more accentuated in the summer period when the ground is more friable and there are many weeds that rapidly grow in it. Then, a similar situation happens also for the vine plants, that during the winter are completely bare and do not have side branches, while in the summer are densely populated by leaves and the branches are grown in all directions. The summer vineyard situation is shown in Figure 6.1b.

6.2 Mapping

In the mapping tests we verified both the different SLAM methods, and the two sensor fusion odometry estimators. We start analyzing the comparison tests between robot_localization and ROAMFREE. As anticipated before, this comparison has been evaluated relying on the mapping results obtained with the same SLAM method. We mainly performed such tests using the data of the winter bag.

The first thing we noted using real data was the huge difference in terms of perceived noise between the simulation environment and the real environment. In addition, in simulation we forced the presence of noise by generating

it with Gaussian distributions, while the real data does not follow any known distribution, turning out to be more unpredictable.

The presence of such noise led, as expected, to a decrement of the performance of our system. We partially (to obtain the same performance of simulation is impossible) solved this performance drop by deeply tuning and optimizing the configuration of our estimators. Nevertheless, for robot_localization estimator the presence of a lot of noise in the sensors readings does not affects much its configuration since it has only a small bunch of parameters that really influences the result of the estimation. For this reason the configuration we adopted for robot_localization on real data is very similar to the one we used for simulation tests. This feature can be seen as a prove of the generality of the estimator and thus an advantage. In practice it is not, conversely it is an important limit for the estimator that is not much adaptable to the increasing complexity of the environment.

The high noise strongly affected the ROAMFREE implementation and configuration. Indeed, spent much effort in setting the covariance of each of the sensors we adopted. We have seen in section 3.3.2 and in section 5.3.2 that the covariance is fundamental for the state estimation. Thus, analyzing the noise of each sensor, we found, after many attempts a good configuration of RF. We made this analysis thanks to a Matlab viewer available in the RF repository, which is able to visualize the trajectory estimated by RF, the trajectory drawn by the GPS readings (if GPS sensor is added to the factor graph) ,and, for the enabled sensors (sensors present in the factor graph), it shows the acquisitions and the relative estimated error. We report in Figure 6.2 the snapshot of the Matlab viewer after the complete execution of the bag using the final configuration of RF.

Observing the figure it can be noted how much the RF estimation is good, in fact the trajectory drawn coincides almost perfectly with the one actually done by the robot. To confirm this we can also look at the similarity with the path drawn by the GPS, that in this bag is RTK with fixed solution (accuracy range 10-30cm) and observe that it almost coincides with RF estimated trajectory. During the testing phase, we mainly worked with the `gmapping` package, since it is the most adopted laser SLAM method in robotics, although in indoor scenarios. This package, explained in section 4.6, contains many configuration parameters. Most of the parameters regards the particle filter, and among them we identified some that are really significant for our experiments. We now list these parameters:

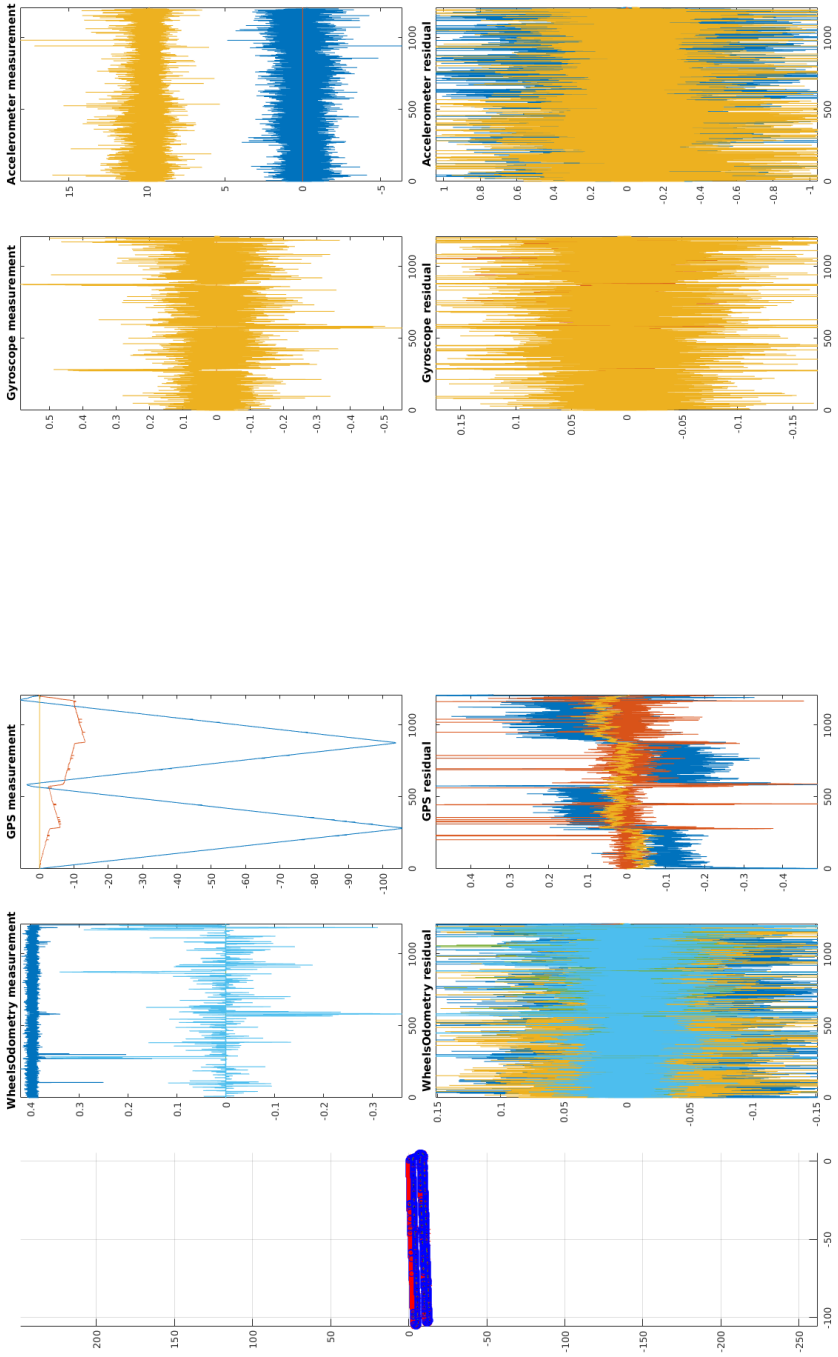


Figura 6.2: Matlab viewer snapshot

- `minimumScore` (float, default: 0.0) = 600: Minimum score for considering the outcome of the scan matching good. If enough high it can

avoid jumping pose estimates in large open spaces when using laser scanners with limited range (e.g. 5m).

- `srr` (float, default: 0.1) = 0.01: Odometry error in translation as a function of translation (rho/rho).
- `srt` (float, default: 0.2) = 0.01: Odometry error in translation as a function of rotation (rho/theta).
- `str` (float, default: 0.1) = 0.01: Odometry error in rotation as a function of translation (theta/rho).
- `stt` (float, default: 0.1) = 0.03: Odometry error in rotation as a function of rotation (theta/theta).
- `delta` (float, default: 0.05) = 0.03: Resolution of the map.

This configuration of `gmapping` relies much more on the odometry, in fact the four parameters `srr`, `srt`, `str` and `stt` specify the entity of the odometry estimation error. Decreasing them means to give more relevance to the estimated odometry, that we know to be good thanks to RF.

For the `slam_karto` package we adopted the default configuration, since all the other configurations we tried, produced worse maps, the `cartographer` configuration, presents many parameters which allow to use it for odometry estimation and SLAM simultaneously. We set these parameters in order to have a standard SLAM method which uses the odometry frame and the laser measurements. In addition we enabled the use of the online correlative scan matcher since it produce clearer maps. Analyzing the maps previously reported some considerations can be derived.

First of all, the superiority of RF with respect to robot_localization is evident, since all the maps built using it are clearly better than the ones obtained using the EKF filter.

The comparison between the SLAM approaches is quite more controversial. As a general remark we consider Gmapping to be the best algorithm in our scenario, since it is the most stable, though in the created map obstacles are not perfectly defined and sometimes a slightly curved map is generated. Indeed, Cartographer creates perfect maps in terms of accuracy and obstacles, but it is less reliable than Gmapping as sometimes it completely fails the generation of the map. This behavior is caused by the fact that Cartographer is a graph-based SLAM algorithm forcing scan-matching, when there

(a) (b) (c)

Figura 6.3: Maps of the MasLlunes vineyard built using robot_localization. (a) KartoSLAM (Feb), completely wrong, (b) Google's Cartographer (Feb), the map presents some erroneous lines in the lower part, (c) Gmapping (Feb), acceptable map, even if slightly curve, but this usually does not interfere with the localization algorithm

(a) (b) (c)

Figura 6.4: Maps of the MasLlunes vineyard built using ROAMFREE. (a) KartoSLAM (Feb), there are some inaccuracies and it is not acceptable without manual cleaning, (b) Google's Cartographer (Feb), it is a perfect map, (c) Gmapping (Feb), it is very good and straight

is not enough matching between two adjacent sub-maps the error created is significant (as it can be seen in the map obtained with robot_localization). This problem affects, even in a heavier way, the KartoSLAM algorithm being a graph based algorithm like Cartographer. This problem is evident in the robot_localization map. Maps produced by the Gmapping algorithm are not as accurate as the one produced by the other algorithms in terms of details, but they are more consistent. In terms of computational complexity Cartographer (executed with online correlative scan matching) is much more heavy than the other ones. As a consequence, if executed on a standard PC it needs to run the data bag with a reduced rate.

Comparing the maps created using February and July data, we can conclude that the main differences come from the different vegetative state of the vineyard. In fact, in February there are no lives on the vines and only small weeds on the terrain, while in July the vines are full of leaves and the grass on the terrain is more developed. In addition the soil cohesion changes, causing robot wheels skid in a different way. Nevertheless we have been able to generate maps in both conditions as it was aimed by the project.

6.3 Autonomous navigation

During the real vineyard tests the complete navigation architecture was installed and successfully tested on the robot platform. Autonomous navigation tests at MasLlunes were carried on for one day and a half during which we collected experimental data in order to continue the tuning and testing of the navigation algorithm in the lab. During this period we also made tests related to the other work-packages of the projects, including field monitoring and manipulation.

These preliminary tests on the field, indeed this was the outcome of the first integration of the hardware and software, have shown very promising results regarding mapping and localization. However they have also shown some issues related to the planner which were not identified in the simulations.

Indeed, the move_base global planner needs a very high inflation of the obstacles in order to avoid planning paths that go through vine rows. This high inflation, on the other hand, generates issues with obstacles which are very small in the real scenario, but become bigger in the costmap such as weeds and long grass. In the February tests this was no evident since the weeds and grass were cut and the the vineyard was not alive, during July

field test this issue has raised. In the next months we plan to face this issue by trying different solutions, among these a possible identification of rows in the vineyard to be considered when planning so not to require the excessive costmap obstacle inflation. A second problem identified for the planner is due to the fact it it has been developed for a differential drive robot while, as we already mentioned, our robot has a skid steering kinematics. As a consequence, the planner has a recovery procedure, in case the robot is lost (the robot is not sure about its localization in the map), in which it makes the robot turning on itself, but the skidding kinematics does not allow for a perfect turn around the robot center and, coupled with the shape of the wheels (knobby wheels), it causes a wrong motion of the platform. In this case a different recovery procedure has to be implemented.

As a general comment, we can conclude that we are very satisfied about the navigation stack and its tuning for the GRAPE scenario, during the tests the localization worked properly, as the robot only a few times got lost in the almost two days of trials. For the odometry estimator and the SLAM node still some improvements are possible, for instance by adding the magnetometer measurement and optimizing the fusion parameters, but most of the effort will be devoted during the integration phase is solving the planner problems previously mentioned above To provide the reader an example of the planner issues and how they have been faced during the field tests we summarized here some crucial moments of the video using some screenshots:

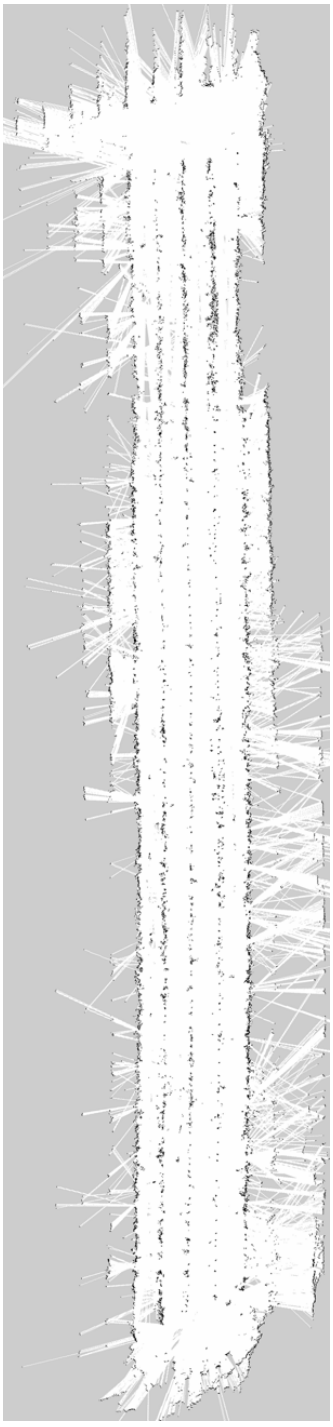
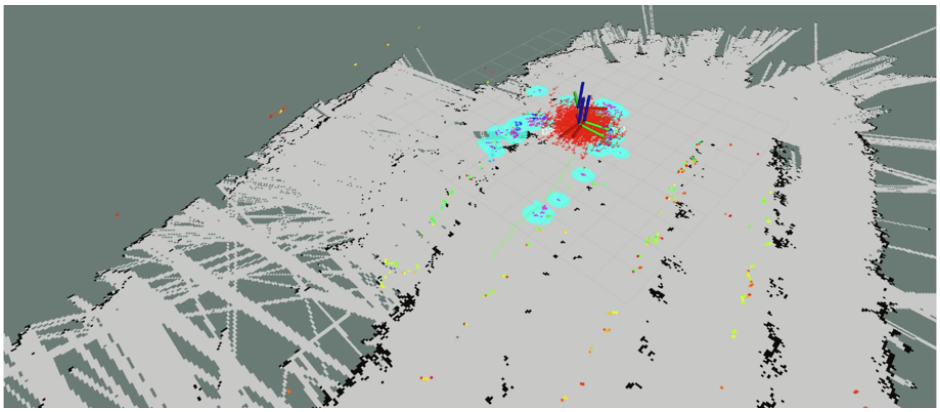
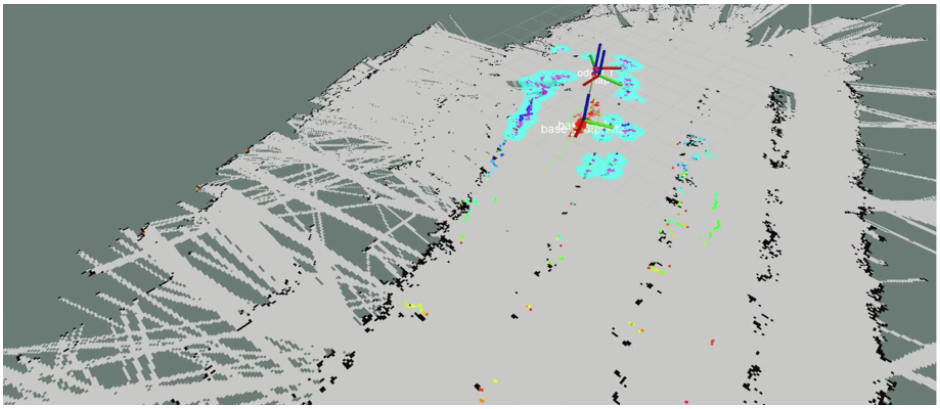


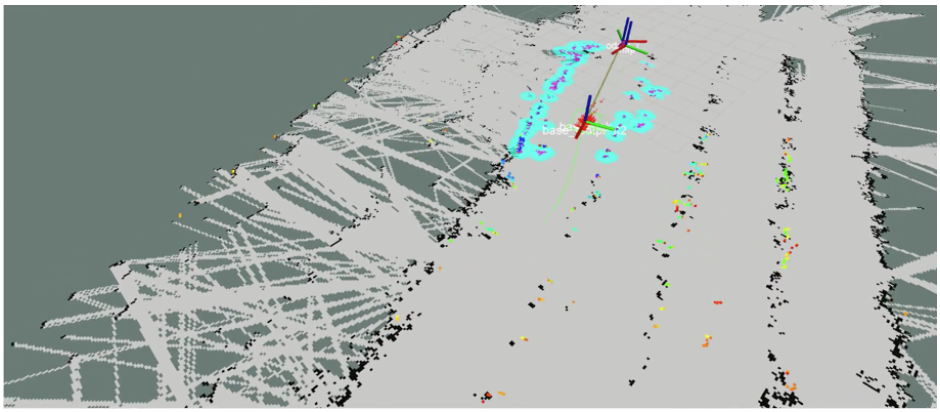
Figura 6.5: Map of the MasLlunes vineyard in July, built using ROAMFREE and Gmapping



(a)

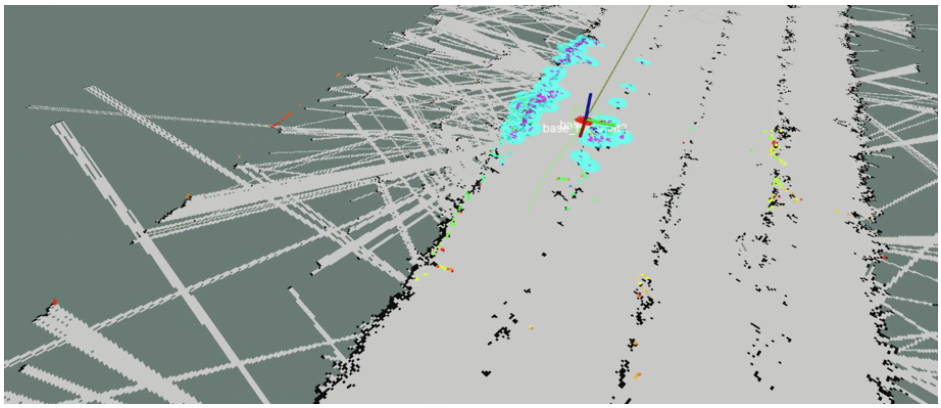


(b)

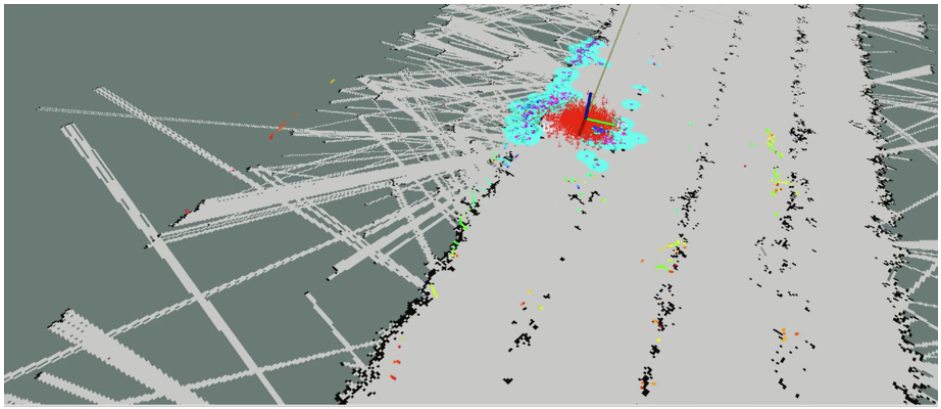


(c)

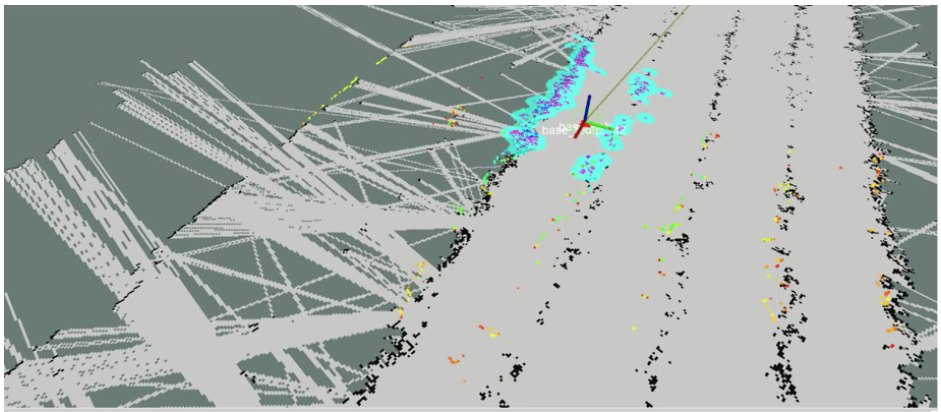
Figura 6.6: Maps of the MasLlunes vineyard built using robot_localization. (a) KartoSLAM (Feb), completely wrong, (b) Google's Cartographer (Feb), the map presents some erroneous lines in the lower part, (c) Gmapping (Feb), acceptable map, even if slightly curve



(a)



(b)



(c)

Figura 6.7: Maps of the MasLlunes vineyard built using robot_localization. (a) KartoSLAM (Feb), completely wrong, (b) Google's Cartographer (Feb), the map presents some erroneous lines in the lower part, (c) Gmapping (Feb), acceptable map, even if slightly curve

7.
Conclusions and future work

This thesis was aimed at building a reliable navigation system for a mobile robot which needs to autonomously navigate in vineyards. We can identify three main challenges faced during the development of such system and how we accomplished these:

- *The variability of the vineyard environment* influences significantly the perception of the robot, since the presence of a rough terrain, the presence of an active vegetation, and the presence of variable weather conditions generate noise in sensor data. A system using only one sensor is not reliable for this case, we need to exploits different type of sensor giving a different view of the world for the filtering of the noise and we fused them.

- *The integration of multiple sensors* requires a complex method able to merge the sensors acquisitions and use them to accurately estimate the robot pose. Our navigation system adopt a factor graph model to accomplish this task, ROAMFREE as method for sensor fusion and odometry estimation.

- *The adaptation of indoor methods to outdoor scopes* has been our last challenge our system try to readapt localization method such as AMCL and SLAM methods such as Gmapping, Cartographer and KartoSLAM that are designed mainly for indoor robot, but with an appropriate odometry estimation and a proper TF tree they can obtain good results also in different scenario as the vineyard is.

The resulting navigation system has proved his reliability first in simulation and then when it has been installed on a real robot and used in a real vineyard. In particular validation has highlighted some strengths:

- An accurate and robust pose estimate
- The ability to build navigable map even in complex/harsh/rough environments
- A flexible structure which components can easily be exchanged/swapped/switched to complete different tasks

It has also shown some limits:

- The difficulty in autonomous navigation due to unexpected obstacles
- Problems in the correct path planning both for global and for local costmap
- Wrong localization recovery behavior

In conclusion we can assert that we are satisfied by the obtained results and that the thesis has reached the prefixed goal. However, we listed some defects that should be fixed in the next future.

Some ideas for the future are

- Implement a planner specific for skid-steering robots.
- Integrates a vision component in the system able to recognize dangerousness of the obstacle individuated by frontal sensor.
- Building of an elevation mapping of the terrain in order the relieve holes etc in case therer area

Bibliografia

[1] C. Thorpe and H. Durrant-whyte, "Field robots," in *International Journal of Pattern Recognition and Artificial Intelligence*, 2001, pp. 39–7.

[2] A. Singhal, "Issues in autonomous mobile robot navigation," Tech. Rep., University of Rochester, 1997.

[3] W. Elmenreich, "An introduction to sensor fusion," Research Report 47/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001.

[4] M. Kam, X. Zhu, and P. Kalata, "Sensor fusion for mobile robot navigation," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 108–119, Jan. 1997.

[5] C. A. Fowler, "Comments on the cost and performance of military systems," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-15, no. 1, pp. 2–10, Jan. 1979.

[6] R. Guzman, J. Ariño, R. Navarro, C. Lopes, J. Graça, M. Reyes, A. Barriguinha, and R. Braga, "Autonomous hybrid gps/reactive navigation of an unmanned ground vehicle for precision viticulture -vinbot," 11 2016.

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

[8] H. Shu, S. Zhang, B. Shen, and Y. Liu, "Suboptimal state estimation for continuous time nonlinear systems from discrete noisy measurements," *International Journal of General Systems*, vol. 45, pp. 648–661, July 2016.

[9] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, Oct 1997.

[10] D. Koller, N. Friedman, L. Getoor, and B. Taskar, "Graphical models in a nutshell," in *An Introduction to Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. MIT Press, 2007.

[11] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Real-time monocular slam: Why filter?," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2657–2664.

[12] T. A. Davis., *Direct methods for sparse linear systems,*, 2006.

[13] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment - a modern synthesis," in *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, London, UK, UK, 2000, ICCV '99, pp. 298–372, Springer-Verlag.

[14] J. B. Moore, "Discrete-time fixed-lag smoothing algorithms," *Automatica*, vol. 9, no. 2, pp. 163–173, Mar. 1973.

[15] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theor.*, vol. 47, no. 2, pp. 498–519, Sept. 2006.

[16] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 4290–4297.

[17] "Turtlebot," https://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot, June 2017.

[18] R. M. Harlan, D. B. Levine, and S. McClarigan, "The khepera robot and the krobot class: A platform for introducing robotics in the undergraduate curriculum," *SIGCSE Bull.*, vol. 33, no. 1, pp. 105–109, Feb. 2001.

[19] M. A. E. Lomba, "On the hardware and software architecture of the robuter mobile platform: a hands-on approach," 2015.

[20] L. Caracciolo, A. de Luca, and S. Iannitti, "Trajectory tracking control of a four-wheel differentially driven mobile robot," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 1999, vol. 4, pp. 2632–2638 vol.4.

[21] J. Yi, D. Song, J. Zhang, and Z. Goodwin, "Adaptive trajectory tracking control of skid-steered mobile robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, Apr. 2007, pp. 2605–2610.

[22] J. Yi, J. Zhang, D. Song, and S. Jayasuriya, "Imu-based localization and slip estimation for skid-steered mobile robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2007, pp. 2845–2850.

[23] P. D. Kozlowski, Krzysztof, "Modeling and control of a 4-wheel skid-steering mobile robot," *International Journal of Applied Mathematics and Computer Science*, vol. 14, no. 4, pp. 477–496, 2004.

[24] J. L. Martínez, A. Mandow, J. Morales, S. Pedraza, and A. García-Cerezo, "Approximating kinematics for tracked mobile robots," *The International Journal of Robotics Research*, vol. 24, no. 10, pp. 867–878, 2005.

[25] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, and Q. Zhao, "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor," vol. 15, pp. 9681–9702, 05 2015.

[26] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, and Q. Zhao, "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor," *Sensors (Basel, Switzerland)*, vol. 15, no. 5, pp. 9681–9702, 05 2015.

[27] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez, "Experimental kinematics for wheeled skid-steer mobile robots," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2007, pp. 1222–1227.

[28] T. Moore and D. Stouch, *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*, pp. 335–348, Springer International Publishing, Cham, 2016.

[29] S. F. S. G. L. Mc Gee, L. A.; Schmidt, *Application Of Statistical Filter Theory To The Optimal Estimation Of Position And Velocity On Board A Circumlunar Vehicle*, 1962.

[30] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[31] G. Welch and G. Bishop, "An introduction to the kalman filter," Tech. Rep., Chapel Hill, NC, USA, 1995.

[32] C. Riggelsen, *MCMC Learning of Bayesian Network Models by Markov Blanket Decomposition*, pp. 329–340, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[33] L. M. Paz, P. Jensfelt, J. D. Tardos, and J. Neira, "Ekf slam updates in o(n) with divide and conquer slam," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 1657–1663.

[34] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[35] A. Karlsson, J. Bjärkefur, J. Rydell, and C. Grönwall, *Smoothing-Based Submap Merging in Large Area SLAM*, pp. 134–145, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[36] K. Sharma, I. Moon, and S. G. Kim, "Extraction of visual landmarks using improved feature matching technique for stereo vision applications," *IETE Technical Review*, vol. 29, no. 6, pp. 473–481, 2012.

[37] W. Maddern, M. Milford, and G. Wyeth, "Towards persistent localization and mapping with a continuous appearance-based topology," 09 2017.

[38] H. Zhang, Y. Liu, and J. Tan, "Loop closing detection in rgb-d slam combining appearance and geometric constraints," *Sensors (Basel, Switzerland)*, vol. 15, no. 6, pp. 14639–14660, 06 2015.

[39] J. S. Lee, C. Kim, and W. K. Chung, "Robust rbpf-slam using sonar sensors in non-static environments," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 250–256.

[40] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb 2007.

[41] S. Thrun and M. Montemerlo, "The GraphSLAM algorithm with applications to large-scale mapping of urban structures," *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005.

[42] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *Int. J. Rob. Res.*, vol. 25, no. 12, pp. 1181–1203, Dec. 2006.

[43] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding window filter with application to planetary landing," *J. Field Robot.*, vol. 27, no. 5, pp. 587–608, Sept. 2010.

[44] I. J. Cox and G. T. Wilfong, Eds., *Autonomous Robot Vehicles*, Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[45] S. P. Engelson and D. V. McDermott, "Error correction in mobile robot map learning," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, May 1992, pp. 2555–2560 vol.3.

[46] S. K. N. Johnson and N. Balakrishnan., *Continuous univariate distributions*, 1995.

[47] J. A. Rice, *Mathematical Statistics and Data Analysis*, 1988.

[48] "amcl documentation," http://wiki.ros.org/amcl, May 2017.

[49] "Huksy robot," https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/, Feb. 2017.

[50] "Jaco2 arm," http://www.kinovarobotics.com.

[51] R. Vincent, B. Limketkai, and M. Eriksen, "Comparison of indoor robot localization techniques in the absence of gps," 04 2010.

[52] "robot_localization documentation," http://docs.ros.org/kinetic/api/robot_localization/ Feb. 2017.

A.
Introduction to ROS

Robot Operating System (ROS) is used in this thesis as the robotic middleware. It is used to develop and test implementations of the mapping system discussed in Chapter 4. According to the ROS website1, "ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms". ROS is a completely open-source, community driven project that is used by many universities, companies and hobbyists worldwide. ROS needs to run on top of an existing Operating System (OS), and is primarily targeted at Ubuntu. The ROS system is very modular; the minimum set of packages is formed by the ROS base installation. It can be extended by any ROS packages as desired. A common set of packages is captured in the ROS full desktop install, which contains of the base and a set of commonly used packages such as packages for simulation and visualization. In addition to the official Ubuntu releases, experimental releases exist for many other systems including OS X, Windows, Raspberry Pi, and embedded systems through OpenEmbedded. ROS can be run in a distributed way over multiple computers, communicating over TCP. Official support is also provided for several robots such as the PR2 and Turtlebot, which are controlled by a computer running Ubuntu. ROS provides hardware abstraction, low-level device control, implementations of commonly used functionality, communication between processes, and package management. Packages provide functionality such as controlling movement of the robot, generating odometry infor- mation, reading and processing sensor data from e.g. a Kinect, camera or laser range sensor, keeping track of a robots joint configuration, etc. Packages can also provide more high level functionality such as SLAM implementations, object recognition, 3D simulation,

compatibil- ity layers to enable the use projects like Point Cloud Library (PCL) and Open Computer Vision (OpenCV), etc. The basis for ROS was laid at Stanford University, where the robotic middleware Switchyard was developed in 2007. Switchyard was further developed under the name ROS by the com- pany Willow Garage (a robotics research institute/incubator). ROS saw is first official release, ROS 1.0, in January 2010. Since then, many releases followed. For this thesis, the 10th official release named Kinetic Kame is used, released in May 2016. In the last years, ROS has quickly grown in popularity and an increasing number of projects, companies, and universities seem to be using it. For example, the ambitious STRANDS project, which is in many ways a successor to the CogX project, is using it. Also, the ROCS toolkit by Pronobis which forms the basis of the work presented in [Pronobis and Jensfelt, 2012], is expected to be released soon for ROS5. ROS can use the Player/Stage, Gazebo and MORSE projects for (multi)robot simulation. Gazebo is capable of simulating full 3D worlds including physics and usually only a few adaptations are needed to run a real world ROS system in simulation.

A.1 ROS concepts

A good overview of the ROS concepts can be found on the ROS website6. ROS divides these concepts in three different levels: the ROS filesystem level, the ROS computation graph level, and the ROS community level.

A.1.1 ROS filesystem

ROS uses various concepts at your local filesystem level, including:

- Packages: At the level of the computers local filesystem, ROS organizes functionality in packages. Packages contain files that together give the package functionality, such as exe- cutables (called ROS nodes), source code, cmake files, ROS message type definitions, ROS service type definitions, roslaunch files, configuration files, etc. Packages contain a manifest file (package.xml) that provide all the metadata about the package.

- Metapackages Metapackages are empty packages that only have dependencies on a set of other packages. Metapackages are used to collect a

set of packages that together provide some functionality. By installing a metapackage, all relevant packages will get installed, similar to how this is sometimes done by apt-get.

- Workspaces and overlaying ROS installs to your /opt folder. The Indigo release for example installs to /opt/ros/indigo. The ROS packages that are installed under this folder should not be edited by the end user. Instead, the end user should create a so called workspace, usually in your home folder. This workspace overlays the packages in /opt/ros: you can create your own packages here, add packages (manually or through a VCS) from other people, or check out original ROS packages from Github for example to make your own changes to it. An overlay, such as your workspace, has a higher precedence than everything that it overlays (e.g. /opt/ros/indigo). That is, you can install the gmapping package through apt-get (which will end under opt), and later add the Github version to your own workspace and make some changes to the code. ROS will now ignore the gmapping package under opt, as it gives priority to the version in your own workspace. You can even chain multiple workspaces using overlaying. Additionally, you can install multiple ROS releases at the same time. Switching releases or workspaces is as easy as sourcing a bash script (which you usually make part of your .bashrc file). Although multiple workspaces can be active at the same time, only one ROS release can be used at the same time.

- Launch files An important kind of file are .launch files, which are executed by roslaunch. These XML based files specify ROS nodes (see next Section) that should be launched, to- gether with parameters. The files can also include (import) other launch files, and can control many advanced settings such as remapping topics, interpreting passed arguments (through command line or launch file inclusions), etc. Launch files form a convenient way to start ROS functionality and are found in many packages to ease the use of the packages. By inclusion of other launch files, a launch file can be a very powerful tool that allows to start up a full, complex system.

A.1.2 ROS computation graph

The ROS computation graph is the peer-to-peer network of processes that together process all data. Figure (ref) shows the structure of such a graph. Master The ROS master is the process that governs the network and provides name regis- tration and lookup to the rest of the processes. Nodes Nodes7 are the processes of the graph. The gmapping package for example contains a gmapping node (executable). Packages can contain multiple nodes. Nodes communicate with other nodes through messages and services, and can be configured using parameters. Nodes are generally programmed in C++ or Python. LISP is also official supported, while experimental support for Java and several others is available as well. Parameters ROS works with global and private parameters. Private parameters are passed to nodes when they are launched and are only accessible to that node. Global parameters are stored in the parameter server and are accessible by all nodes at any time. Global parameter names are of the format /<parent1>/<parent2>/<name>, e.g. /gmapping/map_update_interval. A name can consist of zero or more parents. Messages Messages are used for communication between nodes. Messages are simple data structures (similar to C structs), existing of typed fields. Messages can be exchanged through topics or services. Topics Nodes can publish messages to topics. Other nodes can subscribe to such topics. A topic will always contain messages of only one type. Topics can receive messages from multiple publishing nodes and can have multiple subscribing nodes. Topics are named in the same way as global parameters. An example is a SLAM node that publishes an occupancy grid map as a message to the /map topic. A navigation node can subscribe to /map to use this map for planning a path. Services Services work on a request / reply basis, whereas topics work on a publish / subcribe basis. A node (called the client) can send a request as a message to a service. The node that provides this service then responds with a reply message. Nodes that provide a service provide such a service under a name, which is named like a global parameter. Services always have only one node offering the service, and it can only serve one client at a time. Other tools ROS can record message data and play it back later using the command line tool rosbag. This is especially useful for collecting real-world sensor measurements and playing that data back again later for development and testing. Another important tool is the command line tool rqt, which offers a set of various GUIs. A very useful one is the

rqtgraph, which generates a graphical representation of all the nodes and topics of the currently running system. Lastly, the rviz node (which is part of the rviz package) can be used to visualize a variety of data. One can add all sorts of windows to enable visualization of certain types of data. For example, one can add a window showing the map generated by SLAM, a window that shows the robot, a window that shows laser scan data (reading sensor_msgs/LaserScan messages on the /scan topic), a window showing a planned path for navigation, etc. All windows are combined in one 2D or 3D view.

A.1.3 ROS community

ROS is strongly driven by its community. The wiki provides a lot of documentation, including installation instructions, beginner tutorials and documentation for packages. ROS Answers (answers.ros.org) is a Q&A site for asking ROS related questions and can prove very use- ful from time to time. Furthermore, ROS provides all kinds of tools and documentation to stimulate contributions by the community. There are tools to ease the creation/generation of documentation, tools to generate package wiki pages, to automatically test and update package builds, there is a bug ticket system, there is a ROS Enhancements Proposals (REPs) system, there are tools and guidelines to use Github (the officially preferred VCS), etc.

B.
Sensors specifications

B.1 Xsens MTI-10 IMU

Inertial measurement units (IMU) will be used to obtain orientation and acceleration of the system. The selected model is a MTI-10 series, from XSense.
The following table shows the sensors specifications of the IMU:

	Gyroscopes		**Accelerometers**	
	Typ	**Max**	**Typ**	**Max**
Standard full range	+/- 450° /s	-	50 m/s2	-
Bias repeatability (1 yr)	0.2°/s	0.5°/s	0.03 m/s2	0.05m/s2
In-run bias stability	18°/h	-	40 μg	-
Bandwidth (-3 dB)	415 Hz	N/A	375 Hz	N/A
Noise density	$0.03 degree/s/\sqrt{Hz}$	$0.05°/s/\sqrt{Hz}$	80 °g/$\sqrt{Hz}$	150 $\mu g/\sqrt{Hz}$
Non-orthogonality	0.05 deg		0.05 deg	
Non-linearity	0.03% FS	0.1% FS	0.03% FS	0.5% FS
Magnetometer				
	Typ	**Max**		
Standard full range	-	+/- 80 μT		
Noise density	200 $\mu G/\sqrt{Hz}$	-		
Non-linearity	0.1% FS	-		

Tabella B.1

B.2 IMU sensors specification

The following table shows the systems specifications of the IMU:

Input voltage	4.5-34V or 3V3
Typical power consumption	480-570 mW
IP-rating	IP 67 (encased)
Temperature	-40 to 85 °C
Vibration and shock	MIL STD-202 tested; 2000 g for 0.5 ms
Sampling frequency	10 kHz/channel (60 kS/s)
Output frequency	Up to 2 kHz
Latency	$<$ 2 ms
Interfaces	RS232/RS485/422/UART/USB (no converters)
Standard full range gyro	450 °/s (1000 °/s available as an option)
Standard full range acc	50 m/s^2 (150 m/s^2 available as an option)
In-run bias stability gyro	18°/h
Bandwidth gyro	415 Hz
Bandwidth acc	375 Hz

Tabella B.2

B.3 Hokuyo sensors specification

The selected model is a Hokuyo UTM-30LX-EW from Hokuyo. This laser is a small, accurate, high-speed device for outdoor robotic applications. The next table summarizes the technical data of the laser:

Product Name	**Scanning Laser Range Finder**
Model	UTM-30LX-EW
Light Source	Laser Semiconductor λ = 905nm Laser Class 1
Supply Voltage	12VDC ± 10%
Supply Current	Max: 1A, Normal : 0.7A
Power Consumption	Less than 8W
Detection Range and Detection Object	Guaranteed Range: 0.1 ∼ 30m (White Kent Sheet)
	Maximum Range : 0.1 ∼ 60m Minimum detectable width at 10m : 130mm (Vary)
Accuracy	0.1 10m : ±30mm, 10 30m : ±50mm (White KentSheet) Under 3000lx : White Kent Sheet: ±30mm (0.1m to 10m) Under 100000lx : White Kent Sheet:±50mm (0.1m to 10m)
Measurement Resolution and Repeated Accuracy	1mm 0.1 10m : σ 10mm, 10 30m : σ 30mm (White Kent Sheet) Under 3000lx : σ = 10mm (WhiteKent Sheet up to 10m) Under 100000lx : σ = 30mm (White Kent Sheet up to 10m)
Scan Angle	270°
Angular Resolution	0.25° (360°/1440)
Scan Speed	25ms(Motor speed : 2400rpm)
Interface Ethernet	100BASE-TX(Auto-negotiation)
Output	Synchronous Output 1- Poin
LED Display	Green: Power supply.Red: Normal Operation (Continuous), Malfunction(Blink)
Ambient Condition(Temperature, Humidity)	-10°C ∼ +50°C Less than 85%RH (Without Dew, Frost)
Storage Temperature	-25∼75°C
Environmental Effect	Measured distance will be shorter than the actual distance under rain, snow and direct sunlight
Vibration Resistance	10 ∼ 55Hz Double amplitude 1.5mm in each X, Y, Zaxis for 2hrs. 55 ∼ 200Hz 98m/s2 sweep of 2min ineach X, Y, Z axis for 1hrs.
Impact Resistance	196m/s2 In each X, Y, Z axis 10 times
Protective Structure	Optics: IP67 (Except Ethernet connector)
Insulation Resistance	10MΣ DC500V Megger
Weight	210g (Without cable)
Case	Polycarbonate
External Dimension (WxDxH)	62mm$x$62mm$x$87.5mm

Tabella B.3

B.4 GPS Novatel OEMstar receiver

The selected model is a OEMstar receiver by Novatel. The OEMStar measures only 46 by 71 mm, accepts an input voltage between 3.1 and 5.25 VDC and consumes less than 500 mW. This makes the OEMStar an attractive choice for use in robotics applications.

General Info	Length (mm) 71.00 Width/Diameter (mm) 46.00 Height (mm) 13.00 Weight (g) 18.00 Typical Power Consumption (W) 0.36
Constellation	GPS + GLONASS SBAS capable
Tracking	Max Num of Frequency Single
Number of Com Ports	LVTTL 2 + USB Device 1
Performance	**Accuracy** **(RMS)** Single Point L1 1.5m SBAS 0.7m DGPS 0.5m

Tabella B.4

B.5 Stereo camera: Zed

Zed camera from Stereolabs is a stereo camera up to 2K resolution. This sensor may be used for navigation purposes being able to obtain a rough 3D reconstruction of the entire vineyard. The specifications of this sensor are:

Features	High-Resolution and High Frame-rate 3D Video Capture Depth Perception indoors and outdoors at up to 20m 6-DOF Positional Tracking Large-scale 3D Mapping using ZEDfu	
Video Mode	**Frames per second**	**Output Resolution**
2.2K	15	4416x1242
1080p	30	3840x1080
720p	60	2560x720
WVGA	100	1344x376
Depth	**Depth Resolution** Same as selected video resolution **Depth Range** 0.7 - 20 m (2.3 to 65 ft)	**Depth Format** 32-bits **Stereo Baseline** 120 mm (4.7")
Motion	**6-axis Pose Accuracy** Position: +/- 1mm Orientation: 0.1 ° **Frequency** Up to 100HZ	**Technology** Real-time depth-based visual odometry and SLAM
Lens	Wide-angle all-glass dual lens with reduced distortion Field of View: 110° (D) max. $f/2.0$ aperture	
Sensors	**Sensor Resolution** 4M pixels per sensor with large 2-micron pixels **Sensor Size** 1/3" backside illumination sensors with high low-light sensitivity **Camera Controls** Adjust Resolution, Frame-rate, Exposure, Brightness, Contrast, Saturation, Gamma, Sharpness and White Balance	**Sensor Format** Native 16:9 Format for a greater horizontal field of view **Shutter Sync** Electronic Synchronized Rolling Shutter **ISP Sync** Synchronized Auto Exposure
Connectivity	**Connector** USB 3.0 port with 1.5m integrated cable **Power** Power via USB 5V / 380mA	Mounting Options Mount the camera to the ZED mini tripod or use its 1/4"-20 UNC thread mount Operating Temperature 0°C to +45°C (32°F to 113°F)
Size and Weight	**Dimensions** 175 x 30 x 33 mm (6.89 x 1.18 x 1.3")	**Weight** 159 g (0.35 lb)

Tabella B.5

B.6 3D Lidar: Velodyne Puck Lite

In order to obtain a more precise 3D reconstruction of the vineyard, we equipped the robot with a 3D Lidar system from Velodyne. This sensor provides a 360o point cloud using a 16 rotating lasers.

Features	**VLP-16**
Channels	16
Range	100m
Accuracy	+/- 3cm
Data	Distance/Calibrated Reflectivities
Data rate	300,000 pts/sec
Vertical FOV / Resolution	30° / ∼2.0°
Horizontal FOV / Resolution	360° / 5Hz: 0.1°, 10Hz: 0.2°, 20Hz: 0.4°
Input Voltage	9-32 VDC
Power	8W
Environmental	IP67
Operating Temperature	-10° to 60° C
Size	104mm x 72mm
Weight	0.83kg

Tabella B.6

C.
TF library

The TF library was designed to provide a standard way to keep track of coordinate frames and transform data within the entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system. As robotic systems get more and more complicated, being able to focus on precisely the task frame and only the relevant coordinate frames becomes critical. Most robotic systems are fusing data from many different sensors with different coordinate frames Instead to explain only the `/tf` topic we decide to describe entire TF library, since it is very useful to know it is structure to better understand the three main problem that we face: odometry estimation, localization and SLAM. In general the scope of the TF is to provide the geometric relation between two different frames (see Figure 4.3). The TF library can be separated into two different parts. The first part is disseminating transform information to the entire system. It is called `broadcaster`. The second part of the library, the `listener`, receives the transform information and stores it for later use. The second part is then able to respond to queries about the resultant transform between different coordinate frames. There are often several different sources of information regarding the various coordinate frames in a system. Each of these sources of information is often connected to hardware and produce data(e.g sensor values, actuator feedback) at different frequencies, and could potentially be connected over a link with non-trivial latency or packet drops. As such the TF library must accept asynchronous inputs and be robust to delayed or lost information.

The TF library is designed to be a core library of the ROS ecosystem. To be able to support ROS applications it needed to be robust to distribu-

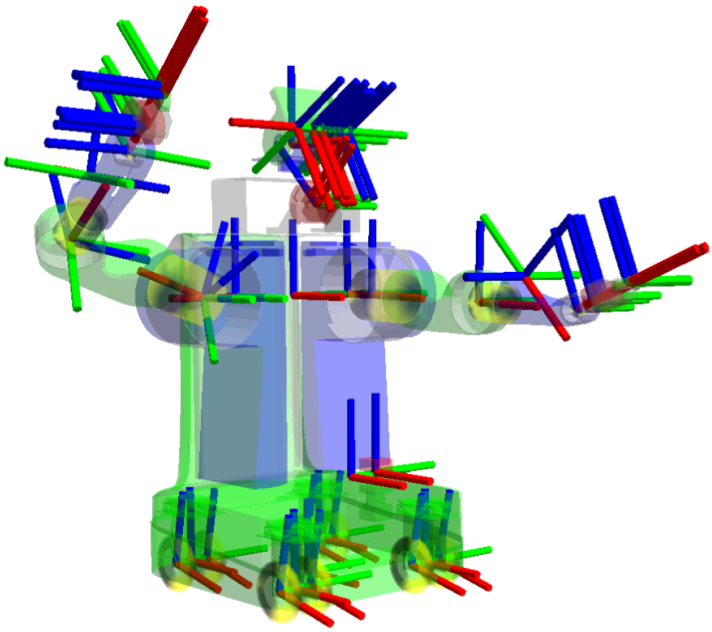


Figura C.1: A view of all the standard TF frames in Willow Garages PR2 Robot with the robot meshes rendered transparently and the edges of the tree hidden. The RGB cylinders represent the X, Y, and Z axes of the coordinate frames.

ted computing environments with unreliable networking and non negligible latency. The design is also influenced by the need to communicate using anonymous publish subscribe message passing.

The library needs to be able to provide a transform between two coordinate frames at a requested time. If data is not available the library provides the user with an appropriate error and not return invalid data. It does not assume that the system have a constant structure, so it also provides the ability to dynamically change the relationships between frames including adding, removing, and changing connections between coordinate frames. Transforms and coordinate frames can be expressed as a graph with the transforms as edges and the coordinate frames as nodes. The graph can exist with one or more disconnected subgraphs and the transform can be computed between nodes within the subgraphs, but not between disconnected subgraphs. Transforms are inherently directed. To traverse up an edge the inverse of the transform can be used. However, with an arbitrary graph, two nodes may have multiple paths between them, resulting in two or more potential

net transforms making the result of the query ambiguous. To avoid this the graph must be acyclic.

Then To provide quick look ups the tree must be quickly searchable. Limiting the graph to a tree enables fast searching for connectivity. This becomes important as graph complexity increases. TF tree is designed to be queried for specific values asynchronously. A tree structure also has the benefit of allowing for dynamic changes to the structure without using extra information except the directed graph edges. When an edge is published to a node referencing a different parent node, the tree will resolve to the new parent without extra information. Each update to the edge of the tree is specific to the time at which it was measured. Likewise, queries against the tree are required to have a specific time at which to make the look up. To make this possible, a history of the values of an edge of the graph is stored in a chronologically sorted list to enable quick look up. Data is stored for a specified duration and within that period it can be expected to be able to query for a net transform within the TF tree. To be able to operate, all data which is going to be transformed by the TF library must contain two pieces of information: the coordinate frame in which it is represented and the time at which it is valid.