POLITECNICO DI TORINO

Master of Science in Mechatronic Engineering

Master's Degree Thesis

GPS-based autonomous navigation of unmanned ground vehicles in precision agriculture applications

Supervisor

Prof. Marcello CHIABERGE

Candidate

Simone CERRATO

October 2020

Abstract

The global population is growing exponentially and the actual agricultural techniques and resources will not be able to feed every person on the Earth in a few years. To account for this serious problem, groups of research are focusing their attention on precision agriculture, because it looks for the improvement of the productivity and efficiency of both agricultural and farming production processes, while reducing the environmental impact, exploiting automation and robotics.
The thesis aims to design and develop a solution, based on GPS, for the autonomous navigation problem in precision agriculture, using only few sensors: an Inertial Measurement Unit, a GPS receiver and a depth camera, in order to be cost effective. The proposed goal has been achieved through a system of inter-operating sub-components, that have to share information and collaborate each other in order to provide a complete autonomous navigation. In particular, the main involved entities are: a localization filter, a global and a local path planning algorithms and an obstacle avoidance approach, that have been developed and can cooperate each other by means of the Robot Operating System.
Eventually, the proposed solution has been tested in a simulation environment, through different possible scenarios providing good results in each of them. However, it may be considered as a starting point for future improvement in the field of autonomous navigation for precision agriculture.

Acknowledgements

During the academic years I have had the chance to improve both my technical knowledge and communication skills as well as I have realized my dream to work on mechatronics systems. First of all, the achievement of such goals have been possible thank to the economic and moral supports of my parents. Second, I would like to thank also my girlfriend Beatrice for having supported me during the hard exam periods.
Moreover, I am grateful to professor Marcello Chiaberge to have given me the opportunity to develop my thesis at the Interdepartmental Centre for Service Robotics, Pic4Ser. Eventually, I would like to thank you all the Pic4Ser staff, in particular Diego, Gianluca and Neil.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This introductory chapter deals with the meaning of precision agriculture (PA) and the main motivations to look for innovative agricultural techniques. Moreover, there will be presented the main objective of the thesis as well as its relationship with two already existing projects. Eventually, an overview of the main thesis' topics will be given.

1.1 Precision agriculture

"Precision Agriculture (PA) is no longer a new term in global agriculture", as stated in [1]. "The concept of precision agriculture first emerged in the United States in the early 1980s"([2]) and "the first substantial PA workshop was held in Minneapolis in 1992"([1]).
"Precision Agriculture is an integrated information- and production-based farming system that is designed to increase long term, site-specific and whole farm production efficiency, productivity and profitability while minimizing unintended impacts on wildlife and the environment", as written in [1]. It is the first formal definition of PA formulated by the US House of Representatives in 1997.
The previous definition is quite general and encompasses a wide range of interpretations and meanings. First of all, PA deals with the "application of precise and correct amount of inputs like water, fertilizer, pesticides etc. at the correct time to the crop for increasing its productivity, maximizing its yields"([2]) and saving on amount of inputs and related costs. In addition, another most important benefit of PA is the reduced environmental impacts, because"applying the right amount of chemicals in the right place and at the right time benefits crops, soils and groundwater, and thus the entire crop cycle", as stated in [2]. Furthermore, PA involves the usage of robotics and innovative technologies to achieve the automation of typical agriculture tasks (e.g. the harvesting, the ploughing, etc.) without the human

intervention. All considered, "precision agriculture has become a cornerstone of sustainable agriculture, since it respects crops, soils and farmers"([2]).
Moreover, PA is often referred to cropping industries, but it may involve also fisheries, animal industries and forestry.

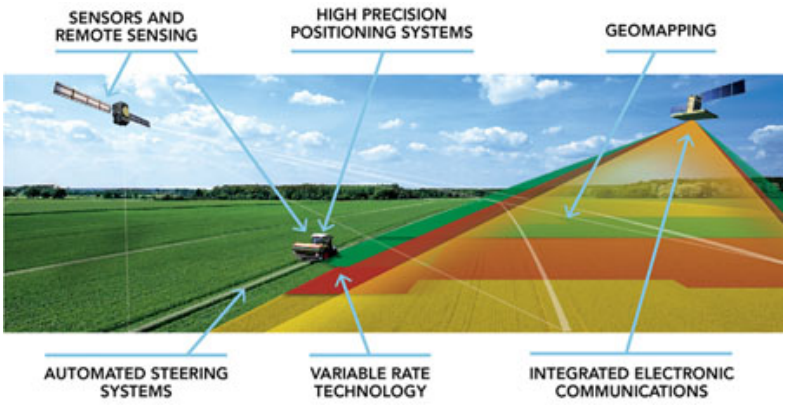


Figure 1.1: Example of a precision agriculture application

1.1.1 Why precision agriculture?

The actual world's population is about 7.7 billion[1] and "it is expected that by 2050, the global population will reach about 9.6 billion, and food production must effectively double from current levels in order to feed every mouth", as stated in [2]. Thanks to new precision farming techniques, "each farmer will be able to feed 265 people on the same acreage"([2]), that is a great step forward compared to 155 people in the 1960s.

1.1.2 Enabling Technologies

The most important enabling technologies is the Global Positioning System (GPS), that is part of the Global Navigation Satellite Systems (GNSS). It allows farmers to precisely identify a position in a field exploiting some corrections algorithms. As a consequence, it "allows for the creation of maps of the spatial variability of as many variables as can be measured (e.g. crop yield, terrain features/topography, organic matter content, moisture levels, nitrogen levels, pH, EC, Mg, K, and others)"([2]). The data, related to the variables of interests, are collected by sensors mounted on GPS-enabled vehicles. "However, recent technological advances have enabled the

[1] https://it.wikipedia.org/wiki/Popolazione_mondiale

use of real-time sensors directly in soil, which can wirelessly transmit data without the need of human presence", as written in [2].
The second enabling technology are the Unmanned Aerial Vehicles (UAVs) "equipped with multispectral or RGB cameras"([2]), that allows to take high quality images to be used for creating maps and optimizing crop inputs.
In addition to UAVs and GNSS, the GPS-based autonomous guidance allows tractors and agricultural machinery to autonomously perform their tasks (e.g. spread fertilizer or plow land), without human intervention unless in case of emergency.
Furthermore, the relatively recent advancements in Internet of Things (IoT) allows the farmers to collect and aggregate a lot of data, through a network of physical objects as well as send commands to IoT devices located in specific places of the farm or fields. Moreover, IoT can be also used for the welfare of animals.
Eventually, machine learning (ML) algorithms can be effectively used in conjunction with autonomous vehicles (e.g. tractors), UAVs and IoT devices, in order to analyse the collected data and send "the appropriate actions back to these devices"([2]).

1.2 Starting points and objective of the thesis

The thesis is focused on the autonomous navigation problem for precision agriculture and it can be considered as the final step to complete two already existing projects. The first one involves autonomous navigation algorithms, based on machine learning and computer vision, while the second project is an innovative approach, which aims to automatically compute a path throughout a vineyard, exploiting Unmanned Aerial Vehicle (UAV) imagery. The two involved projects will be briefly described, while the objective of the thesis will be deeply analysed.

1.2.1 Vine rows autonomous navigation

In [3] and [4] Diego Aghi, Vittorio Mazzia and Marcello Chiaberge propose "a low-cost, power-efficient local motion planner for autonomous navigation in vineyards based only on an RGB-D camera, low range hardware, and a dual layer control algorithm". "The first algorithm makes use of the disparity map and its depth representation to generate a proportional control for the robotic platform. Concurrently, a second back-up algorithm, based on representations learning and resilient to illumination variations, can take control of the machine in case of a momentaneous failure of the first block generating high-level motion primitives", as written in [4]. The designed and developed local motion planner "after several trials with different vineyard rows but similar weather conditions, proved to be able to perform an autonomous navigation along the given paths"([4]). However, the proposed solution is not able to switch from one vine row to the next automatically,

because it was designed "for vineyards rows autonomous navigation", as stated in [4].

1.2.2 Automatic path planning using UAV imagery

In [5] Jurgen Zoto, Maria Angela Musci, Aleem Khaliq, Marcello Chiaberge and Irene Aicardi aim to design and develop "a workflow to generate an automatic coverage path plan for unmanned ground vehicles (UGVs) using georeferenced imagery taken by an unmanned aerial vehicle (UAV)". "First, image acquisition is performed over a vineyard to generate an orthomosaic and a digital surface model, which are then used to identify the vine rows and inter-row terrain. This information is then used by the algorithm to generate a path plan for UGV", as written in [5]. Eventually, "the experimental results show that the work as a whole presents some significant contribution in coverage path planning for UGV in the challenging environment like hilly vineyards", as stated in [5].
The computed global path, through the use of the A* algorithm, is expressed by means of GPS waypoints, that can be directly used by an UGV.
Eventually, the authors has proposed a solution to the problem of ordering the GPS waypoints such that the UGV follows them in the right order. They has developed a sorting functions, which "minimizes the euclidian distance between the current position and the following one"([5]). Where, the current position is referred to the actual position of the UGV, that should be computed before applying the sorting algorithm.

1.2.3 Objective of the thesis

The goal of the thesis is to design and develop an autonomous navigation system based on GPS, that is able to autonomously navigate throughout a vineyard, avoiding dynamical obstacles and switching from one vine row to the next in the most precise way. First of all, a global path made of GPS waypoints should be available thank to the existing methods and algorithms described in [5]. Second, the autonomous navigation inside the vine rows can be easily performed by the local motion planner developed by Diego Aghi, Vittorio Mazzia and Marcello Chiaberge. Eventually, the real purpose of the autonomous navigation system is to given support to the local navigation algorithm inside the vine rows and take the complete control of the autonomous vehicle when it is required to switch from one vine row to the next. All considered, the solution to the autonomous navigation problem should act as a local planner, that is able to follow the GPS waypoints and, at the same time, to work together with the already existing local motion planner presented in [4], in order to achieve the complete autonomous navigation in a vineyard.

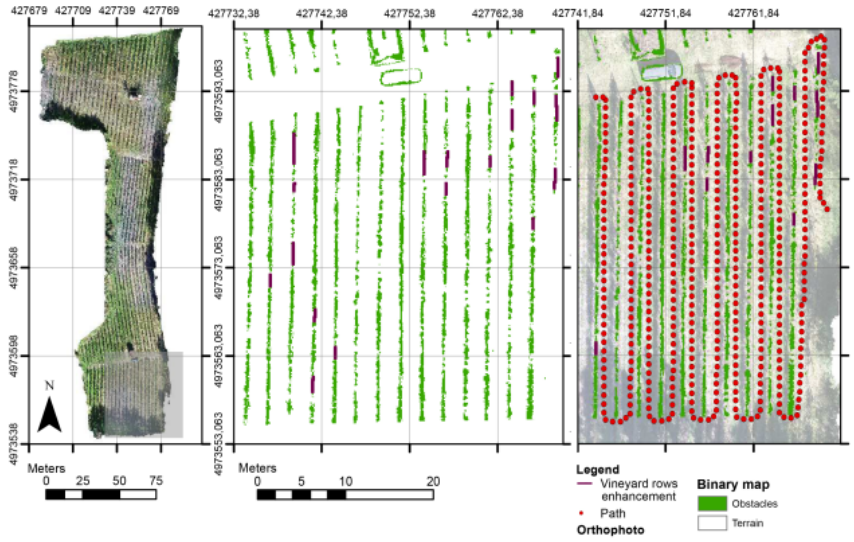


Figure 1.2: Main phases of the used approach. "From left to right: overview, vineyard enhancement, path plan"([5])

1.3 Contents organization

The thesis covers different topics that are organized in chapters as follows:

- Chapter 2: it starts with a short introduction to field robotics in order to understand its the relationship with the main purpose of the thesis. In addition, this chapter describes the main issues related to the application of field robotics in precision agriculture and deeply illustrates the vineyard environment with the related problems.
- Chapter 3: this chapter provides detailed descriptions of tools, approaches, methods and algorithms described in literature and related to the goal of the thesis.
- Chapter 4: it describes the algorithms and software components that has been used to develop the GPS-based autonomous navigation system.
- Chapter 5: it illustrates the simulation tools and the procedures that has been exploited to observe the behavior of the autonomous navigation system in a virtual environment. Eventually, it shows the results coming from simulations.
- Chapter 6: it provides an overview of the most important aspects of the proposed solution as well as the future improvements and works that may be taken into account.

Chapter 2

Challenges in precision agriculture

The following part deals with a brief introduction to field robotics and the challenges that such technology has to face with. In particular, there will be described the issues related to precision agriculture and to the vineyard environment.

2.1 Field Robotics

Field robotics is the wide branch of the robotics world, that deals with "the automation of vehicles and platforms operating in harsh, unstructured environments"([6]). This branch "encompasses the automation of many land, sea and air platforms in applications such as mining, cargo handling, agriculture, underwater exploration and exploitation, highways, planetary exploration, coastal surveillance and rescue, for example", as stated in [6]. In general, "field robots are mobile platforms that work outdoors, often producing forceful interactions with their environments, with no human supervision"([6]).
Field robotics encompasses a wide range of applications, as a consequence it has to face with a lot of challenges, but in the following part there will be described only those related to the precision agriculture application.

2.1.1 Localization issues

"The problem of localization has been the subject of considerable work", as stated in [6]. However, it has become more tractable thanks to the GPS in different field robotics applications. "This is particularly true in environments where good views of the sky can be guaranteed"([6]), otherwise the GPS does not work well, as it is explained in 3.3. In the last years, "the development of low cost inertial

sensing has also been a major advance for these applications"([6]), that have lead to the commonly used combined GPS inertial systems. In addition, when the GPS localization is very poor, "there has been a common use of lasers and radar to observe the relative location of artificial landmarks or beacons and, by referencing these to a map, to deduce position" as stated in [6]. Eventually, "the use of natural landmarks for relative navigation is also reasonably well developed. Good examples include the use of vision to delineate crop lines in agricultural applications", as written in [6]. All considered, the localization problem is still open and depends on the specific robotics application. Nevertheless, it can be partially solved by employing different sensors to acquire the position information and merging such information together through sensor fusion techniques (described later in 3.4).

2.1.2 Perception issues

"Probably the most complex and demanding research issue facing field robotics is the full 3D perception and understanding of typical unstructured environments", as stated by Chuck Thorpe and Hugh Durrant-Whyte in [6]. For instance, "the problem of land-vehicle terrain estimation from sensors such as vision and lasers"([6]). "The reliable construction and understanding of terrain models is still some way off"([6]), although many research groups have employed their efforts to better understand and solve the problem. All considered, "perception systems are becoming adequate for well-defined tasks and well-defined objects"([6]), but the "more general recognition is still difficult", as stated in [6]. "Parts of the problem could be solved by better sensors or innovative combinations of sensing", as written in [6].

2.1.3 Outdoor environment issues

The outdoor environments are often unstructured and some problems coming from them are strictly related to the perception sensors, since a robot can increase its knowledge about the surrounding environment only through sensing. The main problem arising from an unstructured environment is to deal with terrain, because "perception works adequately for some indoor mobility tasks"([6]), but "existing sensing systems are inadequate for driving over outdoor terrain"([6]). "New sensors or sensing strategies need to be developed for detecting surface conditions (mud, ice, loose gravel), inferring partially-observable surfaces (rocks and ditches covered by grass), and estimating other surface properties (bearing strength of dirt, slip angle of sand)", as stated in [6]. "A second major issue is how to plan and make robust decisions in environments with little structure or model"([6]).

2.2 The vineyard environment

A vineyard is composed of vine plants arranged in parallel straight lines, called vine rows, that have almost the same length. Both the plants in the same vine row and different vine rows are placed at fixed distance from each other. This kind of organization can be considered as a partially structured environment or semi-structured environment, because vineyards share almost the same skeleton, though vine plants can be different each other, in terms of dimensions. The main morphological characteristics of a vineyard that can make the autonomous navigation a challenging task are:

- The soil: the vineyard terrain is mainly rough, bumpy and has a different consistency from one place to another (as can be seen in Fig. 2.1). This may cause the wheel slippage event very frequent and make the robot platform wobble, as a consequence the on-board sensor measurements will be very noisy (not considering the GPS).

- The vegetative state of the environment: during different seasons the vegetative aspects of a vineyard can change a lot. In autumn and winter the vineyard is mainly barren, while in spring and summer the vine plants are completely covered with leaves and the ground terrain is plenty of grass and weeds. The main issues are caused by a lush vegetation (e.g. weeds and leaves), because the exteroceptive sensors, as vision sensors or LiDAR, may have problems in identifying real obstacles (different from weeds) and the size of vine rows is enlarged to the robot eyes. This may lead to path planning issues, because the planner tries to find a collision-free path among the huge number of obstacles, although a lot of them are fictitious.

- The crops type: the vineyard can be located on terrains with different slopes and it is possible to identify three main types of crops:

  1. Flat crop: the vineyard is placed on a flat terrain almost on the same sea-level.
  2. Hillside crop: the terrain has a slope less than 40%.
  3. Mountainside crop: the vine plants are placed on a terrain with more than 40% of slope.

  Moreover, the vine rows can be placed horizontally or vertically with respect to the terrain slope (as can be seen in Fig. 2.2). All considered, the slope influences the battery consumption and motion control, as well as other navigation aspects.

Figure 2.1: Three different kind of vineyard soils

Figure 2.2: Two kind of vine rows disposal

Chapter 3

State of the Art

This chapter deals with the required background and tools to suitably achieve the thesis objective. The software components of the autonomous navigation system will be developed by exploiting the Robot Operating System (ROS) and taking into account the relevant theoretical and/or practical information related to autonomous navigation and described in literature.

3.1 ROS

Robot Operating System (ROS) is an open-source, not real-time, meta-operating system largely used in robotics. It offers, almost the same, services as a standard operating system, such as: "hardware abstraction, low-level device control, message passing between processes and package management"([7]). As stated in [7],"It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers", as some of the available robot frameworks.
At run-time ROS can be represented as "a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure"([7]) and possibly distributed over different machines. It supports both the synchronous and asynchronous communication among processes, in order to satisfy the different user needs.
The main goals of ROS are:

- Sharing and Collaboration: supporting the development of distributed software as a collection of loosely coupled processes, that can be grouped into packages and can be reused by different users for different purposes.

- Agnostic libraries: to develop "ROS-agnostic libraries with clean functional interfaces" ([7]).

- Language independence: "the ROS framework can be easily implemented in any modern programming language and it has been already implemented in Python, C++ and Lisp", as written in [7].
- Easy testing: "It has a builtin unit/integration test framework called rostest, that makes the testing phase easy", as stated in [7].
- Scaling: "It is appropriate for large runtime systems and for large development processes" as can be read in [7].

ROS can run on different operating systems, but the official supported one is: Ubuntu Linux.

3.1.1 ROS Filesystem

The ROS Filesystem describes the main ROS resources, that can be stored on a disk:

- Packages: "They are the main unit for organizing software in ROS. A package may contain ROS runtime processes, a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together", as written in [8]. They are the most granular ROS items, that can be built and released.
- Metapackages: "They are special kind of packages which only serve to represent a group of related other packages" ([8]).
- Package Manifest: "It (package.xml) provides metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages"([8]).
- Message types: A message is the unit of data exchange among ROS processes. Each message has a predefined structure described by its message type.
- Service types: ROS processes may ask for a particular task (service) to each others. A service is composed of a request and a response, that have a predefined data structure, described by a service type.

3.1.2 ROS Computation Graph

The Computation Graph is a representation of "the peer-to-peer network of ROS processes (Fig. 3.1), that are processing data together", as stated in [8]. The main Computation Graph concepts of ROS are:

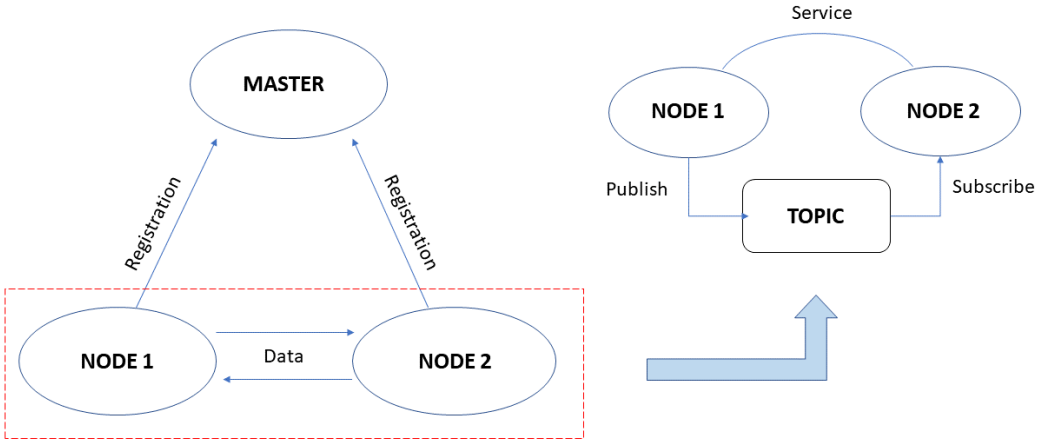


Figure 3.1: Basic peer-to-peer network of ROS processes

- Nodes: "A node is a process that performs computation. ROS is designed to be modular at a fine-grained scale, thus a robot control system usually comprises many nodes"([8]), that communicate and collaborate to each other in order to achieve an advanced task. "A ROS node is written with the use of a ROS client library"([8]).

- Master: The ROS Master is a particular node, "that provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services", as written in [8].

- Messages: "Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields."([8])

- Topics: "Messages are routed via a transport system"([8]) based on the publish/subscribe communication paradigm. "A node sends out a message by publishing it to a given topic. The topic is a name that is used"([8]) to identify a communication flow between publishers and subscribers. "A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. The general idea is to decouple the production of information from

its consumption"([8]), thus publishers and subscribers are not aware of each others' existence.

- Services: The communication paradigm used by topics is a very flexible model, "but its many-to-many, one-way transport is not appropriate for request/response interactions, which are often required in a distributed system"([8]). Request / response can be obtained via services, "which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply", as stated in [8].

- Bags: They are a way "for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensors data, that can be difficult to collect but are necessary for developing and testing algorithms", as written in [8].

"The most common communication protocol used in ROS is called TCPROS, which uses standard TCP/IP sockets"([8]).

3.1.3 ROS Community

The ROS Community is made of resources, "that enable separate communities to exchange software and knowledge"([8]), to achieve collaboration. The main involved ROS resources are:

- Distributions: "They are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions, making easier to install a collection of software, and they also maintain consistent versions across a set of software", as stated in [8] .

- Repositories: "A collection of packages (or a single package) which share a common Version Control System (VCS). A repository is a virtual-place where different institutions can develop and release their own robot software components (packages)"([8]).

- The ROS Wiki: "It is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more"([8]).

- Mailing Lists: "The ros-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software", as stated in [8].

3.2 Map-based autonomous navigation

The map-based autonomous navigation problem can be easily subdivided into the general concepts of localization, path planning and obstacle avoidance, that can be roughly summed up as: where I am? (localization) and how can I move from point A to point B without hurting anything or anyone? (path planning and obstacle avoidance). Moreover, an autonomous vehicle is expected to be equipped with sensors in order to perceive the surrounding environment and act in the most appropriate way.
Path planning deals with the computation of a collision-free path from a starting point to an ending point inside a given map taking into account: static obstacles, the motion model of the autonomous vehicle under study and some optimization functions, such as minimum time, minimum energy etc. There exist path planning algorithms for both 3D map and 2D map, but only those related to 2D map will be described, since the thesis deals with autonomous navigation for UGV. In addition, the path planning task is commonly divided into: global path planning and local path planning. The former, also called off-line path planning, exploits the map information and an optimization function (listed before) to plan a global collision-free path from a point A to a point B, while the latter, also called on-line path planning, is usually responsible for computing a local collision-free path, exploiting sensors information (to detect dynamic obstacles), the dynamical constraints and the dynamical model of the specific vehicle.

3.2.1 Sensors for autonomous navigation

In general, autonomous vehicles, as the unmanned ground vehicles (UGV), are equipped with a set of sensors in order to sense the surrounding spaces, to monitor the internal state of the vehicle and to provide specific information to the navigation and localization algorithms. Furthermore, sensing devices can be divided into two classes: proprioceptive and exteroceptive sensors([9]).

Exteroceptive sensors

This kind of sensors is used to extrapolate (sense) information from the surrounding environment. The most used exteroceptive sensors are:

- GPS receivers: The GPS is based on the global navigation satellite system and can be used to obtain global positioning of an autonomous vehicle as explained in 3.3.
- Laser: It is a range sensor used to obtain information about the objects in the surrounding environment and is commonly known as LiDAR (light detection

and ranging). It is mainly composed of a source of light (laser), which emits light impulses of electromagnetic waves and a receiver, that detects the reflected light from the surrounding obstacles. It exploits the time of arrival (TOA), that is the difference in time between the detection and the emission of a particular light pulse, to compute the range and bearing of an object (obstacle). There exist two dimensional (2-D) and three dimensional (3-D) LiDAR sensors, that are based on the same principle of TOA, though the way to determine the distance vary among the manufactures. This kind of sensors is able to detect from the small particle in the atmosphere (aerosol, $1\mu m - 1nm$) to greater visible objects, thanks to the small wavelength ($10\mu m - 250nm$) of the emitted light. This particular characteristic allows to adopt a LiDAR sensor in a huge number of different applications, though makes it very sensitive to atmospheric conditions. Moreover, a LiDAR sensor is able to observe wide field-of-view (FOV), long ranges and it is precise. All considered, LiDAR are very good range sensors, but they are more expensive compared to other range sensors.

- Radar: It is similar to the LiDAR sensor in terms of working principle, but it exploits a different kind of wave. In particular, it is uses radio waves, that, compared to the light waves, travel at the same speed, but have much lower frequencies and greater wavelength. Moreover, it is able to take into account the Doppler shift of the echo, so it can determine the velocity of a moving object without further numerical processing. With respect to the LiDAR, the radar can operate over longer distances, it is less sensitive to weather conditions, but it has a lower resolution due to the greater wavelength. In addition, the radar FOV is usually wider compared to the LiDAR one, though the bearing measurement of radar is less accurate. Eventually, a radar is cheaper than a LiDAR and, all considered, the radar is better than the LiDAR to detect or track objects.

- Vision sensor: The stereo camera is the most common vision sensor used in autonomous navigation system. It is a particular camera, that exploits two separate lenses to retrieve the same scene from different perspectives, then comparing the two different images a 3D representation of the observed scene can be obtained. Triangulation is the commonly used technique to reconstruct the 3D image, starting from two or more 2D images. To better understand triangulation the ideal case will be described and some considerations about the real approach will be addressed.
  In the ideal case, two different lenses are approximated by two separate pinhole camera, placed on the same horizontal axis, at different positions and with

parallel focal axis, as can be seen in Fig. 3.3. The pinhole camera model[1] has no lenses, as a consequence no distortion due to them can occur. In Fig. 3.3

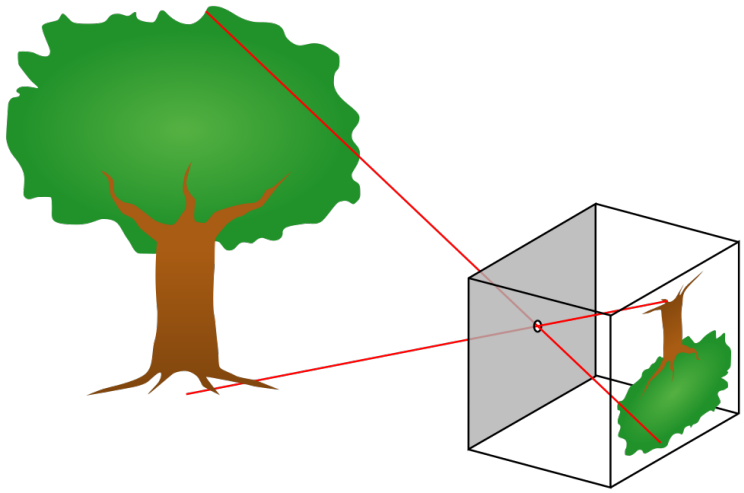


Figure 3.2: Example of a pinhole camera working principle

the focal points of the two cameras are placed behind the corresponding image planes, so that the projected image is no upside down as shown in Fig.3.2 and calculations are more intuitive. However, in reality, they are in front of them. f (focal length) is the distance between image plane and focal point, b (baseline) is the distance between the two focal points, while the global reference frame $(X_|, Y_|, Z_|)$ is set at the origin of the left camera's focal point. $P(x_p, y_p, z_p)$ is the 3D real point, $P1(u_1, v_1)$ is the projection of P on the left image plane and $P2(u_2, v_2)$ is the projection of P on the right image plane. Starting from $P1$ and $P2$ and exploiting geometrical considerations about similar triangles, it is possible to reconstruct P as follows:

$$u_1 = f\frac{x_p}{z_p} \qquad u_2 = f\frac{x_p - b}{z_p} \qquad v_1 = v_2 = f\frac{y_p}{z_p}$$

$$d = u_1 - u_2 = f\frac{b}{z_p}$$

$$x_p = b\frac{u_1}{d} \qquad y_p = b\frac{v_1}{d} \qquad z_p = b\frac{f}{d}$$

where d is the visual disparity. In a real approach each 3D point is mapped into 2D points on the image plane, through pixels. Each 2D image has a huge

[1]https://en.wikipedia.org/wiki/Pinhole_camera_model

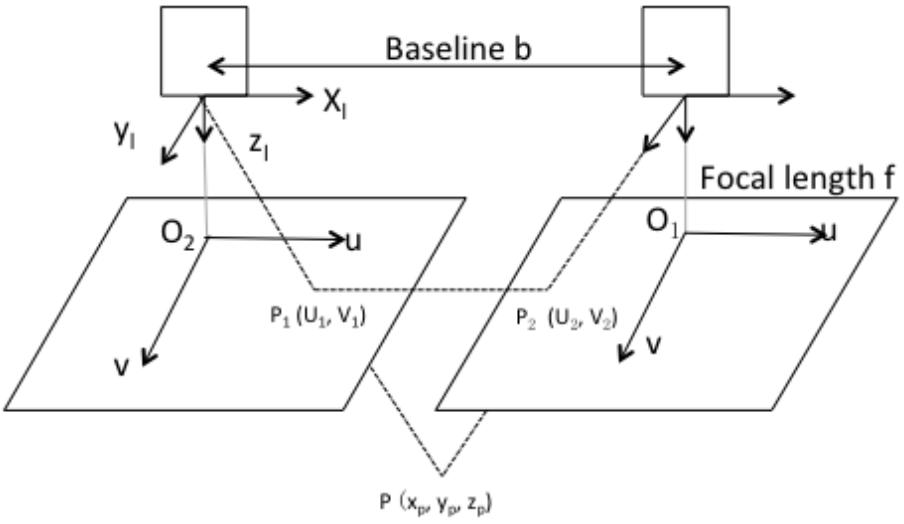


Figure 3.3: Triangulation working principle

number of pixels, so it may be computationally expensive to compare one pixel of the one image plane with all the other pixels (in the worst case scenario) of the second image plane , in order to find the corresponding point P_2, that is the counterpart of P_1. Moreover, the two cameras may not have parallel focal axis and may have different orientation from each other. However, this process can be simplified, thanks to a set of geometric relations between the 3D points and their projections onto the 2D images that lead to some constraints between the image points, described by the epipolar geometry[2]. As can be observed in Fig. 3.4 X, X_1, X_2, X_3 are represented by the same x_L on the left image plane, but can be represented by different x_R on the same line (epipolar line) in the right image plane. e_L and e_R are the projection of the two focal centers, respectively O_R and O_L, in the opposite camera's image plane and are called epipolse or epipolar points. O_R, O_L e_L, e_R lie on the same 3D line

[2]https://en.wikipedia.org/wiki/Epipolar_geometry

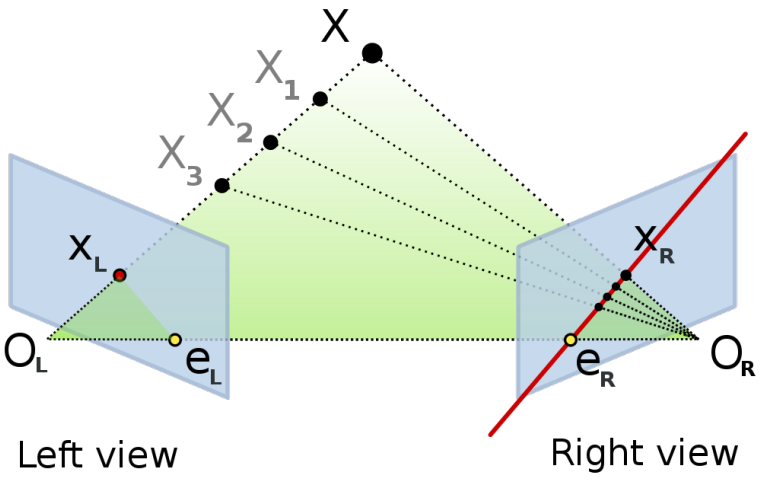


Figure 3.4: Epipolar geometry between real world and image planes

and O_R, O_L with X form the epipolar plane. Eventually, different 3D points form different epipolar planes with O_R, O_L and , as a consequence, different epipolar lines. The previous described geometrical constraints can simplify the computation of the 3D point coordinates supposing to know the relative position of the two cameras, the projection x_L and the epipolar line formed by e_R and x_r. For each 3D point X projected on the left image plane (x_L), there exist a projection on the right image plane (x_R), that lies on a specific epipolar line (formed by e_R and x_r) (epipolar constraint). The epipolar constraints may be described by the 3x3 fundamental matrix F, which relates corresponding points in stereo images and can be estimated through a process, that involves point correspondences. After the epipolar constraints have been applied, x_R and x_L are supposed to be known, as a consequence, also their projection lines are known. Finally, the triangulation method can be easily applied to the projection lines, in order to find the right 3D coordinates of X.
In a real approach, cameras uses lenses to take a picture of the surrounding environment and (as stated before) they may not have parallel focal axis. Indeed "several pre-processing steps are required"([10]), in order to make comparable the two images of the same scene, because they may be affected by distortion and may be not co-planar as in Fig. 3.3. The main steps to be taken into account are:

– Correction of undistorted images: "Distortion is a deviation from rectilinear projection"([11]), caused by lenses; it may be of different types (as shown in Fig. 3.6), but the most common are the radial and the tangential

distortions. Depending on the lens types, different kind of radial distortions may occur, such as: barrel distortion or pincushion distortion, while the tangential distortion occurs when the image plane and the lens are not parallel. Compensating for this undesired lens effect, "ensures that the observed image matches the projection of an ideal pinhole camera"([10]).

– Image rectification[3]: It is a process, based on a transformation matrix, to project images onto a common image plane (as represented in Fig. 3.7), in order to make them co-planar. The used transformation matrix strictly depends on the relative position of the cameras and other cameras' coefficients, that can be obtained through the camera calibration process (described later). Moreover, it could be advisable to eliminate distortion effects from images, before applying image rectification, in order to simplify the whole process.

All considered, stereo vision cameras are complex vision sensors, that involves a lot of intrinsic (focal length and location of the optical center) and extrinsic (position of the camera with respect to the 3D world) parameters. They must be estimated in the most precise manner, through camera calibration[4], in order to allow a correct representation of the surrounding environment. To easily understand how the camera calibration process works, it will be applied to a pinhole camera model, thus it does not take into account any form of distortions due to lenses. The 3D environment is mapped to the 2D image plane, through the following process:

$$pixel = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad X = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \qquad K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

where: *pixel* contains the 2D point position in pixel coordinates, X contains the 3D point position in world coordinates, K is the matrix of the intrinsic parameters, α_x and α_y represent the focal length in terms of pixels, u_0 and y_0 represent the principal point (ideally in the center of the image), γ is the skew coefficient between the x and y axis.

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R \quad T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.1}$$

[3]https://en.wikipedia.org/wiki/Image_rectification

[4]https://en.wikipedia.org/wiki/Camera_resectioning

Eq. 3.1 is the 3D-2D mapping equation, where: w is a scaling factor, R and T are, respectively, the rotation and the translation matrix between the 3D world coordinates frame and the 3D camera coordinates frame, thus they contain the extrinsic parameters. Eventually, the unknown camera parameters can be found through different methods, such as: Direct Linear Transformation (DLT), Zhang's method and Tsai's method, while the distortion effects should be taken into account through specific transformations, depending on the lenses type.

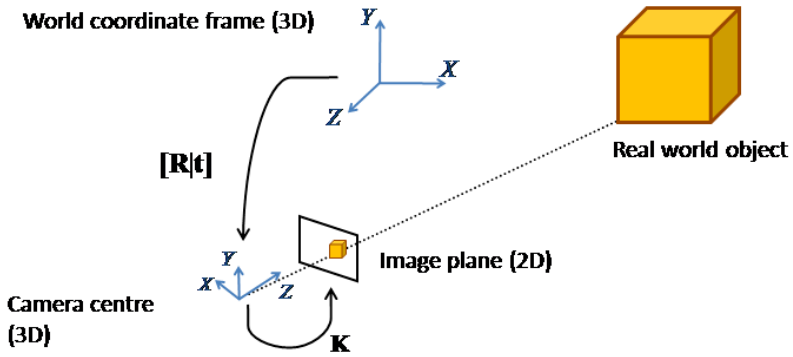
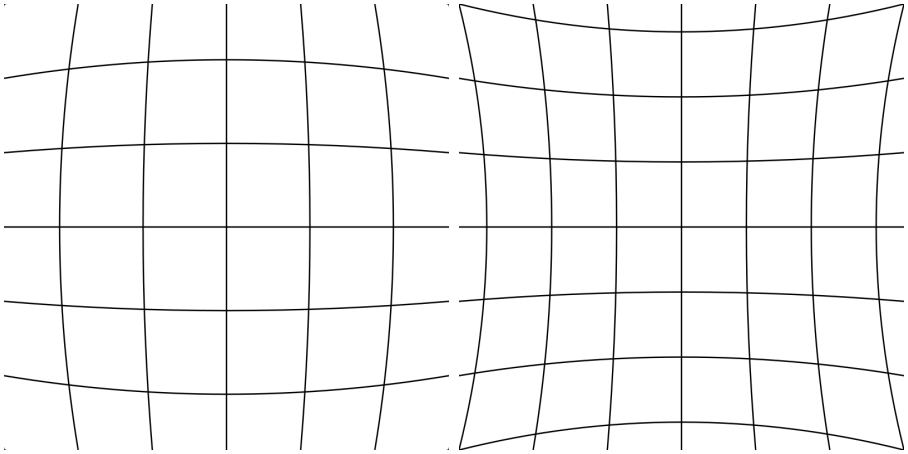


Figure 3.5: From world coordinates frame to camera coordinates frame([12])

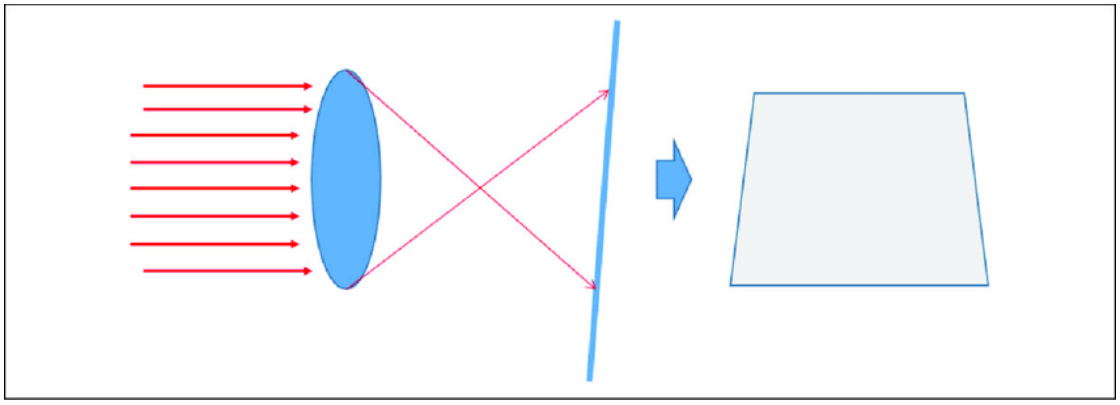
Proprioceptive sensors

The proprioceptive sensors are usually used to collect data about the internal state of dynamical systems. In the particular case of UGV, the velocity, the steering angle, the orientation among the principal axis (x, y, z) and the heading with respect to the magnetic north. Among the proprioceptive sensors, the most common and used are:

- Encoders: they are vehicle motion sensors, typically used to estimate the velocity and the steering angle of a ground vehicle. Among different types, the optical incremental encoders are the most suitable to measure the velocity, while the optical absolute encoders are typically used to estimate the steering angle. The former are composed of a grid disk, a light source and an optical detector. The grid disk is usually attached to the rotating axle of the vehicle, so that when the vehicle moves the light passes through the holes on the disk and the light detector can observe the varying amount of light. The process output is usually a square wave, in which high represents the presence of light and low means absence of light. Starting from the square wave is possible to estimate the angular velocity of the axle, with two main methods. The first



(a) barrel distortion **(b)** pincushion distortion



(c) tangential distortion([13])

Figure 3.6: Image distortions

one is the pulse-counting method, that works as follows:

$$t = N\frac{T}{n} \qquad\qquad \omega(rad/s) = \frac{2\pi}{t} = 2\pi\frac{n}{TN}$$

where n is the number of pulses in the T sampling period, N is the number of windows on the disk, t is the average time for one revolution and ω is the estimated wheels rotational velocity, from which the linear velocity of the vehicle can straightforwardly be obtained.
The second one is the pulse-timing method, that is based on an high frequency clock:

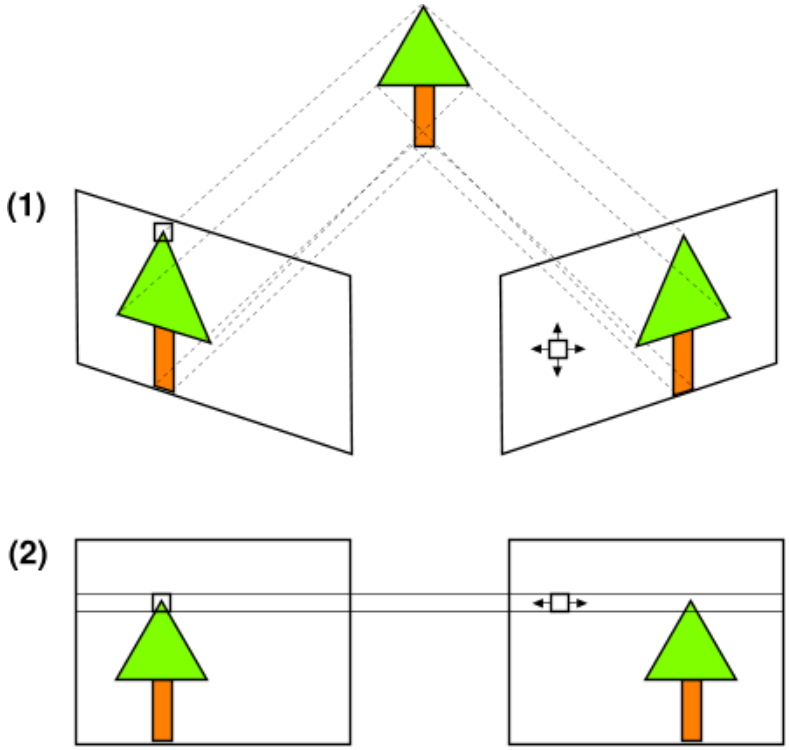
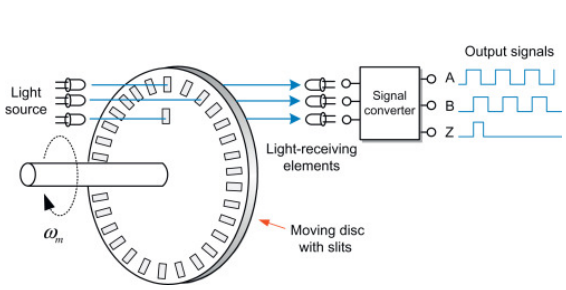


Figure 3.7: Image rectification process

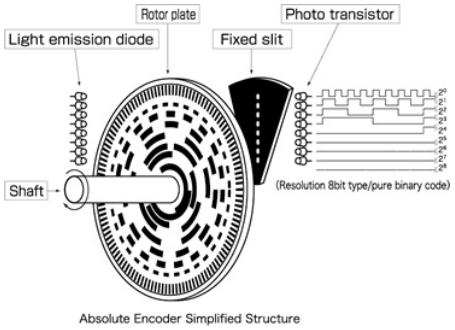
$$t = N\frac{m}{f} \qquad\qquad \omega(rad/s) = \frac{2\pi}{t} = 2\pi\frac{f}{Nm}$$

where, f is the clock frequency, m is the number of counted pulses during the time interval between two adjacent windows of the disk, t is the average time for one revolution and ω is the estimated rotational velocity.
If the disk is equipped with a second pair of light source and optical detector, suitably positioned, it is possible to obtain the rotational direction, measuring the phase difference between the two square waves.
The optical absolute encoder is composed of a coded disk "imprinted with rows of broken arcs"([9]), light sources and light detectors. "The arcs are arranged in patterns to be encoded", as stated in [9]. Each pattern is uniquely identified by means of light sources and sensors to observe them. This kind of encoders are used to measure the angle of the wheels with respect to the forward direction of the vehicle, so that it is possible to estimate the steering angle. As the wheel rotates, a specific pattern is detected and the corresponding wheel's angle is obtained.
The encoder's resolution depends on many factors, but the main ones are:

number of windows on the disk, number of bits used to encode the output, the gear ratio (a gear usually is placed between the axle and the encoder), sampling period T and clock frequency f. Moreover, they are very accurate transducers, although they may estimate inaccurate measurements under certain conditions, such as: wheel slippage, skidding and different wheel diameters among wheels.



(a) Incremental encoder



(b) Absolute encoder

Figure 3.8: Optical encoders

- Inertial Measurement Unit (IMU): it is an inertial sensor, typically used to estimate the linear and rotational motion of a vehicle. The most common IMUs are equipped with: three orthogonal accelerometers, three orthogonal rate gyroscopes and they may contain magnetometers. The accelerometers are used to estimate the acceleration along the three principal axes of motion (x, y, z), while the gyroscopes are used to measure the angular velocity along the same axes of motion. The presence of magnetometers can be exploited to measure the heading of the vehicle with respect to the magnetic north.

3.2.2 Localization in a known environment

Localization is a difficult task and one of the main current challenges in autonomous navigation system. It can be divided into: pure localization when the surrounding environment is known (map-based localization) or simultaneous localization and mapping (SLAM), when the autonomous vehicle does not have a map of the environment and is expected to simultaneously build a map and localize itself. Since, one of the starting points of the thesis is a known geo-referenced map, only the map-based localization technique will be analysed, although some of the described approaches can be used both for map-based localization and SLAM.

Global Localization

The global localization is particularly required in outdoor environments, in which the autonomous vehicle is expected to travel relatively long distances, that may includes different maps, since a map can only contain a bounded portion of the global environment. The main global localization technique is the GPS (described in 3.3), based on the GNSS. It is usually used together (through sensor fusion 3.4) with vehicle motion sensors and/or inertial sensors, in order to compensate the lack of the GPS signals in some specific outdoor environment conditions, such as: the presence of tall trees or any other tall obstacles, that obstruct the receiver-satellite view.

Relative Localization

Autonomous vehicles can exploit their on-board sensors information to estimate their current position and heading with respect to a known initial position. The main and commonly used approaches are:

- Dead reckoning: it is the oldest method used to relatively localize a vehicle. The easiest solution is to integrate over the time the information (velocity and steering angle) coming from the wheel encoders to obtain an estimate of the current position and heading. More advanced technique exploits the fusion of different on-board sensor information to provide an approximate pose (it is a vector, that contains information about position and heading of a vehicle). However, this approach is very weak in presence of wheel slippage, skidding or dynamical variations of some sensor parameters, because the error accumulates over the time, unless it is periodically compensated with the exact and current pose of the vehicle.

- Inertial sensors: it is possible to implement dead reckoning using only the information coming from inertial sensors, such as IMUs. In this case, the current position and heading of the autonomous vehicle is obtained by integrating over the time linear accelerations and angular velocities coming from the accelerometers and gyroscopes (inside the IMU), respectively. Moreover, the vehicle's heading can be directly obtained by the magnetometer (if it is available) inside the IMU. This approach is not subject to unsystematic errors, such as wheel slippage or skidding, but it has some intrinsic limitations of dead reckoning, such as the error drift due to the integration over the time of physical quantities.

- Visual odometry: "the first known real-time visual odometry was implemented in 1980"([9]), but only in recent years it is gaining attentions from research groups and industries, thanks to the improvements in computer vision and

the availability of hardware computational power. The basic idea of visual odometry is to process the images captured at each instant (by cameras) in order to compute the relative transformation by two consecutive images and estimate the vehicle's trajectory. Unlike dead reckoning, visual odometry is not affected by any errors that are terrain or vehicle parameter-dependent, such as a vehicle skidding or wheel slippage. The vehicle's pose can be estimated by means of two main methods: the appearance-based method and the feature-based method, as stated in [9]. "Both the approaches estimate the vehicle's motion by observing the approximate movement of a pixel or a feature from one image to next"([9]), exploiting computer vision techniques.

- LiDAR-based localization: it is an alternative approach with respect to visual odometry, that exploits 2D or 3D LiDAR to match what actually the autonomous vehicle is sensing and the information coming from an available map, as proposed in [14]. Compared with the visual odometry approach, it provides more accurate vehicle's pose information, thanks to the high accuracy measurements of LiDARs, but it is more expensive due to the higher cost of LiDARs with respect to modern cameras.

3.2.3 Map and Obstacle representation

The starting point of path planning algorithms may be a map, that contains information regarding the surrounding environment, such as the presence of static obstacles, their dimensions and the amount of free space.
A given map should be properly represented, in order to be efficiently used by path planning algorithms. In literature, as explained in [15], there exist a lot of methods to suitably represent a map and the relative obstacles, but in the following part, only the most important ones will be presented:

Cell Decomposition Methods

The cell decomposition methods are the most studied and widely used in outdoor robotics navigation, as stated in [15]. They basically decomposed the map space into discrete, non-overlapping regions, called cells, obtaining a graph, in which each cell is adjacent to other cells. As a consequence, the graph can be efficiently used by path planning algorithms to look for the desired path, that will be composed by a sequence of adjacent cells. In addition, the cost of traversing a cell may vary. The main cell decomposition methods are:

- Approximate decomposition: the entire map is broken up into cells of equal size and shape. The center of each grid element (cell) becomes a node in the graph used by the path planning algorithm. Each cell can assume one of three

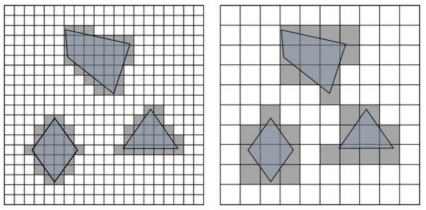
predefined value: full (if an obstacle occupies the entire grid element), partially full (if an obstacle partially occupies the grid element) or empty (if the cell is completely free). The approximated map is called either bitmap if each cell has a binary representation or occupancy grid if each grid element has a range of values. The main advantages of this technique are: the simplicity, thanks to equal sized and shaped grid elements and the flexibility as the size of grid elements can be increased or decreased at run-time, while the main disadvantages are: the memory occupation as the number of grid cells increases and the conservativeness, due to the non-traversability of a partially occupied cell. All considered, it is one of the most used technique in outdoor robotics thanks to its easy application and flexibility.

- Adaptive cell decomposition: this approach aims to reduce the memory occupation and computation time. It exploits the fact that large areas in outdoor environments are more likely to be free, as a consequence, as stated in [15], "The regular shape of the cells is maintained, but the cells are recursively reduced in size in order to both use the space more efficiently and maintain as much detail as possible. The result is less memory required and less processing time". The main drawback of this technique is the re-computation of the entire data structure of the map, in case of map update with new sensed obstacles. One of the most common types of adaptive cell decomposition methods is called quadtree.

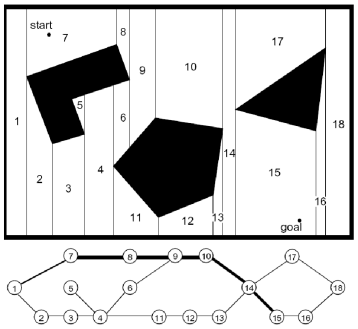
- Exact cell decomposition: this method tries to overcome some problems coming from regular decomposition grids. The cells have not equal size and shape, but are determined based on the environment map, the location and shape of obstacles within it. As stated in [15], "Unfortunately, there is no simple rule for how to decompose the space into cells. This method is quite difficult to apply for outdoor environments where obstacles are often poorly defined and of irregular shape."

Roadmap Methods

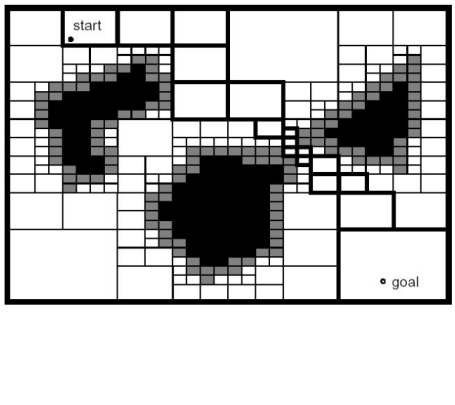
As stated in [15], "Roadmaps are graphs which represent how to get from one place to another". This method builds the connections between free space regions as a set of one dimensional curves, that will be used by some planning algorithm to look for a desire path. The nodes of the graph are usually created using distinctive locations that the autonomous vehicle can easily identify. On one hand, as written in [15]: "Roadmaps provide a huge advantage over cell decompositions in the number of nodes a planner needs to search through in order to find a path." On the other hand, the main disadvantages of roadmaps are: the selection of the most suitable



(a) Approximate cell decomposition ([16]).



(b) Exact cell decomposition([17]).



(c) Adaptive cell decomposition: quadtree

Figure 3.9: Examples of cell decomposition methods.

nodes and the recomputation of the entire roadmap as new sensor information arrive. The main type of roadmaps are the following ones:

- Visibility Graphs: it is one of the earliest roadmap methods , that can be applied to 2D maps. In this case, the set of one dimensional curves are represented by straight line segments which connect the nodes of the polygonal obstacles, without crossing the interior of them. The straight line segments are tangential to the obstacles and the autonomous vehicle may be stuck whenever it travels close to one of them. One possible solution to this problem is to enlarge each obstacle region of a certain quantity, in order to provide a safety margin, although this results in incompleteness and inefficiency of the path planner. A second drawback is the representation of the obstacles, that must be approximated by polygons, although in outdoor environments they usually take on round or amorphous shapes.

- Voronoi Diagrams: the roadmap is made up of paths, called voronoi edges,

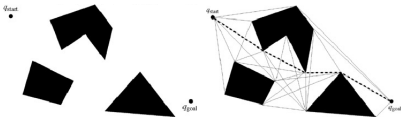
which are equidistant from all the points in the obstacle region. The vertices are the points where the voronoi edges meet and often have a physical correspondence to aspects of the environment which can be sensed, such as intersections of hallways. Voronoi paths are build as far as possible from obstacles, as a consequence the autonomous vehicle, that follows one of them, will not collide with any modeled obstructions, without growing the size of occupied regions. "This makes Voronoi methods safe, but the paths generated inefficient", as stated in [15].

- Probabilistic Roadmaps: the Probabilistic RoadMap (PRM) is a discrete version of a given continuous map, that is generated by randomly sampling the entire map and then connecting the sampled points into a roadmap. The main idea of PRM is that a relatively small number of points and paths are usually sufficient to capture the connectivity of free space regions; assumption that can speed up the planning process. The PRM is divided in two phases: the construction of the roadmap phase and the path query phase. In the first phase, randomly chosen points are added to a list of special points, then a mapping algorithm tries to connect the whole list points, through straight lines. If a point is contained in an obstacle region is discarded and if a connecting line between two points intersect an obstacle is discarded. The process of random selection and connectivity evaluation is repeated a large number of times, until a suitable list of connections (the roadmap) is created. In the second phase a suitable algorithm exploits the previous build roadmap, in order to look for the least-cost path between the initial and the final desired vehicle location. The main advantage of such technique is the very efficient query path phase, while the main disadvantage is the slow and computational expensive roadmap creation process, that should be repeated as soon as obstacles are added or removed from the map.

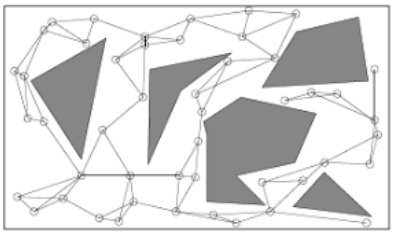
- Rapidly Exploring Random Trees (RRT): it is a variation of the PRM, in which the points are chosen randomly expanding a tree (a path), in order to cover the whole map. The expansion should be toward the areas which have not been filled up yet. Moreover, the RRT are well suited to include dynamic or non-holonomic constraints.

Potential Fields

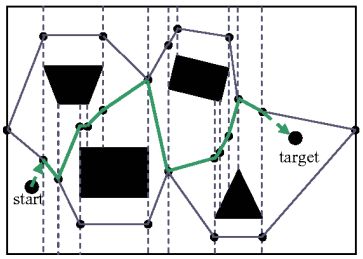
"Potential Fields is the third major type of representation used in path planning", as stated in [15]. It is different from the previous described methods, because it imposes a mathematical function over the entire considered map. The basic idea is to model the autonomous vehicle as a point under the influence of attractive forces generated by the goals and repulsive forces generated by obstacles, like an electron



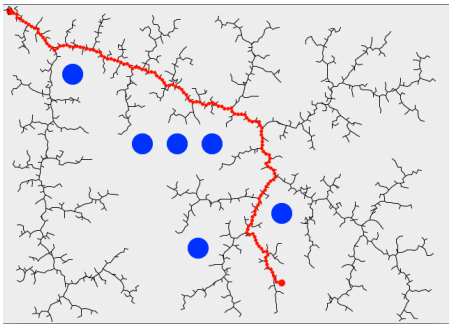
(a) Visibility Graph([18])



(b) Probabilistic Roadmap



(c) Voronoi Diagram([19])



(d) Rapidly Exploring Random Trees

Figure 3.10: Examples of roadmap methods.

in an electric field. As explained in [15] "These forces are stronger near to the obstacle or goal and have less effect at a distance." This kind of representation was originally developed for real-time obstacle avoidance, with emphasis on real-time efficiency, but exploiting particular algorithms it can be also used for global path planning. The main drawback of this approach is the presence of local minima different from the goal, in which the autonomous vehicle can get stuck.
The main path planning algorithms based on potential fields are: Navigation Functions, Depth-First planning, Best-First planning and Wavefront based planning.

3.2.4 Global Path Planning

Once a map representation method has been chosen, it is possible apply some path planning algorithms in order to find the best path according to an optimization criteria. The global path planning algorithms tries to find the optimal path working on a-priori information of the surrounding environment (suitable represented in a map).
In literature, there exist a lot of algorithms, that solves such problems and they can be classified into two main categories: heuristic approach and artificial intelligence (AI) algorithm, as stated in [20]. In the following part the heuristic algorithms will be deeply analysed, while the artificial intelligence ones will be briefly described.

Heuristic Approach

The heuristic algorithms can be classified as graph search algorithms and are more used with cell decomposition methods or roadmap methods, because the result of such representation methods is a graph, that can be easily used by the path search algorithm. The main property of an heuristic algorithm is the use of an heuristic function in order to keep track of the best direction towards the goal. The most common path search algorithms based on an heuristic functions are the following ones:

- Dijkstra Algorithm: it does not contain any heuristic function (it is equal to zero), but it will be described in this section because it is the starting point of the A* and D* algorithms.
  The Dijkstra algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956 and aims at founding the shortest path from a starting node to a goal node in a graph. The edges in the graph can have fixed or variable cost according to the specific application. In the algorithm described later, S is an empty set, Q is a set containing all the graph nodes, $w(u, v)$ is the weight of the edge connecting u and v, w is a set containing all the edges, $dist[u]$ is the minimum distance of u from the starting node s, $prec[v]$ is the previous node connected to v with the minimum distance. The weight $w(u, v)$ can be assigned according to the algorithm application; for instance the weight of an edge can represents how good is the terrain through which an UGV will travel.
  The efficiency of such algorithm is not too high because it visits almost all the graph nodes before funding the shortest path.

- A* algorithm: it was first published in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International). It is based on the Dijkstra algorithm and aims to achieve better performance in terms of efficiency. As stated in [15] "A* is probably the most widely used search algorithm in robotics", thanks to its higher efficiency to find the shortest path with respect to other search algorithms. The key property of the A* algorithm is the usage of an heuristic function $h(n)$ to adjust the cost of traversing from one node to another one. The $h(n)$ function is usually considered as the Euclidean distance between two nodes and represents the nearness to the goal. Defining $f(n) = g(n) + h(n)$ to be the total cost from the start node to the n node, where $g(n)$ is the cost from the starting node to the n node and $h(n)$ is the predicted cost of the optimal path from the node n to the goal node, the shortest path can be found evaluating, at each step, only the neighbours of the node with the lowest value of the $f(n)$ cost. Compared to the Dijkstra algorithm, A* is more efficient, because it visits a

Algorithm 1 Dijkstra Algorithm

```
1: procedure DIJKSTRA(Q, s, w)
2:     Initialization
3:     for each v in Q do
4:         dist[v] = ∞
5:         prec[v] = null
6:     end for
7:     dist[s] = 0
8:
9:     Compute the best path
10:    while Q ≠ empty do
11:        u = vertex with the minimum dist[] in Q
12:        remove u from Q
13:        if dist[u] = ∞ then
14:            break
15:        end if
16:        for each neighbour v of u do
17:            temp = dist[u] + w(u, v)
18:            if temp ≤ dist[v] then
19:                dist[v] = temp
20:                prec[v] = u
21:
22:            end if
23:        end for
24:    end while
25:    return dist, prec
26: end procedure
```

lower number of cell (as can be seen in Fig. 3.11), although it has to compute the heuristic function.

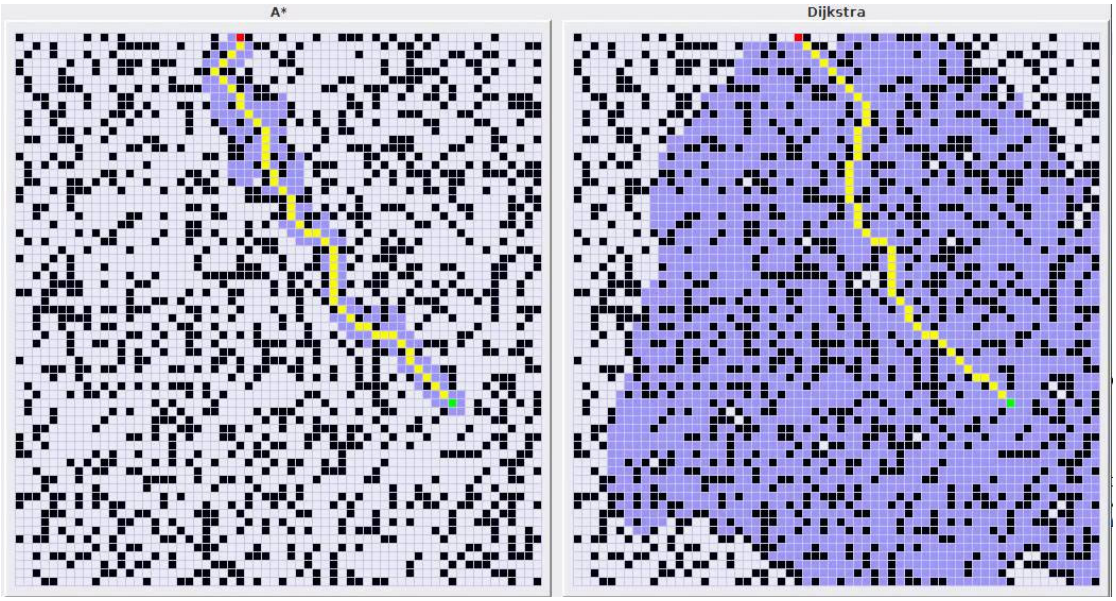


Figure 3.11: Visual comparison between Dijkstra and A* algorithm

- D* algorithm: it refers to three implementation variants: the original D*, the Focussed D* and the D* Lite. The first one is an incremental search algorithm, while the second one is an incremental heuristic search algorithm, that combines ideas of the A* and the original D*; both of them have been developed by Anthony Stentz. The third one is an incremental heuristic search algorithm developed by Sven Koenig and Maxim Likhachev, that builds on Life Long Planning A* (LPA*). The D* algorithm can be used whenever a map is update frequently with new obstacles, since it can re-plan the shortest path in a more efficient way than repeated A* searches. "The name D* comes from Dynamic A*, since the algorithm behaves like A* except that the arc costs can change as the algorithm runs", as stated in [21].
  The nodes can assume one of the following states:
  - NEW: "the node has never been placed on the OPEN list"([21]).
  - OPEN: "the node is currently on the OPEN list"([21]).
  - CLOSED: "the node is no longer on the OPEN list"([21]).
  - RAISE: "the cost of the node is higher with respect to the last time that was in the OPEN list"([21]).

  - LOWER: "the cost of the node is lower with respect to the last time that was in the OPEN list"([21]).

  The OPEN list contains the nodes that have to be evaluated.
  The main operation of the D* algorithm is called expansion, in which "the algorithm iteratively select a node from the OPEN list, evaluates it and propagates the node's changes to all of the neighboring nodes and places them on the OPEN list"([21]). "Each node stores a backpointer to the next node leading to the target and knows the exact cost to the goal", as stated in [21]. The expansion process starts from the goal node and terminates when the start node is the next to be expanded. In case of obstacle detection "along the intended path, all the affected nodes are again placed on the OPEN list, marked as RAISE"([21]). "Before raising the node's cost, the algorithm checks its neighbors, in order to examine whether it can reduce the node's cost", as written in [21]. "If not, the RAISE state is propagated to all the nodes that have backpointers to it"([21]), called nodes' descendants. "When the cost of a RAISED node can be reduced, its backpointer is updated, and passes the LOWER state to its neighbors", as stated in [21]. This propagation of RAISE and LOWER states is the working principle of D*. Moreover the algorithm re-visits only the nodes which are affected by change of cost.

All considered, the Dijkstra algorithm is not of practical interest, due to its high time of computation for an optimal path, the A* algorithm is very useful in case of static and completely known environment and the D* algorithm is the most efficient among the three when the environment is partially known and dynamic.

Artificial Intelligence Algorithms

The AI algorithms takes inspiration from the natural and biological world. They have been employed in several autonomous navigation systems, as stated in [20]. In the following part, there will be described the main AI algorithms, without going deep into their implementation details, but providing a general idea of the working principles for completeness:

- Genetic Algorithm (GA): it was first proposed by Holland in 1975. As written in [20] "In the GA, all the possible solutions of the problem are encoded to chromosomes, and all the chromosomes form an initial population". Then, applying iteratively the biological operation of crossover, mutation and selection on the chromosomes, it is possible to compute the right solution. The main advantages of the GA algorithm are: the simplicity, the robustness, high search capabilities and high search efficiency. In contrast, the algorithm is prone to premature convergence.

- Ant Colony Optimization algorithm (ACO): it was inspired by the ants that look for food and was proposed by Marco Dorigo in 1992. The basic principle of the ACO is each ant will release pheromone on the path it walk through and will perceive the pheromone released by other ants. When the level of pheromone on a path is higher with respect to other ones, the ant colony will spontaneously move, avoiding obstacles, to this path releasing more and more pheromone during the movement, so that the latter ants will be attracted from the higher and higher level of pheromone. After a period of time, the ant colony will be concentrated on the shortest path, that is the optimal path. The shortest path can be easily found whether there are enough ants in the nest. The whole process can be straightforwardly compared to the path planning problem in autonomous vehicles.
- Particle Swarm Optimization (PSO): it was first proposed by Eberhart and Kennedy in 1995 and was based on the regularity of the bird cluster activity. Starting from a random solution, it finds the optimal one through iteration. At each step, it evaluates the quality of the solution through a suitable chosen fitness value, compares the currently searched optimal value and finds the global optimal solution. "This algorithm is used to solve the robotic path planning with the advantages of easy implementation, high precision and fast convergence", as stated in [20].

3.2.5 Local Path Planning and Obstacle Avoidance

The local path planning problem is strictly related to the obstacle avoidance one. A well-designed local path planner should able to plan a collision-free local path taking into account dynamic information, coming from sensors, and dynamical constraints of the specific autonomous vehicle. The planned path is local, because the planner exploits only the sensed information coming from the immediate surrounding environment, rather than a pre-built static map (as the global path planning does). As stated in [20], "The path search algorithm for the local path planning can be divided into five categories: artificial potential field method, behavior decomposition method, cased-based Learning method, rolling windows algorithm, and artificial intelligence algorithm". They can be summed up as follows:

- Artificial Potential Field (APF) method: it is based on the same principle of Potential Fields previously described in 3.2.3.
- Behavior Decomposition method: "it is a relatively new trend in path planning for mobile robots"([20]), that deals with breaking down the navigation problem into a set of relatively independent navigation unit, called behavioral primitives, such as collision avoidance, tracking, target guidance etc. Then, these units coordinate with each other to accomplish the overall navigation tasks.

- Cased-based Learning method: First, the autonomous vehicle needs to build a proper case database before path planning. Then, when it has to solve a new problem, it will search the needed information from the previous established database and, based on the results, it will make a comparison and an analysis to find the solution, that is most similar to the encountered new problem.
- Rolling Windows algorithm: The path planning problem is solved by recursively calculating a suitable window, based on the local environment information. Then, in the current rolling window, the real-time planning is implemented using sub-targets. Such sub-targets are computed by heuristic method and are updated each time the rolling window moves, until the planning task is completed.
- Artificial Intelligence algorithms: they have been previously described in 3.2.4.

Among all the existing local path search algorithms, there will be described the dynamic window approach (DWA), because it is one of the most used local planners in ROS for what concern the autonomous navigation of UGV.
In addition, local path planning can be solved by recursively applying some of the previous described path planning algorithms such as A*, D* etc. However, this approach can be very slow and computationally expensive, as a consequence it may not be suitable for certain real-time applications.

Dynamic Window Approach

It is a collision avoidance algorithm developed by Dieter Fox, Wolfram Burgard and Sebastian Thrun in 1997 ([22]). This approach allows "to incorporate the dynamics of the robot"([22]), "by reducing the search space to the dynamic window , which consists of the velocities reachable within a short time interval"([22]). Then, "within the dynamic window it only considers admissible velocities yielding a trajectory on which the robot is able to stop safely", as written in [22]. "Among all the velocities a combination of translational and rotational velocity is chosen by maximizing an objective function"([22]), which consist of "a measure of progress towards a goal location, the forward velocity of the robot, and the distance to the next obstacle on the trajectory", as stated in [22].
The different steps, employed by the algorithm, can be summed up as follows:

1. Search Space: exploiting a three step procedure, it is possible to reduce the search space:
    - Circular trajectories: as stated in [22], "the dynamic window approach takes into account only circular trajectories, called curvatures, uniquely determined by pairs (v, ω) of translational and rotational velocities. This results in a two-dimensional velocity search space".

- Admissible velocities: with this restriction, "only safe trajectories are considered"([22]). " A pair of (v, ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature"([22]).
- Dynamic window: "The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the dynamical constraints (e.g. limited accelerations) of the robot", as stated in [22].

2. Optimization:

$$G(v, \omega) = \sigma(\alpha \cdot angle(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega)) \tag{3.2}$$

$G(v, \omega)$ is the objective function to be maximized and, "with respect to the current position and orientation of the robot, it trades off the following aspects"([22]):

- Target heading: as written in [22], "*angle* is a measure of progress towards the goal location. It is maximal if the robot moves directly towards the target."
- Clearance: "*dist* is the distance to the closest obstacle on the trajectory. The smaller the distance to an obstacle the higher is the robot's desire to move around it"([22]).
- Velocity: "*vel* is the forward velocity of the robot and supports fast movements", as stated in [22].

σ is a smoothness function.

The trajectories of an autonomous vehicle (e.g. UGV) are approximated by a sequence of circular arcs (circular trajectories), that can be computed by exploiting the motion model of the vehicle under study.
The set of admissible velocities can be represented as follows:

$$V_a = \{(v, \omega) | v \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{v}_b} \wedge \omega \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{\omega}_b}\}$$

where: $\dot{v}_b$ and $\dot{\omega}_b$ are the acceleration required for breakage before colliding.
The dynamic window can be represented by the following set:

$$V_d = \{(v, \omega) | v \epsilon [v_a - \dot{v} \cdot t, v_a + \dot{v} \cdot t] \wedge \omega \epsilon [\omega_a - \dot{\omega} \cdot t, \omega_a + \dot{\omega} \cdot t]\}$$

where: t is the considered time interval, $\dot{v}$ and $\dot{\omega}$ are the accelerations, that will be used during the time interval t and (v_a, ω_a) is the actual velocity. The main drawback of such algorithm is the possibility of getting trapped in local minima

"in the case where no admissible trajectory allows the autonomous vehicle to translate"([23]). "In such situations, the robot rotates away from the obstacle until it is able to translate again", as written in [23]. Eventually, the DWA "only assumes geometric information about the relative location of obstacles. As a consequence, it is well-suited for the usage of ultrasonic transducers, laser-range finders or even infrared sensors and cameras for the detection of obstacles"([23]).

3.3 Global Positioning System

"The Global Positioning System (GPS), originally NAVSTAR GPS,is a satellite-based radionavigation system owned by the United States government and operated by the United States Space Force", as stated in [24]. "The GPS project was started by the U.S. Department of Defense in 1973"([24]) and twenty years later "the full constellation of 24 satellites"([24]) was operative. "Originally, the GPS usage was limited to the United States military, then, from 1980s, civilian use was allowed following an executive order from President Ronald Reagan", as written in [24]. "The GPS service is provided by the United States government, which can selectively deny access to the system, or degrade the service at any time"([24]). "As a result, several countries have developed or are in the process of setting up other global or regional satellite navigation systems"([24]), such as:

- Russia: "The Russian Global Navigation Satellite System (GLONASS) was developed contemporaneously with GPS, but suffered from incomplete coverage of the globe until the mid-2000s"([24]).

- China: "The BeiDou Navigation Satellite System began global services in 2018, with full deployment scheduled for 2020", as stated in [24].

- Europe: "The Global Navigation Satellite System (GNSS) Galileo was created by the European Union through the European GNSS Agency (GSA) and went live in 2016"([25]).

- India: "The Indian Regional Navigation Satellite System (IRNSS), with an operational name of Navigation with Indian Constellation (NavIC) is an autonomous regional satellite navigation system. It covers India and a region extending 1,500 km around it, with plans for further extension", as stated in [26].

- Japan: "The Quasi-Zenith Satellite System (QZSS) is a GNSS satellite-based augmentation system to enhance GNSS's accuracy in Asia-Oceania, with satellite navigation independent of GPS scheduled for 2023"([24]).

3.3.1 GPS structure

The actual GPS structure is organized in three segments:

- Space Segment (SS): "It is composed of 24 to 32 satellites, also called Space Vehicles (SV)"([24]), in medium Earth orbit[5], that is the space region around the Earth between 2.000 km and 35.786 km above the sea level. The Space Vehicles are organised in "six orbital planes with four satellites each"([24]). "The orbits are arranged so that at least six satellites are always within line of sight from everywhere on the Earth's surface", as stated in [24]. "The additional satellites over 24 improve the precision of GPS receiver calculations by providing redundant measurements"([24]). "Indeed with the expanded constellation, nine satellites are usually visible from any point on the ground at any one time, ensuring considerable redundancy"([24]).

- Control Segment (CS):"It is composed of a master control station (MCS), an alternative master control station, four dedicated ground antennas and six dedicated monitor stations", as written in [24]. "The flight paths of the satellites are tracked by dedicated U.S. Space Force monitoring stations"([24]), along with shared National Geospatial-Intelligence Agency (NGA) monitor stations spread around the world. "The tracking information is sent to the MCS at Schriever Air Force Base 25 km ESE of Colorado Springs, which is operated by the 2nd Space Operations Squadron (2 SOPS) of the U.S. Space Force. Then 2 SOPS contacts each GPS satellite regularly with a navigational update using dedicated or shared ground antennas", as stated in [24]. The update procedure is intended for updating some internal parameters of each satellites.

- User Segment (US): "It is composed of a huge number of U.S. and allied military users of the secure GPS Precise Positioning Service (PPS), and tens of millions of civil, commercial and scientific users of the Standard Positioning Service (SPS)"([24]). "In general, GPS receivers are composed of an antenna, tuned to the frequencies transmitted by the satellites, receiver-processors, and a highly stable clock (often a crystal oscillator) and they may also include a display for providing location and speed information to the user"([24]). The SPS can only receive the L1 frequency (1.57542 GHz) and the Coarse/Acquisition (C/A) code due to military security issues.

[5]https://en.wikipedia.org/wiki/Medium_Earth_orbit

3.3.2 How does it work?

Required Concepts

"The GPS is based on time and the known position of GPS specialized satellites, that carry very stable atomic clocks synchronized with one another and with the ground clocks", as written in [24]. "Any drift from time maintained on the ground is corrected daily and, in the same manner, the satellite locations are known with great precision"([24]). "GPS receivers have clocks as well, but they are less stable and less precise"([24]), thus they cannot be synchronized with the atomic clocks of the GPS satellites.
"Each GPS satellite continuously transmits a radio signal containing the current time and data about its position. Since the speed of radio waves is constant and independent of the satellite speed, the time delay between the transmission and the reception of the signal, it is proportional to the distance satellite-receiver. A GPS receiver monitors multiple satellites and solves equations to determine the precise position of the receiver and its deviation from true time", as stated in [24]. "At a minimum, four satellites must be in view of the receiver"([24]) in order to allow it to compute the three unknown position coordinates and the unknown clock deviation from satellite time.
The transmitted radio signal is a carrier wave with modulation, that includes:

- "A sequence of ones and zeros, called pseudorandom code, that is known to the receiver. By time-aligning a receiver-generated version and the receiver-measured version of the code, the time of arrival (TOA) of a defined point in the code sequence, called an epoch, can be found in the receiver clock time scale", as written in [24].

- "A message that contains the time of transmission (TOT) of the code epoch (in GPS time scale) and the satellite position at that time"([24]).

"Each navigation message is transmitted on L1 (1575.42 MHz for public use) and L2 (1227.60 MHz) frequencies at a rate of 50 bits per second and takes 750 seconds (twelve and half minutes) to be completely transmitted"([24]). Since "all satellites broadcast messages at the same frequencies, they use unique code division multiple access (CDMA) to encode signals, so receivers can distinguish individual satellites from each other"([24]). "The whole system uses two distinct CDMA encoding types: the coarse/acquisition (C/A) code, which is accessible by the general public, and the precise (P(Y)) code, which is encrypted so that only the U.S. military and other NATO nations who have been given access to the encryption code can use it", as stated in [24].
For public use, "all the satellite signals are modulated onto the same L1 carrier frequency, as a consequence the signals must be separated after demodulation.

Each satellite has been assigned a unique binary sequence, known as a Gold code, that is used by the receiver to correctly decode the signals relative to the satellites monitored by the receiver"([24]).

Basic Procedure

First of all, "the receiver computes the TOAs, using the received signals from four or more satellites. Then, exploiting the TOAs and the TOTs, the receiver creates the time of flight (TOF) values, also called pseudo-ranges, which are approximately equivalent to receiver-satellite distances (or ranges) plus time difference between the receiver and GPS satellites multiplied by speed of light", as written in [24]. Finally, the receiver computes (simultaneously) its three-dimensional position and clock deviation processing the pre-computed TOFs, by means of the navigation equations. The receiver position is relative to a three dimensional Cartesian coordinates with origin at the Earth's center and can be converted to latitude, longitude and height using an ellipsoidal Earth model. Moreover, "the height may be transformed to an altitude relative to the geoid"([24]).

Navigation Equations

(x_i, y_i, z_i) denotes the three-dimensional position of the i^{th} satellite.
s_i denotes the time of the i^{th} satellite.
t_{ri} is the time of message (sent by the i^{th} satellite) reception indicated by the on-board receiver clock.
b "is the receiver's clock bias from the much more accurate GPS clocks employed by the satellites"([24]).
n is the number of the visible satellites by the receiver.
$t_i = t_{ri} - b$ is the true reception time of message sent by the i^{th} satellite
"Assuming the messages travel at the speed of light"([24]), denoted as c, the traveled distance is : $d_i = (t_{ri} - b - s_i)\, c$, for i = 1,2, ..., n, that can be also written as $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (y - y_i)^2}$ where (x, y, z) are the three-dimensional components of the receiver's position.
$p_i = (t_{ri} - s_i)c$ are the pseudoranges (a biased version of the true distance), that can be also written as: $p_i = d_i + bc$
To suitably solve the above equations in order to find the four unknowns (x, y, z, b), "signals from at least four different satellites should be available. They can be solved by algebraic or numerical methods"([24]).

Error sources

The analysis of the errors, that affect the GPS is important to understand how much bigger is their magnitude and how they can be mitigated. In GPS applications,

the errors are usually referred as user equivalent range errors (UERE). The main error sources are:

- Signal arrival time measurement: "To measure the delay (difference between time of arrival and time of transmission of a message), the receiver compares the bit sequence received from the satellite with an internally generated version. By comparing the rising and falling edges of the bit transitions, modern electronics can measure signal offset to within about one percent of a bit pulse width. The GPS signals propagate at the speed of light and the C/A code, for civilian use, transmits data at 1.023 million pulses per second"([27]). From the previous considerations an approximate error range can be computed as

$$\Delta_{signal} = \frac{0.01 * c}{1.023 * 10^6} \simeq \pm 3m$$

- Atmospheric effects: "The speed of the GPS signals may change due to inconsistencies of atmospheric conditions, indeed they pass through the Earth's atmosphere, especially the ionosphere. The effects of the ionosphere on a microwave signal depend on the its frequency, generally change slowly, and can be averaged over time", as stated in [27]. Humidity and atmospheric pressure occur in the troposphere and may also affect the signal reception delay. On one hand, "the humidity effect is similar to that caused by the ionosphere, but it changes more quickly and is not frequency dependent, as a consequence it is more difficult to compensate humidity errors than ionospheric effects"([27]). On the other hand, "the atmospheric pressure effect is caused by the dry gases present at the troposphere and it varies with local temperature and atmospheric pressure in quite a predictable manner using the laws of the ideal gases"([27]). A possible estimate of the error ranges, caused by the ionospheric and the tropospheric effects, are:

$$\Delta_{ionospheric} \simeq \pm 5m$$

$$\Delta_{tropospheric} \simeq \pm 0.5m$$

- Multipath effects: "The radio signals, sent by GPS satellites, reflect off surrounding terrain, buildings, canyon walls, hard ground, etc. These delayed signals cause measurement errors that are different for each type of GPS signal due to its dependency on the wavelength"([27]). A possible estimate of the error range, due to the multipath effect, is:

$$\Delta_{multipath} \simeq \pm 1m$$

- Ephemeris and clock errors: "The ephemeris[6] (it gives the trajectory of artificial satellites in the sky) data is transmitted every 30 seconds, while the information itself may be up to two hours old. Variability in solar radiation pressure has an indirect effect on GPS accuracy due to its effect on ephemeris errors", as written in [27]. "The satellites' atomic clocks experience noise and clock drift errors. Although the navigation message contains corrections for these errors and estimates of the accuracy of the atomic clock, they are based on observations and may not indicate the clock's current state"([27]). A possible estimates of the ephemeris and clock error ranges are:

$$\Delta_{ephemeris} \simeq \pm 2.5m$$

$$\Delta_{clock} \simeq \pm 2m$$

- Numerical error: The receiver's position computation process has a numerical error with the following estimated value of its standard deviation:

$$\sigma_{numerical} \simeq 1m$$

Taking into account, all the previous described sources of error, it is possible to compute the standard deviation of the user equivalent range errors, related to the public use of the GPS (C/A code), as follows:

$$3*\sigma_R = \sqrt{\Delta^2_{signal} + \Delta^2_{ionospheric} + \Delta^2_{tropospheric} + \Delta^2_{multipath} + \Delta^2_{ephemeris} + \Delta^2_{clock}} \simeq 6.7m$$

While the standard deviation of the error in receiver position can be computed as:

$$\sigma_{rc} = \sqrt{PDOP^2 * \sigma^2_R + \sigma^2_{numerical}}$$

It depends on the Position Dilution Of Precision[7] (PDOP), that is a term used in satellite navigation to describe error caused by the relative position of the GPS satellites.

3.3.3 Standard GPS

The standard GPS is based on the computation of the GPS receiver's position, applying one of the following analytical methods to the navigation equations:

[6]https://en.wikipedia.org/wiki/Ephemeris

[7]https://gisgeography.com/gps-accuracy-hdop-pdop-gdop-multipath/

- Least squares: When the visible satellites are more than four, the system of equations is over-determined, thus it has not a unique solution. In this case, it is possible to apply the least squares method, that requires to solve the following optimization problem:

$$(\hat{x}, \hat{y}, \hat{z}, \hat{b}) = \min_{(x,y,z,b)} \sum_i (\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + bc - p_i)^2 \quad (3.3)$$

- Iterative: The equations to be solved are non linear and required special methods to be solved. "A common approach is by iteration on a linearized form of the equations, for example the Gauss–Newton algorithm can be used",([24]).

- Closed-form: "S. Bancroft developed a closed-form solution to the above set of equations", as stated in [24]. The key step to find a solution is either the inversion of a 4x4 matrix in the case of four visible satellites or the use of a pseudoinverse when more than four satellites are visible. "It is an algebraic method that provides one or two solutions for the unknown quantities. When there are two (usually the case), only one is a near-Earth sensible solution", as written in [24].

To better understand how the receiver is able to compute its position exploiting GPS signals coming from the satellites, it may be worth to consider the following geometric interpretations of the solution methods:

- Spheres: "The pseudoranges, contain clock errors, but in a simplified idealization in which the satellites and receiver are synchronized, the pseudoranges can be considered as true ranges, that represent the radii of spheres, each centered on one of the transmitting satellites. The solution for the position of the receiver is then at the intersection of the surfaces of these spheres. In this ideal case, when the visible satellites are three, the spheres intersect exactly in two points; one point is the location of the receiver, and the other moves rapidly in successive measurements and would not usually be on Earth's surface"([24]). However, the visible satellites usually are more than three and the corresponding spheres will intersect in more than two points, so an approximate unique intersection should be computed, typically via least-squares.

- Hyperboloids: "If the pseudorange between the receiver and satellite i and the pseudorange between the receiver and satellite j are subtracted, $p_i - p_j$, the common receiver clock bias (b) cancels out, resulting in a difference of distances $d_i - d_j$. The locus of points having a constant difference in distance to two points (here, two satellites) is a hyperbola on a plane and a hyperboloid of revolution in 3D space. Thus, from four pseudorange measurements, the

receiver can be placed at the intersection of the surfaces of three hyperboloids each with foci at a pair of satellites. In the case of additional satellites, the multiple intersections are not necessarily unique, and a best-fitting solution is sought instead", as stated in [24].

- Spherical cones: "The clock on-board the receiver is usually not of the same quality as the ones used in the satellites and will not be accurately synchronised to them. This produces large errors in the computed distances to the satellites. Therefore, in practice, the time difference between the receiver clock and the satellite time is defined as an unknown clock bias b. The equations are then solved simultaneously for the receiver position and the clock bias. The solution space (x, y, z, b) can be seen as a four-dimensional geometric space. In that case each of the equations describes a spherical cone with the cusp located at the satellite, and the base a sphere around the satellite. The receiver is at the intersection of four or more of such cones"([24]).

The modern GPS receivers,that does not employ any technique to compensate the errors (previously described), usually have an accuracy of about 5 meters horizontally.

3.3.4 Differential GPS

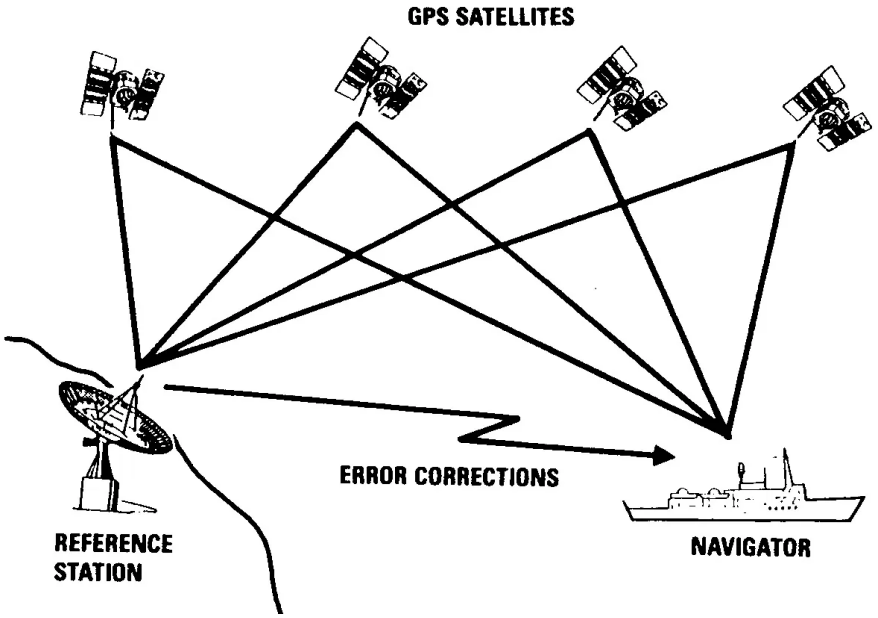


Figure 3.12: An example of DGPS working principle

Figure 3.13: Example of a ground/reference station

The GNSS should employ some strategies to attenuate or, possibly, cancel out most of the errors coming from satellites signals transmission, in order to improve the accuracy of the position computation in GPS receivers. An available approach is the so called Differential GPS (DGPS), a figured example is illustrated in Fig. 3.12. It is based on some GPS ground (or reference) stations (shown in Fig. 3.13), scattered around the world, which position is computed with very high precision, through geodetic techniques and additional high precision tools. The main function of the ground stations is to compute the error between the measured pseudorange (satellite-receiver distance) and the real pseudorange (internally computed, thanks to its known position with high precision). Then, there are mainly two ways[8] to broadcast the correction message to GPS receivers:

- Satellite-Based Augmentation System(SBAS): SBAS is sometimes synonymous with wide-area Differential GPS (WADGPS). The ground stations send correction messagges to some geostationary[9] satellites (they are at a specific height above the Earth's equator), so that they can broadcast the correction signals to one or more GPS receivers on the Earth.

- Ground-Based Augmentation System (GBAS) and Ground-Based Regional

[8]https://it.wikipedia.org/wiki/GNSS_augmentation

[9]https://it.wikipedia.org/wiki/Orbita_geostazionaria

Augmentation System (GRAS): GBAS system works on limited local area (about 23 km, i.e. airports), while GRAS system can cover wider regional area. The ground stations can be equipped by a radio transmitter and directly send the correction messages to the GPS receivers. This approach has a main drawback: the accuracy of the correction messages decreases proportionally with ground station-GPS receivers distance. This effect can be caused by either the ground station and GPS receiver do not receive the GPS signals from the same satellites or they are too far from each other on the Earth's surface, so that the correction cannot be considered adequate.

Both the SBAS[10] approach and GBAS[11] approach can increase the accuracy with position errors below 1 meter, but the latter degrades its accuracy with ground station-receiver distance increasing.

3.3.5 RTK-GPS

"Real Time Kinematic (RTK) is an additional satellite navigation technique used to enhance the precision of position data derived from satellite-based positioning systems, such as GPS", as stated in [28]. In particular, when it is applied to the GPS, it is commonly known as carrier phase enhancement, or carrier phase GPS (CPGPS), or RTK-GPS. This system is similar to DGPS, because it exploits ground/reference stations at known locations to communicate with GPS receivers. However, the RTK-GPS is based on a different approach to compute the error corrections, that can be summed up as follows: the main involved entities are a reference station and a mobile unit (rover)."RTK uses the satellite signal's carrier wave as its signal, ignoring the information contained within"([28]). In addition, "the range to a satellite is essentially calculated by multiplying the carrier wavelength times the number of whole cycles between the satellite and the rover and adding the phase difference", as stated in [28]. "Determining the number of cycles is non-trivial, since signals may be shifted in phase by one or more cycles, as a consequence the error is equal to the error in the estimated number of cycles times the wavelength, which is 19 cm for the L1 signal (civilian use). Solving this problem, called integer ambiguity search problem, results in centimeter precision" ([28]). "The error can be reduced with sophisticated statistical methods that compare the measurements from the C/A signals and by comparing the resulting ranges between multiple satellites", as written in [28]. "In practice, RTK systems use a single base-station that re-broadcasts (through radio signal) the phase of the observed carrier wave to a number of mobile units, that compare their own phase measurements with

[10]https://gssc.esa.int/navipedia/index.php/SBAS_Fundamentals

[11]https://gssc.esa.int/navipedia/index.php/GBAS_Fundamentals

the one received from the base station"([28]). "This allows the units to calculate their relative position to within millimeters, although their absolute position is accurate only to the same accuracy as the computed position of the base station. The typical nominal accuracy for these systems is 1 centimetre horizontally and 2 centimetres vertically"([28]). The RTK technique provides accuracy enhancements up to about 20 km from the reference station.

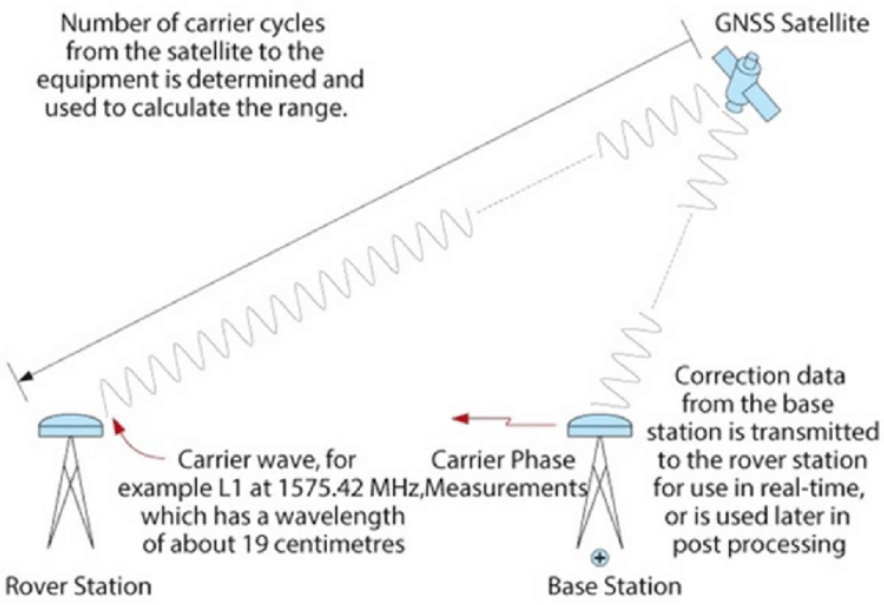


Figure 3.14: RTK working principle

3.4 Sensor Fusion for localization

Sensor fusion is a way of simultaneously taking into account different sources of information to achieve the highest accuracy of a measure. In particular, it can be used to properly localize an autonomous vehicle, exploiting several data coming from different sensors.
Among the different approaches to estimate a particular quantity (in this case the vehicle's pose), there will be described the Kalman Filter, the Extended Kalman Filter (EKF), the Unescent Kalman Filter and the Particle Filter, that are based on the Bayesian filtering approach.

3.4.1 System representation

Each physical system can be associated to a mathematical model, that describes the evolution over time of some variable of interests, such as position, linear velocity, angular acceleration etc. There exist linear or non-linear, time-invariant or time-variant, continuous or discrete system, that can be suitably approximated by a mathematical model. For instance, a continuous, linear, time variant system can be represented by the following state-space model:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + v1(t) \tag{3.4}$$
$$z(t) = C(t)x(t) + v2(t) \tag{3.5}$$

where: $x(t)$ is the state vector, that contains the variables of interest, $A(t)$ is the state matrix, $B(t)$ is the input matrix, $u(t)$ is the input vector, $v1(t)$ is the process noise, $C(t)$ is the output matrix, $z(t)$ is the output (or measurement) vector and $v2(t)$ is the measurements noise. The same continuous model can be described in discrete-time as follows:

$$x(k+1) = A(k)x(k) + B(k)u(k) + v1(k) \tag{3.6}$$
$$z(k) = C(k)x(k) + v2(k) \tag{3.7}$$

System models as well as sensor measurements are affected by uncertainties, since they cannot perfectly match the real physical system. Among different approaches of representing uncertainties, the probabilistic one is the most common used for data fusion.

3.4.2 Bayes Filter

It is one of the most used filters in estimation problems with probabilistic framework. It strongly relies on the well-known Bayes' rule:

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)} \tag{3.8}$$

where, in the particular case of localization, x can be the vehicle's pose to be estimated, z is a sensor measurement and $P(x|z)$ is the conditional probability of x given z. In case of different sensors data the Bayes' rule can be rewritten as follows:

$$P(x_k|z_{1:k}) = \frac{P(z_{1:k}|x_k)P(x_k)}{P(z_{1:k})} = \frac{P(z_1, z_2, ..., z_k|x_k)P(x_k)}{P(z_1, z_2, ..., z_k)} \tag{3.9}$$

where $z_{1:k}$ contains all the available sensors information up to the discrete-time k and x_k is the state vector at discrete-time k. In practice, it is very hard to use

the equation 3.9 as it is, because it requires to know the common and conditional distribution of probability $P(z_1, z_2, ..., z_k|x_k)$. However, assuming that each sensor information z_i, given a particular state x_k, is independent from each other, it is possible to write:

$$P(z_i|x_k, z_1, ..., z_{i-1}, z_{i+1}, ..., z_k) = P(z_i|x_k) \tag{3.10}$$

$$P(z_1, z_2, ..., z_k|x_k) = P(z_1|x_k)...P(z_k|x_k) = \prod_i P(z_i|x_k) \tag{3.11}$$

Substituting the eq. 3.11 in eq. 3.9 a more practical application of Bayes' rule, called Bayes filter, is obtained:

$$P(x_k|z_{1:k}) = \frac{P(z_{1:k}|x_k)P(x_k)}{P(z_{1:k})} = \frac{\prod_i P(z_i|x_k)P(x_k)}{P(z_1, z_2, ..., z_k)} \tag{3.12}$$

where $P(z_1, z_2, ..., z_k) \neq \prod_i P(z_i)$ because each sensor data z_i has in common a particular state of the system under study, as a consequence each $P(z_i)$ depends on each other.

Bayesian recursive filter

The equation 3.12 requires to store all the past information to compute the probability of a particular state and as a new information arrives the probability should be recomputed. To overcome this issue, it is possible to implement the Bayesian filter in a recursive way. Assuming that, as a new sensor data is available, $z_{1:k} = \{z_k, z_{1:k-1}\}$ and each sensor measurement is independent from each other, given a particular state, the recursive implementation can be derived as follows:

$$\begin{aligned} P(x_k, z_{1:k}) &= P(x_k|z_{1:k})P(z_{1:k}) \\ &= P(z_k, z_{1:k-1}|x_k)P(x_k) \\ &= P(z_k|x_k)P(z_{1:k-1}|x_k)P(x_k) \end{aligned}$$

that can be rewritten as:

$$\begin{aligned} P(x_k|z_{1:k})P(z_{1:k}) &= P(z_k|x_k)P(z_{1:k-1}|x_k)P(x_k) \\ &= P(z_k|x_k)P(x_k|z_{1:k-1})P(z_{1:k-1}) \end{aligned}$$

from which the final form of the Bayesian recursive filter can be obtained considering that $P(z_{1:k})/P(z_{1:k-1}) = P(z_k|z_{1:k-1})$:

$$P(x_k|z_{1:k}) = \frac{P(z_k|x_k)P(x_k|z_{1:k-1})}{P(z_k|z_{1:k-1})} \tag{3.13}$$

where $P(x_k|z_{1:k-1}) = \int P(x_k|x_{k-1})P(x_{k-1}|z_{1:k-1})dx_{k-1}$. $P(x_k|x_{k-1})$ is computed using the dynamical model related to the physical system under study. The

advantage of the recursive form is to store and update only $P(x_k|z_{1:k})$, as new sensors data are available. Both the Bayes filter and its recursive form have been described not taking into account any input u, however, it is possible to include it and derive the corresponding filters in a straightforwardly way.

3.4.3 Kalman Filter

Kalman filter is a linear recursive estimator, based on the following assumptions:

- The dynamical system can be described by a linear mathematical model as 3.4.
- The process and the measurement noises are both white and Gaussian, that means, $v1(t) \sim WN(0, V1(t))$ and $v2(t) \sim WN(0, V2(t))$, where $V1(t)$ and $V2(t)$ are the known variance matrices.
- $v1(t)$ and $v2(t)$ are uncorrelated with each other at different time instants, but correlated if considered at the same time instant, with a covariance matrix given by $V_{12}(t)$.

The Kalman filter can be applied whenever a quantity or some quantities should be estimated, as in the case of a vehicle's pose. The estimation algorithm is divided into two steps: a prediction step and a correction step and it involves some gains, that are chosen such that the estimate $\hat{x}(t)$ minimizes:

$$L(t) = \int_{-\infty}^{\infty} (x(t) - \hat{x}(t))^T (x(t) - \hat{x}(t)) P(x(t)|Z^k) dx$$

To solve this problem, the derivative with respect to $\hat{x}(t)$ is applied on $L(t)$, then it is imposed equal zero and after some calculations:

$$\hat{x}(t) = \int_{-\infty}^{\infty} x(t) P(x(t)|Z^k) dx = E(x(t)|Z^k)$$

that is the conditional mean estimate or Bayesian estimate.
Since the Kalman filter algorithm is typically implemented on a calculator, that works with discrete values, it is worth to consider the discrete-time version of the state-space model of a physical system (3.6). Moreover, taking into account the recursive Bayesian estimation approach, the estimate of a particular state at discrete-time instant $k+1$ can be written as follows:

$$\begin{aligned} \hat{x}(k+1|k) &= E(x(k+1)|Z^k) \\ &= E(x(k+1)|z^k, Z^{k-1}) \\ &= E(x(k+1)|Z^{k-1}) + E(x(k+1)|e(k)) \end{aligned}$$

where $e(k)$ is the innovation of z^k given Z^{k-1} and defined by:

$$e(k) = z^k - E(z^k|Z^{k-1})$$

where $E(z^k|Z^{k-1}) = \hat{z}(k|k-1)$ is the estimate of the output measure at discrete-time instant k. Considering the state-space model 3.6, the previous estimate can be written as:

$$\hat{z}(k|k-1) = C(k)E(x(k)|Z^{k-1}) + E(v2(k)) = C(k)\hat{x}(k|k-1)$$

All considered, the recursive one-step state prediction can be written as:

$$\begin{aligned}\hat{x}(k+1|k) &= A(k)E(x(k)|Z^{k-1}) + B(k)u(k) + \Sigma_{x(k+1)e(k)}\Sigma^{-1}_{e(k)e(k)}e(k)\\ &= A(k)\hat{x}(k|k-1) + B(k)u(k) + K(k)e(k)\end{aligned}$$

where $K(k)$ is the one-step Kalman prediction gain matrix and involves the following equations:

$$\begin{aligned}\Sigma_{x(k+1)e(k)} &= A(k)P(k)C(k)^T + V_{12}(k)\\ \Sigma_{e(k)e(k)} &= C(k)P(k)C(k)^T + V2(k)\\ K(k) &= \Sigma_{x(k+1)e(k)}\Sigma^{-1}_{e(k)e(k)}\\ &= (A(k)P(k)C(k)^T + V12(k))(C(k)P(k)C(k)^T + V2(k))^{-1}\\ P(k+1) &= A(k)P(k)A(k)^T + V1(k) - K(k)(C(k)P(k)C(k)^T + V2(k))K(k)^T\end{aligned}$$

$P(k)$ is the prediction error variance matrix of the state $x(k)$ and is computed using the Difference Riccati Equation (DRE).
Starting from the state prediction is possible to estimate the current state $x(k)$ given the current measurements, as follows:

$$\begin{aligned}\hat{x}(k|k) &= E(x(k)|Z^k) = E(x(k)|Z^{k-1}, z^k)\\ &= E(x(k)|Z^{k-1}) + E(x(k)|e(k)) = \hat{x}(k|k-1) + E(x(k)|e(k))\\ &= \hat{x}(k|k-1) + \Sigma_{x(k)e(k)}\Sigma^{-1}_{e(k)e(k)}e(k) = \hat{x}(k|k-1) + K_0(k)e(k)\end{aligned}$$

where $K_0(k)$ is the Kalman filter gain matrix and involves the following equations:

$$\begin{aligned}\Sigma_{x(k)e(k)} &= P(k)C(k)^T\\ \Sigma_{e(k)e(k)} &= C(k)P(k)C(k)^T + V2(k)\\ K_0(k) &= \Sigma_{x(k)e(k)}\Sigma^{-1}_{e(k)e(k)}\\ &= P(k)C(k)^T(C(k)P(k)C(k)^T + V2(k))^{-1}\end{aligned}$$

All considered, the Kalman filter is based on the following equations:

$$\hat{x}(k+1|k) = A(k)\hat{x}(k|k-1) + B(k)u(k) + K(k)e(k) \tag{3.14}$$
$$\hat{z}(k|k-1) = C(k)\hat{x}(k|k-1) \tag{3.15}$$
$$\hat{x}(k|k) = \hat{x}(k|k-1) + K_0(k)e(k) \tag{3.16}$$
$$e(k) = z(k) - z(k|k-1) \tag{3.17}$$

that, can be implemented recursively on a calculator and are represented in a schematic way in Fig. 3.15.

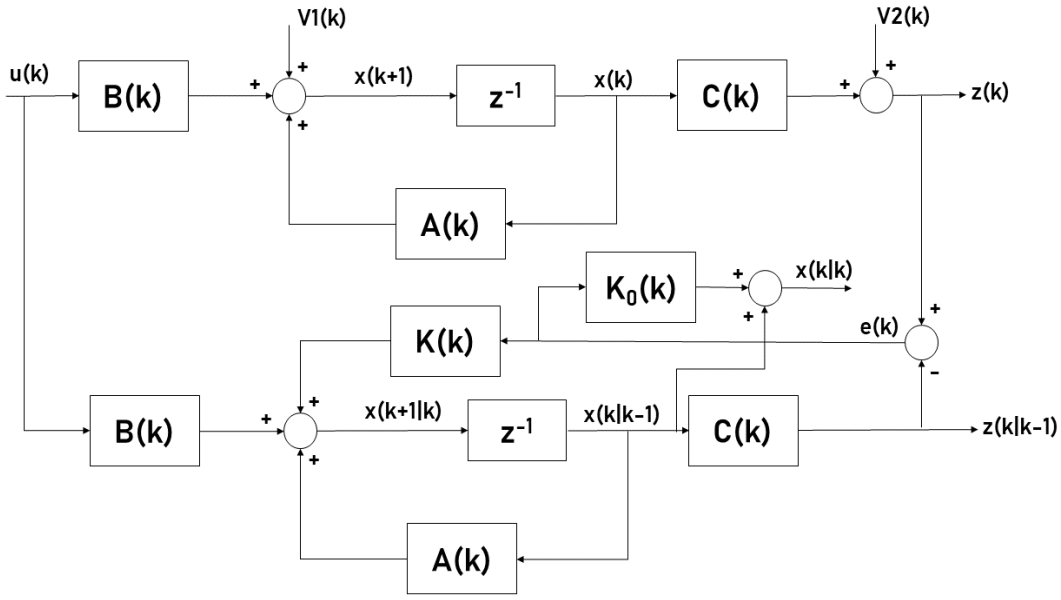


Figure 3.15: Block scheme diagram of Kalman filter

3.4.4 Extended Kalman Filter

Sometimes physical system cannot be represented by linear state-space model, especially in robotic fields. However, it is possible to model such systems with non linear relationship as follows:

$$x(k+1) = f(k, x(k), u(k)) + v1(k) \tag{3.18}$$
$$z(k) = h(k, x(k)) + v2(k) \tag{3.19}$$

that is the representation of a discrete-time, non-linear, time-variant, dynamic system. The assumptions on $v1(k)$ and $v2(k)$ are the same of those described for

the Kalman filter in 3.4.3
The Extended Kalman Filter is usually used to estimate the state of non-linear system. It exploits the same approach of the Kalman filter, but the matrices $A(k)$ and $C(k)$ are computed by linearizing the system around the last state estimate as follows:

$$\hat{A}(k|k-1) = \left.\frac{\partial f(t,x,u)}{\partial x}\right|_{t=k,u=u(k),x=x(k|k-1)} \qquad \hat{C}(k|k-1) = \left.\frac{\partial h(t,x)}{\partial x}\right|_{t=k,x=x(k|k-1)}$$

Eventually, the equations of the EKF are the following ones:

$$\hat{x}(k+1|k) = f(k,\hat{x}(k|k-1),u(k)) + \hat{K}(k)e(k) \tag{3.20}$$
$$\hat{z}(k|k-1) = h(k,\hat{x}(k|k-1)) \tag{3.21}$$
$$\hat{x}(k|k) = \hat{x}(k|k-1) + \hat{K}_0(k)e(k) \tag{3.22}$$
$$e(k) = z(k) - z(k|k-1) \tag{3.23}$$

where $\hat{K}(k)$ is the extended Kalman predictor gain matrix and $\hat{K}_0(k)$ is the extended Kalman filter gain matrix. They are computed replacing $A(k)$ with $\hat{A}(k|k-1)$ and $C(k)$ with $\hat{C}(k|k-1)$ in the previous described equation of $K(k)$ and $K_0(k)$.

3.4.5 Unscented Kalman Filter

The described Kalman filters (classical KF, EKF and UKF) are based on the assumptions of having Gaussian process and measurements noises. This assumption is usually realistic and leads to consider the state distribution as a Gaussian random variable (GRV).
The EKF propagates the state distribution (represented as a GRV) through the first-order linearization of the non-linear system and this may lead to large errors in the true posterior mean and covariance of the transformed GRV, obtaining sub-optimal performance and sometimes divergence of the filter.
The UKF aims to compensate the lack of accuracy of the EKF, approximating the state distribution as a GRV, but representing it through a set of carefully chosen sample points (deterministic sampling approach). As stated in [29], "These sample points completely capture the true mean and covariance of the GRV, and when propagated through the true nonlinear system, captures the posterior mean and covariance accurately to the 2nd order (Taylor series expansion) for any nonlinearity". Moreover, the UKF algorithm strongly relies on the unscented transformation (UT).

Unscented Transformation

It is a technique to compute the statistic of a random variable subject to a non-linear transformation. Starting from a random variable x (dimension L) with mean $\bar{x}$ and

covariance matrix P_x , that is subject to the non-linear transformation $y = f(x)$, it is possible to obtain the statistic of y forming a matrix X of $2L + 1$ sigma vectors X_i computed as follows:

$$\begin{aligned} X_0 &= \bar{x} \\ X_i &= \bar{x} + \left(\gamma\sqrt{P_x}\right)_i & i = 1, ..., L \\ X_i &= \bar{x} - \left(\gamma\sqrt{P_x}\right)_{i-L} & i = L+1, ...,2L \end{aligned}$$

where $\gamma = \sqrt{(L + \lambda)}$ and $\lambda = \alpha^2(L + k) - L$. α is a constant, that determines the spread of the sigma points around $\bar{x}$ and it usually assumes small positive values. k is a secondary scaling parameter and $(\gamma\sqrt{P_x})_i$ is the ith column of the matrix square root. Then, each sigma vector is propagated through the non-linear function as follows:

$$Y_i = f(X_i) \qquad i = 0, ...,2L$$

from which the mean $(\bar{y})$ and the covariance (P_y) of y are approximated using a weighted sample mean and covariance of posterior sigma points, represented by:

$$\bar{y} \approx \sum_{i=0}^{2L} W_i^{(m)} Y_i \qquad P_y \approx \sum_{i=0}^{2L} W_i^{(c)} (Y_i - \bar{y})(Y_i - \bar{y})^T$$

where each weights W_i is computed as follows:

$$\begin{aligned} W_0^{(m)} &= \frac{\lambda}{(L + \lambda)} & W_0^{(c)} &= \frac{\lambda}{(L + \lambda)} + (1 - \alpha^2 + \beta) \\ W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(L + \lambda)} & i &= 1, ...,2L \end{aligned}$$

β is used to include prior knowledge on the distribution on x.

Unscented Kalman Filter

The UKF is a straightforward extension of the UT, since it exploits recursively the equations defined by the unscented transformation. In the general case (when the noises are not simply additive), an augmented state should be used: $x_k^a = [x_k^T \quad v1_k^T \quad v2_k^T]^T$. However, without loss of generality, the non-linear model 3.18 will be taken into account, in order to make the UKF comparable with the EKF. In the considered model, the noises are additive, so the augmented state is not

require and the overall UKF's equations can be written as follows:
Initialization step:

$$\hat{x}_0 = E[x_0] \tag{3.24}$$
$$P_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \tag{3.25}$$

then, for each $k \in \{1, ..., \infty\}$, compute the sigma points:

$$X_{k-1} = [\hat{x}_{k-1} \quad \hat{x}_{k-1} + \gamma\sqrt{P_{k-1}} \quad \hat{x}_{k-1} - \gamma\sqrt{P_{k-1}}] \tag{3.26}$$

Time update step:

$$X_{k|k-1} = f(X_{k-1}, u_{k-1}) \tag{3.27}$$
$$\hat{x}_{k|k-1} = \sum_{i=0}^{2L} W_i^{(m)} X_{i,k|k-1} \tag{3.28}$$
$$P_{k|k-1} = \sum_{i=0}^{2L} W_i^{(c)} [X_{i,k|k-1} - \hat{x}_{k|k-1}][X_{i,k|k-1} - \hat{x}_{k|k-1}]^T + V1 \tag{3.29}$$
$$Y_{k|k-1} = h(X_{k|k-1}) \tag{3.30}$$
$$\hat{y}_{k|k-1} = \sum_{i=0}^{2L} W_i^{(m)} Y_{i,k|k-1} \tag{3.31}$$
$$\tag{3.32}$$

Measurement update step:

$$P_{y_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [Y_{i,k|k-1} - \hat{y}_{k|k-1}][Y_{i,k|k-1} - \hat{y}_{k|k-1}]^T + V2 \tag{3.33}$$
$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [X_{i,k|k-1} - \hat{x}_{k|k-1}][Y_{i,k|k-1} - \hat{y}_{k|k-1}]^T \tag{3.34}$$
$$K_k = P_{x_k y_k} P_{y_k y_k}^{-1} \tag{3.35}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - \hat{y}_{k|k-1}) \tag{3.36}$$
$$P_k = P_{k|k-1} - K_k P_{y_k y_k} K_k^T \tag{3.37}$$

As can be seen in Fig. 3.16 the state estimate provided by the UKF is more accurate than that furnished by the EKF, although the former is more computationally expensive than the latter.

3.4.6 Particle Filter

The particle filters are Bayesian filters used whenever no assumptions about the kind of noise can be made. In the particular case of localization, they are also

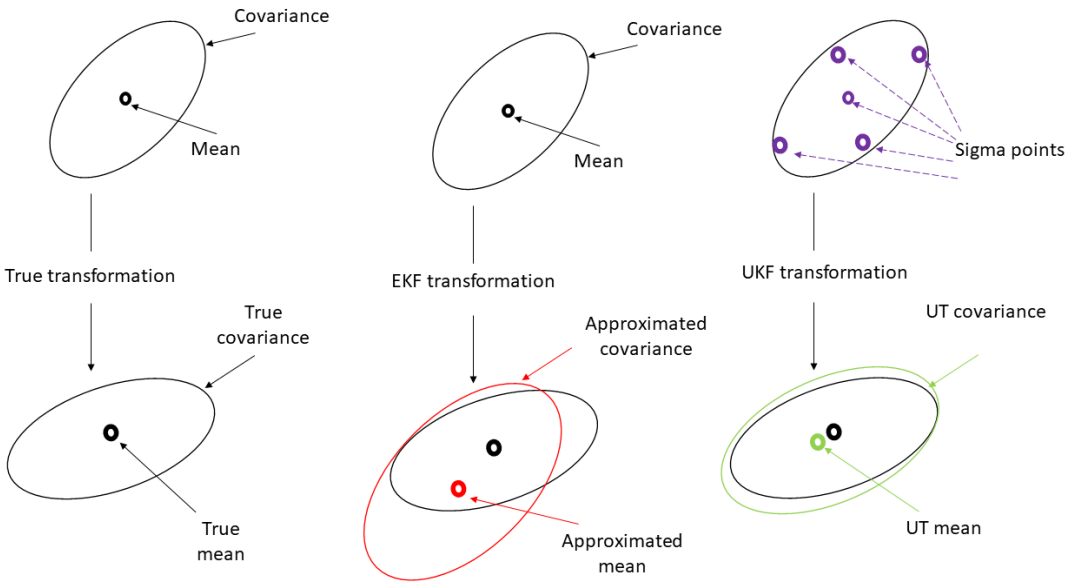


Figure 3.16: Comparison between EKF and UKF

called Monte Carlo Localization (MCL).
The MCL filter approximates the belief about the state of an autonomous vehicle (e.g. the pose) as a set of weighted particles:

$$Bel(x_k) = \{w_k^i, x_k^i\} \qquad\qquad i = 1, ..., N$$

where: x_k is the current state (to be estimated), N is the number of samples (particles), x_k^i is the ith particle associated with the ith normalized weight w_k^i at discrete-time instant k. Each particle (x_k^i) represents an hypothesis about the current state, while the related weight is a measure of the hypothesis' confidence. The normalized weights are such that: $\sum_{i=1}^{N} w_k^i = 1$.
Since particle filters are based on the Bayesian filtering approach the belief about the current state x_k can be written as:

$$Bel(x_k) = \eta_k P(z_k|x_k) \sum_{\forall N particles} P(x_k|x_{k-1}) Bel(x_{k-1}) \tag{3.38}$$

In the particular case of localization, the particle filter algorithm can be summed up as follows:

1. Motion update: propagate each particle by using the motion model of the system under study and evaluate: $\sum_{\forall N particles} P(x_k|x_{k-1}) Bel(x_{k-1})$.

2. Observation update: compute the new weights by exploiting sensors information as follows: $w_k^i = \eta^i p(z_k|x_k^i)$, where η^i is a normalization factor and can be chosen equal to w_{k-1}^i.

3. Resampling: sample a new set of weighted particles from the probability distribution generated by the particles generated at step 1. The main purpose of resampling is to select only the particles with large weights, in order to propagate only the most significant ones. In literature there exist different methods to account for this problem: multinomial resampling, residual sampling, systematic resampling, etc.

4. Compute the estimate: among the existing different approaches to estimate the state x_k, one possible solution can be: $\hat{x}_k = \sum_{i=1}^{N} w_k^i x_k^i$.

5. Time update: set $k = k + 1$ and repeat from step 1.

All considered, the main advantages of using a particle filters are:

- The ability "to model non-linear system dynamics"([30]).
- The noise can assume whatever form different from the Gaussian model.
- "In practice, it performs well in presence of a large amount of noise"([30]).
- "Simple implementation", as written in [30].

In contrast, the main drawbacks [30] of this filtering technique are:

- It is more computationally expensive compare to other filters (EKF or UKF).
- "Computational complexity grows exponentially"([30]) as the size of the state vector increases.
- "It is more likely to diverge with more accurate measurements", as stated in [30].

This kind of filter seems to work well in a noisy environment, because the noise affect in a good manner the resampling step. In presence of small amount of noise the resampling step could select the wrong particles and neglect the most significant one (the particles that are close to the real pose of the vehicle), as a consequence the filter tends to diverge.

3.5 Existing projects related to autonomous navigation for precision agriculture

Through different years, research groups, engineers, companies etc. have tried to develop several autonomous navigation system for precision agriculture, exploiting different approaches. Among all the existing projects related to autonomous navigation for precision agriculture, there will be briefly described only some of them, that deal with the autonomous navigation in vineyards, orchards and farms.

3.5.1 Autonomous navigation in orchards

Flavio Callegati, Alessandro Samorì, Roberto Tazzari, Nicola Mimmo and Lorenzo Marconi in [31], proposed an autonomous tracked UGV able to detect the beginning and the end of rows in an orchards, "to keep almost fixed lateral distance from trees"()[31] and to stop "once it has exited a row"([31]). The UGV is equipped "with GPS receiver, 3D laser scanner and an inertial measurement unit, which provide to the robot the capability of navigating and localizing itself in the agricultural environment", as stated in [31]. The overall system is mainly composed by a Human Machine Interface (HMI) based on Windows and a control system implemented via ROS. The HMI is used to check the rover position and/or send commands/mission, while ROS is used by the rover to compute an estimate of its position, send commands to the actuators and perform autonomous navigation in the rows. Eventually, the developed autonomous system has been tested in a plum field and in a vineyard and in both cases it gives good results, as stated in [31].

3.5.2 Autonomous navigation in farms

Mark A. Post, Alessandro Bianco and Xiu T. Yan in [32] proposed an autonomous UGV based on ROS for agricultural fields. The main purpose of their project "is to combine several approaches to navigation and control into one autonomous rover, which shall be able to navigate autonomously in a farm field while constructing a map of the environment", as stated in [32]. The project has been developed using cost-effective and off the shelf hardware such as: "an NVidia Jetson TK1 single-board computer (SBC), an SPI-connected Arduino board that handles fast kinematic control of the drive and steering motors, and a SparkFun Razor attitude and heading reference system (AHRS) board. Four sensors allow the perception of the environment: the Razor AHRS board containing a three degree of freedom accelerometer, angular rate sensor, and magnetometer, a Hokuyo UTM-30LX-EW scanning laser rangefinder mounted on the front of the chassis, a Stereolabs ZED stereo vision camera mounted on top of the laser and a global navigation satellite

system (GNSS) receiver mounted on top of the camera", as written in [32]. In order to achieve their goal, they have been used some already implemented ROS nodes for autonomous navigation combined with custom nodes to deal with the specific and challenging farming environment.

3.5.3 Autonomous navigation in vineyards

In [33] Pietro Astolfi, Alessandro Gabrielli, Luca Bascetta and Matteo Matteucci have proposed an autonomous navigation system for UGVs in a vineyard. The main component of the project is a rover "based on the Husky platform[12]"([33]). "It is equipped with frontal Hokuyo laser range finder, a Velodyne Puck, a common PC with WiFi module, and an IMU sensor. In addition, the rover mounts a robotic arm, a Kinova Jaco2 6 which incorporates a low range Hokuyo, and a vertical support for the GPS antenna and a Zed stereo camera", as written in [33]. The interesting characteristic of such project is the usage of standard ROS packages for indoor navigation, such as the move base package, the AMCL localization package etc. to achieve outdoor autonomous navigation and mapping. All considered, "the resulting navigation system has proved his reliability first in simulation and then in on-field validation tests, which has highlighted strengths, i.e., an accurate and robust pose estimate (odometry estimation), the ability to build navigable maps despite a rough and complex vineyard scenario, a flexible structure in which components can easily be swapped or substituted based on the task assigned to the robot", as stated in [33].

[12]http://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot

Chapter 4

Autonomous GPS-based navigation in vineyards

In the following part will be given a detailed description of the proposed solution to perform the autonomous GPS-based navigation in precision agriculture. First of all, the outdoor localization problem will be tackled and a possible solution will be described exploiting an already available ROS package. Then, the core software components for the autonomous navigation will be shown making an explicit difference to what has been exploited from ROS and what has been developed from scratch.

4.1 Outdoor Localization

Among the existing localization filters described in literature and summed up in 3.4, the EKF has been chosen as possible candidate to solve the outdoor localization problem. It exploites the non-linear dynamical model of the system under study to predict the pose of the vehicle and correct the prediction with the sensors information. In this case, the vehicle is represented by the Jackal UGV (described in A), which is related to a non-linear model (as shown in A), while the available sensors are: an IMU, a RTK capable GPS receiver and a depth camera (listed in the Table 4.1). The camera will not be used for localization, but only for obstacle avoidance purposes.

The usage of UKF has been discarded because of the need of choosing the sigma points, required for the unescented transformation (as explained in 3.4.5). This task may be complex and hard, although the UKF is more precise compared to EKF.

The particle filter, in particular the MCL, has not been used because of its growing complexity with the increase of the state vector. Moreover, as written before in

Table 4.1: Available sensors

Sensor	**Model**
IMU	Provided with the Jackal. It contains a gyroscope, an accelerometer and a magnetometer.
GPS receiver	Novatel FlexPak6 (deeply described in B.1)
Depth camera	Intel RealSense D435i (described in details in B.2)

3.4.6, it works quite well in presence of noisy sensor measurements, but, in this case, the GPS receiver with RTK corrections is supposed to be quite accurate, although the IMU may produce noisy measurements due to the uneven terrain. All considered, the EKF has been chosen as localization filter for the autonomous navigation system, thank to its support for the non-linear systems, its simple implementation and the quite accurate pose estimation, without choosing neither sigma points (as in the UKF) nor the number of particles (as in the MCL).

4.1.1 Robot localization ROS package

ROS offers a ready-to-use package for localization, called *robot_localization*. It mainly contains two state estimation nodes, the *ekf_localization_node* and the *ukf_localization_node*. Moreover, it allows the integration of GPS data into the estimation nodes through the *navsat_transform_node*. Both the *ekf_localization_node* and the *ukf_localization_node* shares common features, such as:

- "Fusion of an arbitrary number of sensors", as stated in [34]. For instance, they allow to fuse "multiple IMUs or multiple sources of odometry information"([34]).
- "Support for multiple ROS message types"([34]).
- Each sensor can have a customized input. "If a given sensor message contains data that you do not want to include in your state estimate, the state estimation nodes allow you to exclude that data on a per-sensor basis"([34]).
- Continuous estimation. "Each state estimation node begins estimating the vehicle's state as soon as it receives a single measurement", as written in [34]. If sensors do not send data for a long time, "the filter will continue to estimate the robot's state via an internal motion model"([34]).

The estimated state is a 15-dimensional vector composed by

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

The internal motion model, used for state prediction, is an omnidirectional motion model. In this case, the Jackal UGV is not represented by an omndirectional motion model, but the state estimation may be precise enough thank to the correction of state prediction through the perceived sensor data. That means, the error due to the prediction step may be very large in some cases, but they can be mitigated by applying a correction through the information coming from the IMU and the RTK-GPS receiver. The internal motion model of the state estimation nodes has not been modified, in order to not add computational complexity.

4.1.2 EKF localization node

The *ekf_localization_node* implements the extended kalman filter and allows to set a large number of parameters to control the accuracy of the filter as well as what sensors data include in the estimation process. In order to properly set each parameter, they are grouped together in configurations files, that are represented by *file_name.yaml*, in ROS. Among all the parameters to set, the two most important are: the configuration matrix relative to a specific source of information and the process covariance matrix. The former is composed as follows:

$$\begin{bmatrix} X & Y & Z \\ roll & pitch & yaw \\ \dot{X} & \dot{Y} & \dot{Z} \\ \dot{roll} & \dot{pitch} & \dot{yaw} \\ \ddot{X} & \ddot{Y} & \ddot{Z} \end{bmatrix}$$

Setting to *true* or *false* each element in the matrix, the filter will take into account only those set to *true*, when it fuse the sensor data. For instance, setting to *true* the first three elements of the matrix, the filter will consider only the (X, Y, Z) information coming from a specific sensor, although the sensing device is able to provide also $(roll, pitch, yaw)$.
The latter is a 15 by 15 matrix, that controls the amount of uncertainty in the prediction phase of the filter, that means how much the internal motion model of the filter matches the real motion model of the physical system. It is customizable, in order to choose the right values for a specific filter application, although it is difficult to tune each single parameter.
The *ekf_localization_node* can obtain the sensors data by subscribing to ROS topics on which each sensor publishes the sensed information written in a specific ROS message format. It accepts odometry messages as well as IMU messages. In this case the odometry information are obtained from the RTK-GPS receiver, that is supposed to be quite accurate. The GPS data are transformed from GPS message format to the odometry message format by means of the *navsat_transform_node*, that takes as inputs: the information coming from the RTK-GPS receiver, the

odometry information produced by the *ekf_localization_node* and the IMU data (for the robot's heading), in order to provide (as output) odometry messages represented in the robot's world frame.
In the Fig. 4.1 are represented the main ROS nodes (ellipses) and topics (rectangles)

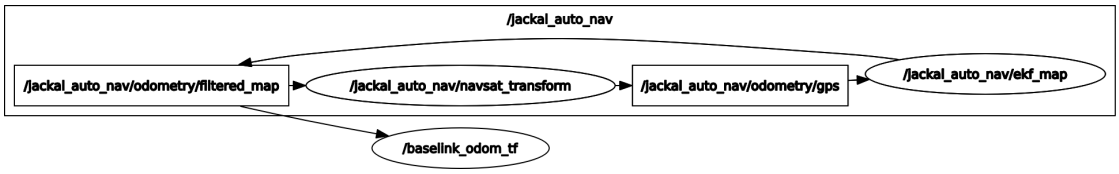


Figure 4.1: Main nodes and topics involved in the outdoor localization

used in the outdoor localization, by means of a ROS graph. Furthermore, the sensor data have not been included in the graphical representation to make the figure more understandable. The main involved nodes are:

- */baselink_odom_tf*: it is a custom node used only to publish the transformation from the *base_link* coordinate frame (attached to the robot) to the *odom* coordinate frame. It takes as inputs: the odometry information coming from the */jackal_auto_nav/ekf_map* node and provides the right data transformation.

- */jackal_auto_nav/ekf_map*: it is a *ekf_localization_node* used for pose estimation (including sensor data) and it is also responsible for the transformation between the *odom* coordinate frame and the *map* coordinate frame. The filter inputs are: the IMU and the GPS data, so that in case of wheels slippage the estimated pose does not suffer from big errors.

- */jackal_auto_nav/navsat_transform*: it computes the conversion between GPS and odometry message formats.

This particular setup is required because ROS is not able to publish directly the *base_link* $\rightarrow$ *map* transformation.
In this case, the reference frame is *map*, that is a global reference frame in ROS and it is used for navigation systems that exploite the GNSS, in order to obtain a global position. On the other hand, the *odom* reference frame is used mainly for navigation system that are based on relative localization.
All the transformations are obtained and published by means of the *tf* package (deeply described in C.2). Moreover, the conversion from one coordinate frame to another is always required in a ROS-based robot, because ROS associates to each robot's component (e.g. wheels, shaft, sensors, etc.) a specific coordinate frame (as better explained in C.1).

4.2 Autonomous GPS-based Navigation

ROS provides a ready-to-use stack for UGV navigation, called navigation stack and shown in Fig. 4.2. The major component of such stack is the *move_base* package, that is commonly used for 2D autonomous navigation in indoor evironments and is composed by the following nodes (as can be seen in Fig. 4.2):

- Global planner: it is responsible for computing a collision-free global plan between a starting point and a goal point, basing on the available global costmap and a path search algorithm.
- Local planner: it provides a local path, basing on the global path and the local costmap, that will be exploited to send the right velocity (both linear and angular) commands to the controller.
- Global costmap: it allows the user to set some parameters (e.g. the global reference frame, the update frequency, the usage of static or dynamic map, etc.) in order to adapt its usage for different applications. It is represented as an occupancy grid map with customizable dimensions and resolution.
- Local costmap: it shares the same characteristics of the global costmap.
- Recovery behaviors: it works as a backup node, when the robot is stuck, that means it is not able to find a valid path between a starting point and a goal point. The two main backup actions are: the in-place robot's rotation and the costmaps clearing, in order to look for an alternative path to reach the goal.

They share messages, through topics, and collaborate each other in order to achieve a complete autonomous navigation.
Referring to the Fig. 4.2, the *map_server* node and the *amcl* node have not been used. The first one is responsible for providing a map in the right format, but (as will be described later in 4.2.1) the navigation will be performed without a map. While, the second one, is a localization node based on the Adaptive MCL (AMCL), mainly used for localize a vehicle in a know environment, however the robot's pose information is provided by an EKF localization node, as stated in 4.1.

4.2.1 Known environment VS Unknown environment

Both the global and the local costmaps accepts static or dynamic maps, where dynamic means the map is updated in real-time through sensors information. As a consequence, the choice, about what kind of map to use, should be made.
The EKF localization node is able to provide a quite accurate pose of the UGV both in a know environment and an unknown environment, because the estimation

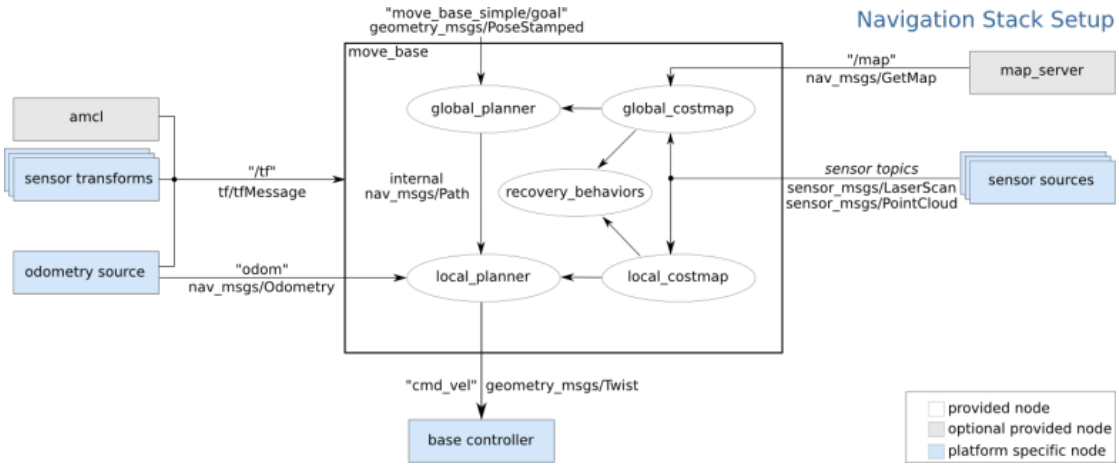


Figure 4.2: ROS navigation stack ([35])

process exploites an internal motion model combined with sensors data, without making reference to a map. The starting point of the thesis is the availability of GPS waypoints, that the UGV should follow in the most precise manner, as stated in 1.2.3. These GPS route points are computed on a know geo-referenced map, as a consequence the environment, in which the vehicle should navigate, can be assumed as know. However, the available geo-referenced map should be processed, in order to obtain a ROS-compatible map format.
On the other hand, there is the chance of assuming the environment as completely unknown and make the UGV navigate throughout it. The main problem of this approach is to avoid obstacles dynamically, since there is not a pre-computed static map, on which the navigation algorithm can base its path calculations.
All considered, the second option has been chosen, in order to avoid the computation of ROS-compatible map, each time the vehicle changes place of navigation. Moreover, the available GPS waypoints form a collision-free global path, because they have been computed basing a known geo-referenced map, in which the static obstacles are well defined. As a consequence, the issue of avoiding obstacles dynamically is partially solved, because if the UGV follows the provided global path in a quite accurate manner, it will not collide any static obstacles (e.g. vine plants, wooden poles, etc.) for sure. While, in case of dynamic obstacles, that means obstructions not stored on the geo-referenced map, the DWA will be used (as described later in 4.2.2) in order to avoid them.

4.2.2 Local autonomous navigation

The *move_base* package has been used as a starting point to develop the autonomous navigation system. On one hand, this package contains an already

built-in *global_planner* node, that cannot be eliminated due to the strong relationship with the *local_planner* node. On the other hand the global plan is already available and composed by GPS waypoints. The solution is to substitute the global planner with a fake one, that it takes as input the GPS waypoints and provides time by time the global plan to the *local_planner* node, without making any complex computations.

Fake global planner

The fake global planner computes the global plan as a simple straight line, that connects the actual robot's pose with the next GPS waypoint to be reached. The desired node does not exist, so it has been developed and named *Gps_planner*. It publishes the generated global plan and the goal on specific topics subscribed by the *local_planner* node.
The GPS waypoints are provided to the fake global planner by an additional custom node, named *ReadWaypoints*. Actually, it reads the waypoints on a file, although, for future improvements, the GPS waypoints can be furnished through a pre-defined ROS topic. Then, the *ReadWaypoints* node transform each GPS route point into UTM[1] (Universal Transverse of Mercator) format, through a function of *navsat_transform_node* and from UTM to cartesian coordinate thank to the *tf* package. After this step, it sends each waypoints, as a goal request, to the fake global planner using an already defined ROS action of the *move_base* package and waits until the UGV is inside an imaginary circle, that has as center the sent waypoint and as radius a tunable parameter. Eventually, when the UGV has reached the imaginary circle, the node sends another GPS route point, following the above procedure, until all the waypoints has been visited.
The imaginary circle has been obtained by computing the Euclidean distance between the pose of the UGV (provided by the *ekf_localization* node) and the GPS waypoint to be reached. In addition, the radius is a tunable parameter to allow the user to change the UGV's speed at which a route point is attained. This step is necessary to not stop at each waypoint, that is seen as a goal point by the *local_planner* node and to be robust against poor accuracy pose estimates.
The goal sent to the fake global planner has a specific ROS message format, named *PoseStamped*, in which it stores the cartesian coordinates of the waypoint and the orientation (expressed in quaternion form) at which the UGV should reach the specified goal. In this case, the orientation has been computed by the *ReadWaypoints* node as the angle of a straight line, that connects the current waypoint with the next one, with respect to the x axes defined in the map frame. The computed angle represents the yaw, while the pitch and the roll are set to

[1]https://it.wikipedia.org/wiki/Proiezione_universale_trasversa_di_Mercatore

zero, because it has been assumed that the UGV moves in a quite planar surface, although some vineyards may have a huge slope. Then, the yaw, expressed in radians, is converted to the quaternion form through a function of the *tf* package.

Local planner

Then, the local planner, based on the DWA (deeply described in 3.2.5), looks for the right velocity commands to send to the *base_controller* node. The velocity commands are generated taking into account the global plan, the dynamic obstacles and the goal point. Among the local planning algorithms available in literature, the DWA approach has been chosen because it is highly employed for collision avoidance purposes in ROS-enables robots and it has tunable gains, that are used to compute the cost functions related to each sampled trajectory. In this case, the used cost function is the following one:

$$\begin{aligned} Cost = & path\_gain \times distance\_path + goal\_gain \times goal\_distance \\ & + obstacle\_gain \times obstacle\_cost \end{aligned}$$

where

- $path\_gain, goal\_gain, obstacle\_gain$ represent the weightings for how much the generated local trajectory should stay close to the global path, should attempt to reach the local goal and should try to avoid obstacles, respectively. They are tunable parameters and have been chosen by exploiting a trial and error approach, through different simulations.
- *distance_path* is the distance from the global path.
- *goal_distance* is the distance from the local goal.
- *obstacle_cost* is the maximum obstacle cost along the considered local trajectory, represented in the range 0-254.

the *distance_path* and *goal_distance* are computed from the endpoint of the actual local trajectory in meters or map cells, according to the chosen option.
The required obstacle information are provided by the *local_costmap* node, through an updated costmap, as described in the following section.

Local costmap

The dynamic obstacles are provided by the *local_costmap* node, that constantly updates the costmap by taking as inputs the pointcloud generated by the depth camera. Moreover, it provides a plugin called *inflation layer*, that allows to properly represent the obstacles and free-spaces inside the costmap, such that the local

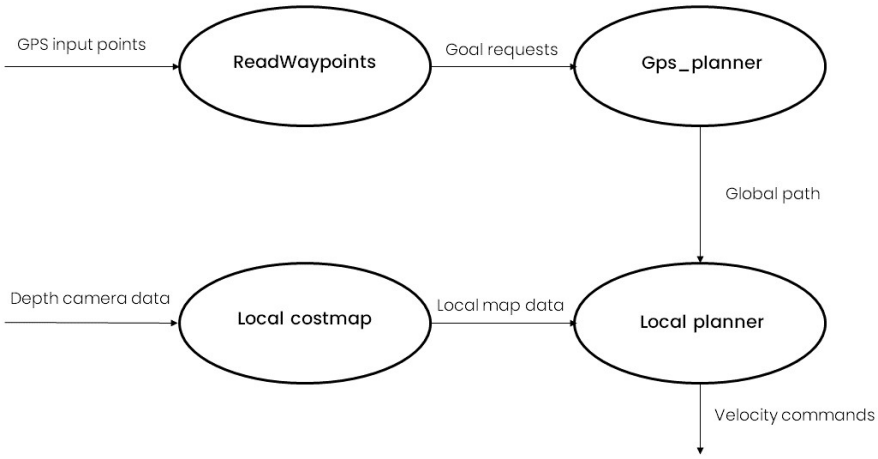
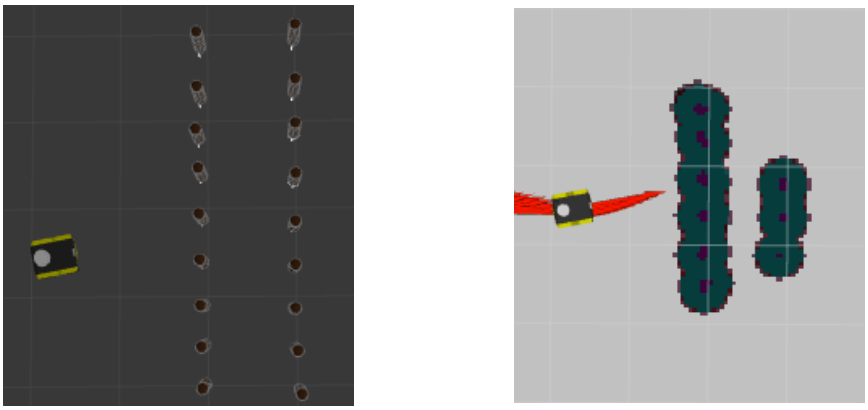


Figure 4.3: Navigation nodes

planner is able to compute a collision-free local plan. The *inflation layer* assign a cost value (from 0 to 254) to each cell of the costmap according to the shape of the UGV and the data coming from the camera. Although each cell can assume 255 different cost values, the *local_costmap* node assigns to each of them one of three states: occupied, free and unknown. This choice is made by taking the cost values produced by the *inflation layer* and according to a threshold (that can be easily changed) one of the three state is assigned to each cell. Eventually, through the inflation process, the obstacles are represented with an increased size proportional to the robot's size. as shown in Fig. 4.4.



(a) Virtual environment **(b)** Obstacles representation

Figure 4.4: Comparison between simulation environment and what the robot senses.

Chapter 5

Simulation

Simulation is one of the most important task in case of development of software components based on new algorithms, that has never been used and tested. A good simulation environment allows developers to verify the correct behavior of a new software item and, at the same time, to observe the interaction with the simulated surrounding environment without causing damages to objects or people. The autonomous navigation system (illustrated in 4) has been tested using the Gazebo simulator and the Rviz 3D visualization tool because they work very well with ROS and offer a lot of customization options. Besides a brief description of Gazebo and Rviz, the following part deals with the creation of a custom simulated environment as well as the presentation of the obtained results.

5.1 Gazebo

Gazebo[1] is a 3D simulator, that works well with ROS and offers several tools and options to make the simulation environment as much as possible realistic. It provides an easy to use Graphical User Interface (GUI) (as can seen in Fig. 5.1), that allows users to add, delete or update their 3D models. In addition, the main technical characteristics of Gazebo are:

- Dynamics simulation: it support dynamics simulation of 3D robot model, through high-performance physics engine (e.g. ODE, Simbody, DART).
- Advanced 3D graphics: Exploiting OGRE[2], it is possible to render realistic scenarios (as can be seen in 5.2), including high-quality lighting, textures or shadows.

[1] http://gazebosim.org/

[2] https://www.ogre3d.org/

- Sensors and noise: it provides a way to simulate a huge number of sensors, like LiDARs, cameras, GPS receivers, etc. as well as the noise associated with them.
- Robot models: either it offers ready to use 3D robot models or it is possible to build your own 3D model following the SDF[3] format.

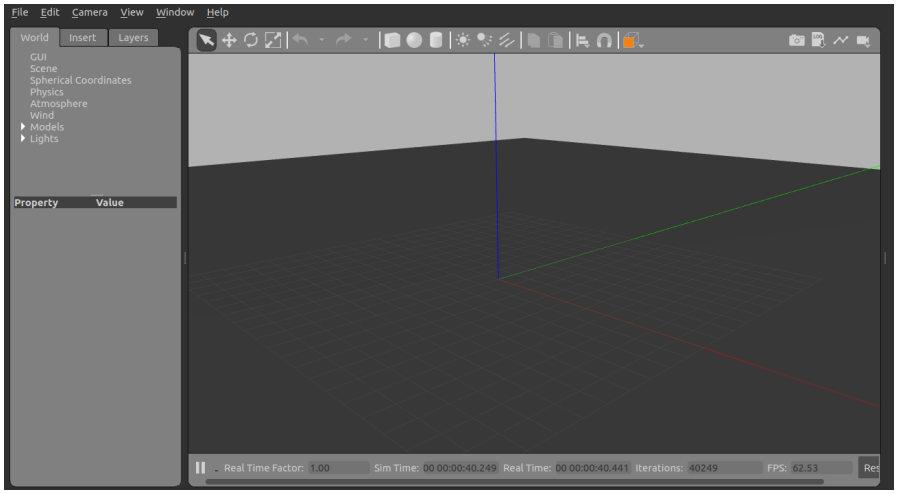


Figure 5.1: GUI of Gazebo

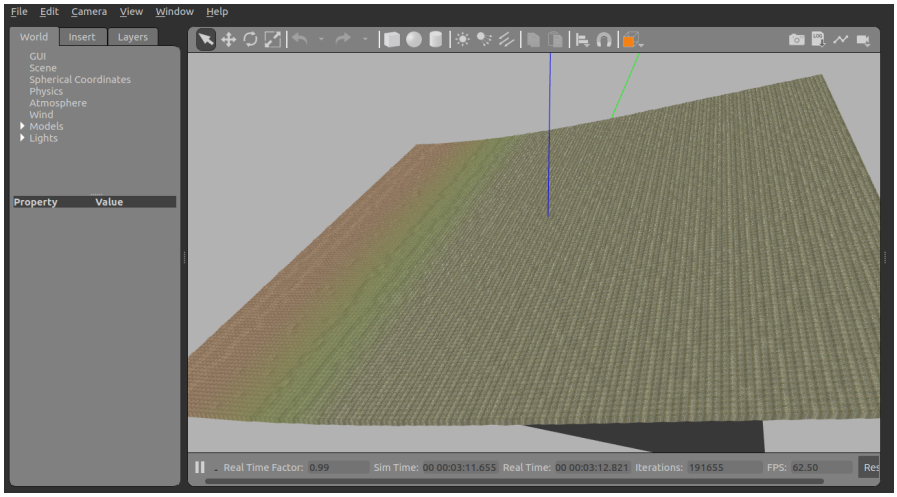


Figure 5.2: Example of a realistic simulated uphill terrain

[3]http://sdformat.org/

5.2 Rviz

RViz is a ROS 3D visualization tool, that allows users to be aware of how robots perceive and interact with the surrounding environment. Through an intuitive GUI (represented in Fig. 5.3), it is possible to choose a variety of different information to visualize, such as sensor incoming data, maps of the surrounding environment, how robots perceive the obstacles etc. Moreover, RViz allows to interact with robots in real-time, through a tool, called interactive markers. Eventually, RViz is highly employed with Gazebo for simulation and testing purposes of newly developed algorithms, such as navigation, localization and obstacle avoidance algorithms.

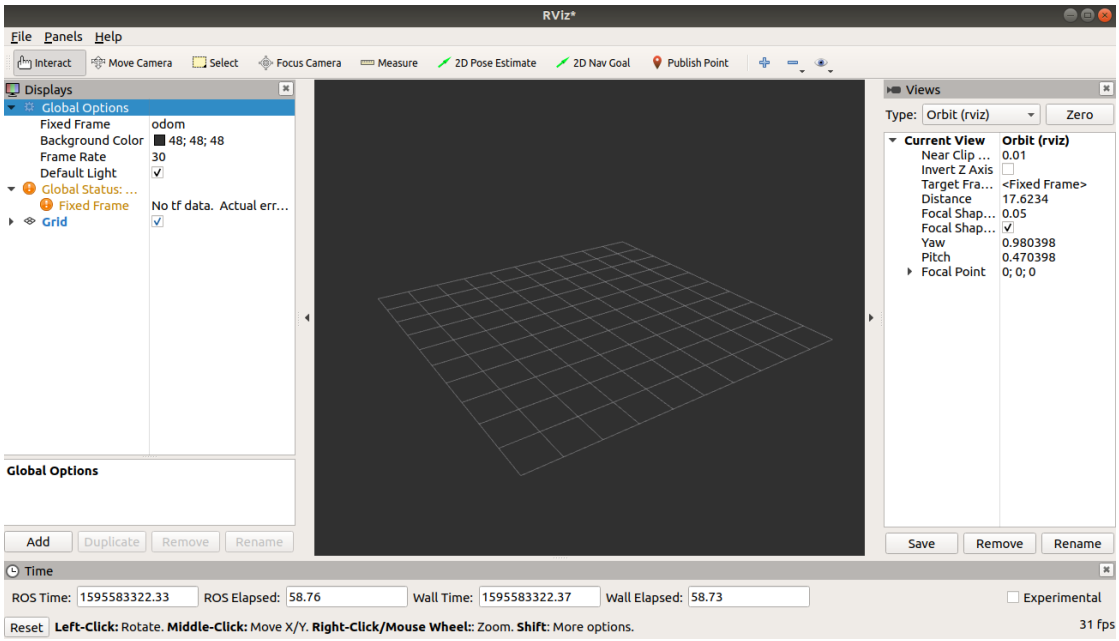


Figure 5.3: GUI of RViz

5.3 Simulation environments and results

The developed autonomous GPS-based navigation system has been tested through two different virtual environments, in order to check the functionalities in different scenarios. The main properties, that have been verified are:

- the obstacle avoidance: it can be tested only visually, because there is no other ways of doing that.
- the achievement of GPS waypoints: it can be verified both visually and comparing the UGV's pose with the route point to be reached.

- the localization: it can be checked by comparing the pose provided by the EKF localization node and the real UGV's pose furnished by a Gazebo plugin.

The simulation environments are composed by three main models: the Jakcal model (equipped with the used sensors), the terrain model and the vine plant model. The first one is provided by the Clearpath Robotics through the Unified Robot Description Format (URDF), that "is an XML format for representing a robot model"([36]). The furnished URDF file contains all the mechanical specifications to correctly simulate the Jackal UGV. Moreover, it is possible to add or remove sensors' models according to the applications. In this case, Gazebo provides ready to use plugins to simulate the GPS receiver, the IMU sensor and the Intel RealSense depth camera, although it is not possible to simulate an RTK-GPS receiver. For each simulated sensor, it is possible to set a variety of parameters, such as the amount and the type of noise, the sample frequency, etc.
The second one should represent a real 3D vineyard terrain, that is often uneven and may cause wheel slippage. Gazebo allows to represent a real terrain through the Simulation Description Format (SDF), which is an XML format used for environment or robot modelling. In particular, a real terrain surface can be simulated in Gazebo exploiting the heightmap option of the simulator. An heightmap can be easily rendered in Gazebo by providing a 8-bit grey scale image to the SDF file, specifing its dimension, the minimum and the maximum height of the terrain that should be simulated. Then, Gazebo will take the grey scale image as input and assign a specific value to each pixel according to the minimum and the maximum pre-defined heights as well as the pixel value (between 0 to 254).
Eventually, the vine plant model can be rendered by Gazebo, through an SDF file in which there is a reference to a 3D model of the vine plant in the COLLADA format.
Each simulation environment has wooden poles instead of simulated vine plants because a single model of a real vine plant is quite huge in terms of memory occupancy and slows done the entire simulation. However, this does not affect too much the testing purposes.

5.3.1 First simulation environment

The first simulation environment (shown in Fig. 5.4) contains wooden poles organized in such a way to reproduce a vine row, but it does not contain the vineyard terrain in order to not add too much noise to sensor measurements.

Taking into account Fig. 5.5, it is possible to observe that the EKF localization filter works quite well since the green line and the dashed dot red line are very similar. The *True pose* line has been obtained through a Gazebo plugin, that provides the true pose of the robot inside the simulation environment. In addition, the autonomous navigation system is able to follow the provided GPS route points

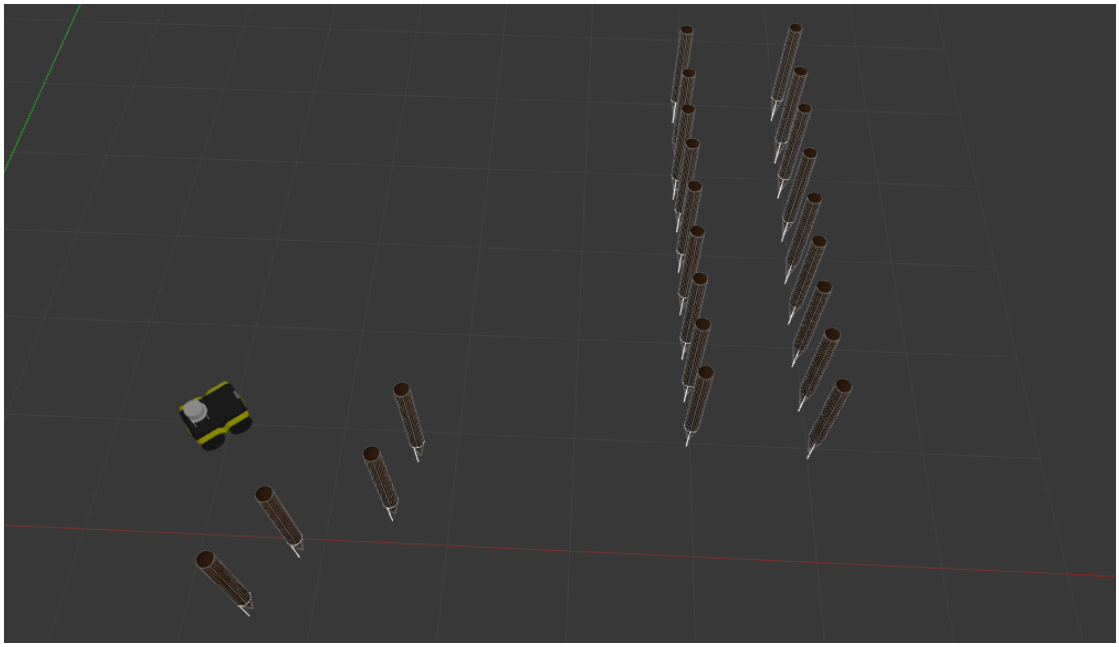


Figure 5.4: First simulation environment

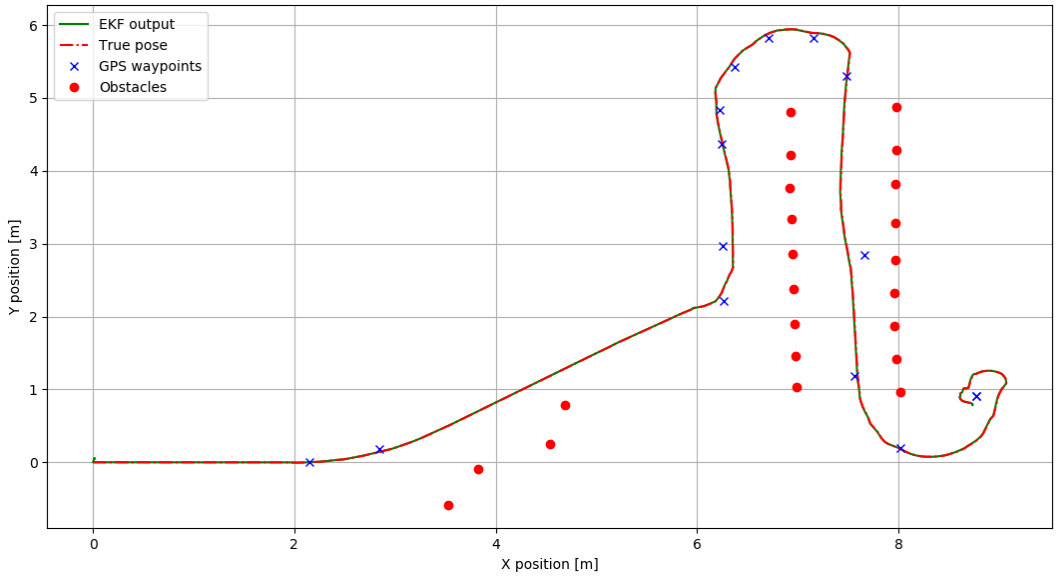


Figure 5.5: Simulation results

in a quite accurate manner without hurting any obstacles. All considered, the overall obtained results are very good, although in some cases, the UGV does not

enter in the vine row in a smooth way due to the obstacles size increasing, as will be better explained in 5.3.3.

5.3.2 Second simulation environment

The second simulation environment (shown in Fig. 5.6) is similar to the first one in terms of wooden poles organization and it contains also the simulated vineyard terrain in order to check if the autonomous system performs well in presence of uneven terrain.

Taking into account Fig. 5.7, it is possible to observe that the EKF localization

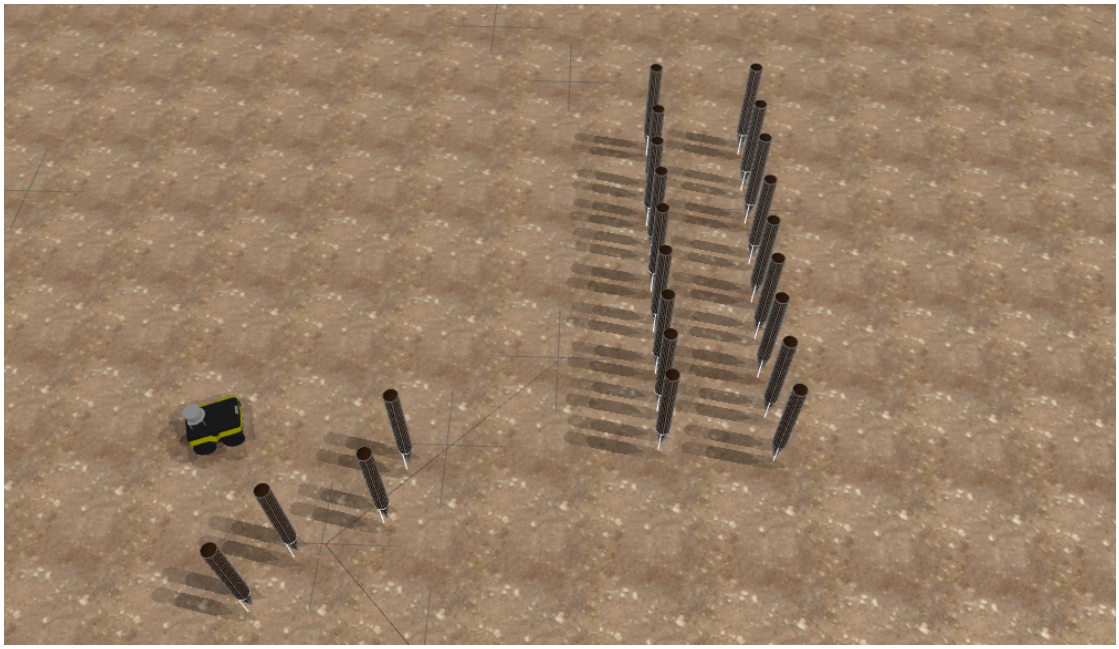


Figure 5.6: Second simulation environment

filter works reasonably well also in presence of uneven terrain, that may cause noisy IMU measurements. In addition, the autonomous navigation system is able to follow the provided GPS route points in a quite accurate manner without hurting any obstacles, as in 5.3.1. Also in this simulated scenario, sometimes the UGV does not enter in the vine row in a smooth way due to the obstacles size increasing. All considered, the overall obtained results are pretty good, although in some cases the UGV slides either left or right, due to the terrain and it has to perform some back and forth maneuvers to avoid the obstacles, as highlighted in Fig. 5.7 with the two orange ellipses.

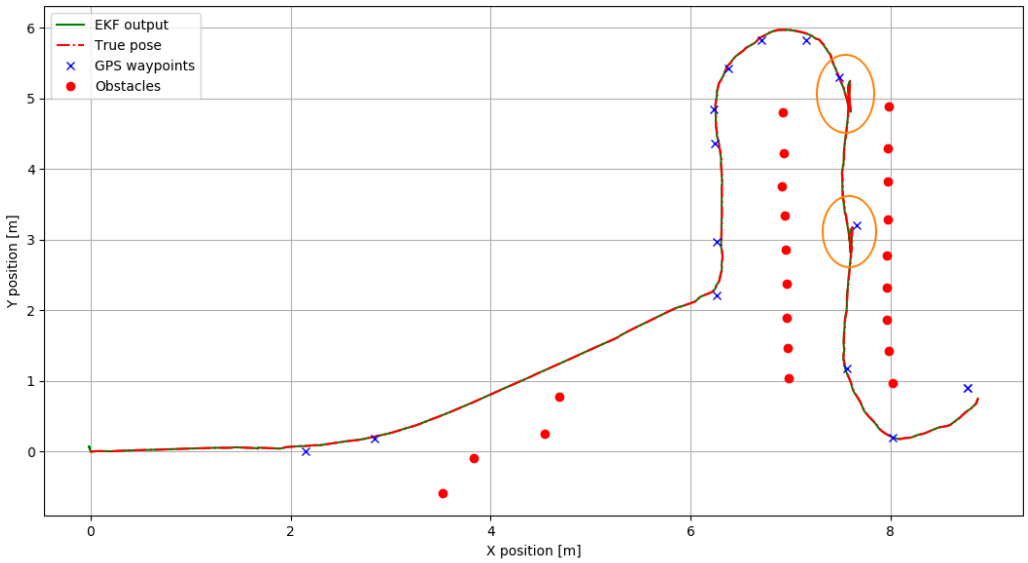
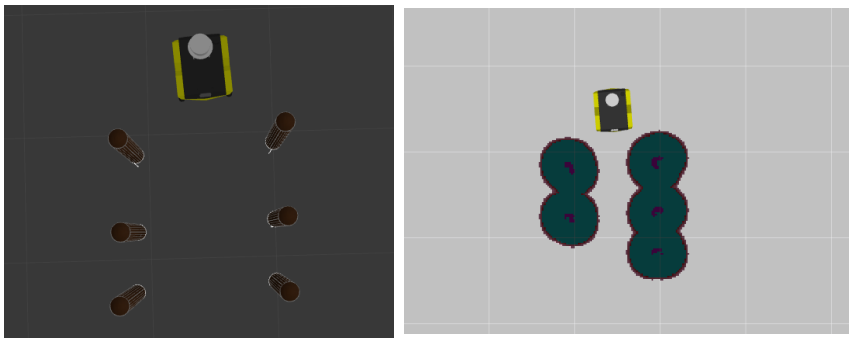


Figure 5.7: Simulation results

5.3.3 Common issue

The common simulation problem is the entry point of a vine row, because, in some cases, the UGV has to perform some additional maneuvers (as can be seen in Fig. 5.8) in order to be able to safely enter in the vine row. This problem is mainly caused by the increased size of the obstacles in the costmap used by the local planner, as can be obeserved in Fig. 5.9. Indeed, the DWA (used in the local planner) tries to find a local optimal path, trading off the distance from the global path, the goal achievement and the obstacle avoidance, and in some cases it prefers to perform additional maneuvers in order to be better aligned with respect to the next goal and as much as possible far away from obstacles. However, these anomalous situations does not happen frequently and does not penalize so much the whole path in terms of time and performances. Moreover, it may be mitigated by the collaboration with the local planner (described in 1.2.1) once the vine row entry has been reached.

Figure 5.8: Three examples of difficult entry in the vine row



(a) Simulation in Gazebo **(b)** Rviz view

Figure 5.9: Entry point of a vine row

Chapter 6

Conclusions and future works

This chapter deals with the peculiar characteristics of the proposed solution to the autonomous navigation problem in precision agriculture applications. In particular, there will be described its strengths and weakness as well as the future improvements and works, that may be made in order to achieve a more effective autonomous navigation.

6.1 Key properties of the autonomous navigation system

The previous illustrated solution has been designed and developed in order to be:

- Flexible: it is mainly focused on the vineyards scenario, but it can be employed in different agricultural environments provided that a global path made of GPS route points is available.
- Modular: thank to ROS, the proposed solution is highly modular, so that software updates and bug corrections can be made in the most easy way, wasting less time.
- Robust against uneven terrains: in agricultural scenarios the terrain is often irregular and bumpy, as a consequence the UGV's pose, estimated using wheel encoders, may be very inaccurate. However, in this case the localization task has been solved through an EKF, that includes only IMU and GPS receiver measurements, so that the wheel slippage phenomena cannot influences in a bad way the estimation process, although the IMU measurements may be noisy due to the irregularity of the terrain.

- Safe: it exploits the depth camera to detect dynamical obstacles and avoids them through the Dynamic Window Approach.

Besides the previous strengths, the autonomous navigation algorithm is affected by the following limitations:

- Conservativeness against obstacles: the agricultural environments may be surrounded by a lush vegetation, such as grass, weeds and leaves of the plants (as happened in a vineyard during the spring and summer seasons). In such cases, the depth camera recognizes as obstacles also the surrounding vegetation, because it is not able to make a difference between a solid obstruction with respect to grass or weeds that can be easily trampled on. However, it assumed that the agricultural scenario in which the UGV should operate is well-maintained, so that the navigation algorithm is able to perform its task.
- Difficult entry in vine rows: as previously described in 5.3.3, sometimes the Dynamic Window Approach experiences some troubles to enter in a new vine row. However, such minor issue can be easily solved by integrating the provided solution with the local planner, based on machine learning and described in 1.2.1.

All considered and as shown in 5.3.1 and in 5.3.2, the overall autonomous navigation has provided very good results in a simulated environment and can be tested in a real environment.

6.2 Next works

The proposed solution should be deployed and integrated on the Jackal UGV, with the local planner developed by Diego Aghi, Vittorio Mazzia, Marcello Chiaberge and briefly described in 1.2.1, in order to perform some field tests and check the behaviour of the overall system in a real scenario. Moreover, some improvements are required in order to solve some limitations described in 6.1. For instance, the information provided by the depth camera should be processed by an advanced algorithm, maybe based on machine learning, to classify the real obstacles and the fictitious ones.
All considered, the provided solution can be a starting point for future developments in the field of autonomous navigation for precision agriculture applications.

Appendix A

Jackal UGV

Jackal is an unmanned ground vehicle (UGV) developed by the Clearpath Robotics. "It is a small, fast, entry-level field robotics research platform, equipped with an onboard computer, GPS and IMU fully integrated with ROS for out-of-the-box autonomous capability. As with all Clearpath robots, Jackal is plug-and-play compatible with a huge list of robot accessories to quickly expand your research and development", as stated in [37].

A.1 Main characteristics

Jackal is a versatile UGV, suitable for different robotics applications, because it is:

- powerful and customizable: it is possible to easily add and connect sensors (e.g. cameras, LiDARs, etc.). In addition, the internal area has some empty spaces for additional computing power or storage.
- adapt for all-terrain and weatherproof: "Jackal is built from a sturdy aluminum chassis made with a high torque 4×4 drivetrain for rugged all-terrain operation. It has an IP62 weatherproof casing and is rated to operate from -20 Celsius or +45 Celsius", as written in [37]. IP62 is the Ingress Protection Code, that indicates the level of protection against water, dust, etc.

Moreover, the Clearpath Robotics provides a ready to use Gazebo model of the Jackal UGV, through an URDF file from which the main technical specifications have been obtained and summed up in Tab. A.1

A.2 Mathematical modeling

The Jackal UGV has four fixed wheels (two for each side) and it is able to make curvilinear trajectories by differentially driving the wheel pairs. This mechanical

Table A.1: Main parameters of Jackal

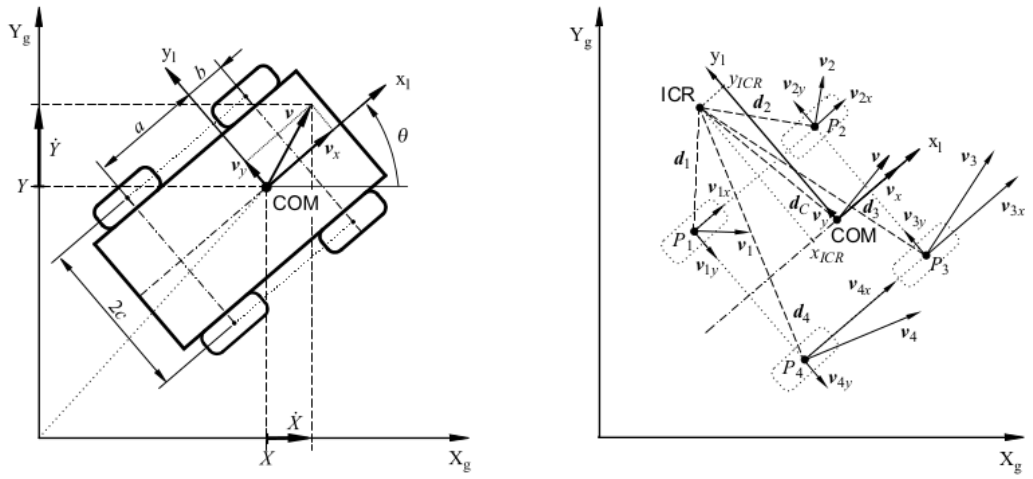
Parameter	Value
length body	0.42 $[m]$
width body	0.31 $[m]$
height body	0.184 $[m]$
wheelbase	0.262 $[m]$
track width	0.37559 $[m]$
wheel width	0.04 $[m]$
wheel radius	0.098 $[m]$
mass body	16.523 $[kg]$
mass wheel	0.477 $[kg]$
inertia body around z-axis	0.4485 $[kg \cdot m^2]$

Figure A.1: Jackal by Clearpath Robotics

configuration make it a skid-steering mobile robot (SSMR) and "the control of an SSMR is a challenging task because the wheels must skid laterally to follow a curved path", as stated in [38]. In the following part the kinematic and the dynamical model of a skid-steering mobile robot will be presented, assuming that the mobile platform is moving on a planar surface (to make the calculations easier).

A.2.1 Kinematic Model

Referring to the Fig. A.2, the inertial reference frame is denoted by (X_g, Y_g, Z_g), while the local reference frame attached to the center of mass (COM) is denoted by (x_l, y_l, z_l). The coordinate of the COM can be written as (X, Y, Z) in the inertial reference frame. Supposing that the UGV moves in the plane with a linear



(a) Body diagram

(b) Velocities diagram

Figure A.2: Skid-steering mobile robots diagrams [38]

velocity $v = [v_x, v_y, 0]^T$ and an angular velocity $\omega = [0, 0, \omega]^T$, it is possible to represents the generalized coordinates as $q = [X, Y, \theta]$ and the generalized velocities as $\dot{q} = [\dot{X}, \dot{Y}, \dot{\theta}]$. In the planar case, $\dot{\theta} = \omega$. All considered, the following equation represents the free-body kinematics of the robot:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{A.1}$$

Then, taking into account the relationship between wheel angular velocities and local velocities of the robot, it is possible to compute some constraints on the robot movements. In the following part $\omega_i(t)$ with $i = 1,2,3,4$ are the wheels angular velocities and the thickness of the wheels will be neglected, such that each wheel has only one contact point with the plane denoted as P_i. Moreover, it is assumed that there is no longitudinal slippage between the wheel and the surface. Considering all the previous assumptions the following equations have been derived:

$$\begin{aligned} v_{ix} &= r_i \cdot \omega_i \\ d_i &= [d_{ix}, d_{iy}]^T \\ d_C &= [d_{Cx}, d_{Cy}]^T \end{aligned}$$

where v_{ix}, d_i and d_C are represented in the local reference frame taking into account the instantaneous center of rotation (ICR) and r_i is the effective rolling radius of the ith wheel.

Basing this analysis on the geometry described in Fig. A.2, the following expression has been derived:

$$\frac{\|v_i\|}{\|d_i\|} = \frac{\|v\|}{\|d_C\|} = |\omega| \tag{A.2}$$

that, in a more detailed form can be written as:

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{Cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{Cx}} = \omega \tag{A.3}$$

The equality A.3 can be re-written as:

$$\frac{v_x}{y_{ICR}} = -\frac{v_y}{x_{ICR}} = \omega \tag{A.4}$$

considering the local coordinates of the ICR=$(x_{ICR}, y_{ICR}) = (-d_{xC}, -d_{yC})$.
In addition, (always taking into account the Fig.A.2) it is possible to define the following relationships:

$$\begin{aligned} d_{1y} &= d_{2y} = d_{Cy} + c \\ d_{3y} &= d_{4y} = d_{Cy} - c \\ d_{1x} &= d_{4x} = d_{Cx} - a \\ d_{2x} &= d_{3x} = d_{Cx} + b \end{aligned} \tag{A.5}$$

"where a, b and c are positive kinematic parameters", as written in [38]. Combining the equations A.3 and A.5 "the following relationships between wheel velocities can be obtained"([38]):

$$\begin{aligned} v_L &= v_{1x} = v_{2x} \\ v_R &= v_{3x} = v_{4x} \\ v_F &= v_{2y} = v_{3y} \\ v_B &= v_{1y} = v_{4y} \end{aligned} \tag{A.6}$$

"where v_L and v_R denote the longitudinal coordinates of the left and right wheel velocities, v_F and v_B are the lateral coordinates of the velocities of the front and rear wheels, respectively", as stated in [38].
Moreover, exploiting the equations A.3 and A.6, the following additional relationship has been derived:

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & c \\ 0 & -x_{ICR} + b \\ 0 & -x_{ICR} - a \end{bmatrix} = \begin{bmatrix} v_x \\ \omega \end{bmatrix} \tag{A.7}$$

then, assuming $r_i = r$ for each wheel, it is possible to write:

$$\omega_\omega = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \tag{A.8}$$

"where ω_L and ω_R are the angular velocities of the left and right wheels, respectively"([38]).
Eventually, combining the equations A.7 and A.8, the relationship between the velocities of the robot and the angular wheel velocities has been obtained:

$$\eta = \begin{bmatrix} v_x \\ \omega \end{bmatrix} = r \begin{bmatrix} \frac{\omega_L + \omega_R}{2} \\ \frac{-\omega_L + \omega_R}{2c} \end{bmatrix} \tag{A.9}$$

the accuracy of the above relations decreases proportionally to the increase of the slippage phenomena. To complete the Eq. A.9, it is necessary to define the following additional constraint, that comes from the Eq. A.4:

$$v_y + x_{ICR}\dot{\theta} = 0 \tag{A.10}$$

it can be expressed in Pfaffian form as follows:

$$\begin{bmatrix} -sin\theta & cos\theta & x_{ICR} \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = A(q)\dot{q} = 0$$

All considered, the following equation defines the kinematic model of a SSMR:

$$\dot{q} = S(q)\eta \tag{A.11}$$

where: $S^T(q)A^T(q) = 0$ and

$$S(q) = \begin{bmatrix} cos\theta & x_{ICR}sin\theta \\ sin\theta & -x_{ICR}cos\theta \\ 0 & 1 \end{bmatrix}$$

The above kinematic model describes a non-holonomic system due to the constraint expressed in the Eq. A.10; such equation is not integrable as a consequence it defines a non-holonomic constraint.

A.2.2 Dynamical Model

The dynamical model will be described "using the Lagrange-Euler formula with Lagrange multipliers to include the nonholonomic constraint"([38]). In the Lagrangian will not be included the potential energy, since it is zero due to the plane motion assumption. As a consequence the Lagrangian can be defined as follows:

$$L(q, \dot{q}) = T(q, \dot{q})$$

where $T(q, \dot{q})$ is computed taking into account "the kinetic energy of the vehicle and neglecting the energy of rotating wheels"([38]).

$$T = \frac{1}{2}mv^T v + \frac{1}{2}I\omega^2 \tag{A.12}$$

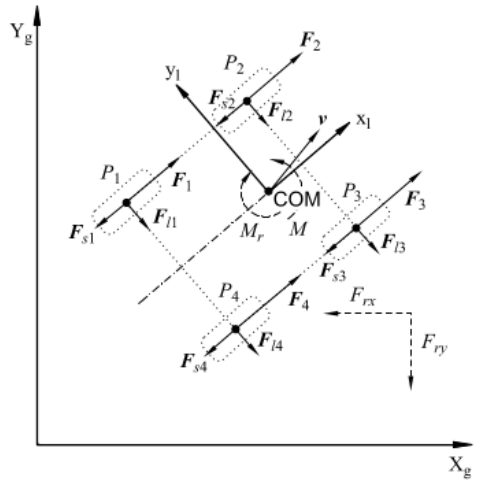


Figure A.3: Forces applied on the robot [38]

In the above equation, m is the mass of the robot, I is the moment of inertia of the robot about its COM and $v^T v = v_x^2 + v_y^2 = \dot{X}^2 + \dot{Y}^2$, that leads to rewrite Eq.A.12 as follows:

$$T = \frac{1}{2} m(\dot{X}^2 + \dot{Y}^2) + \frac{1}{2} I\dot{\theta}^2 \tag{A.13}$$

Applying the partial derivative and the time derivative to $T(q, \dot{q})$ (that is the kinetic energy), the following inertial forces have been obtained:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}} = \begin{bmatrix} m\ddot{X} \\ m\ddot{Y} \\ I\ddot{\theta} \end{bmatrix} = M(q)\ddot{q} \tag{A.14}$$

where:

$$M(q) = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}$$

Moreover the following generalized resistive forces vector has been taken into account:

$$R(\dot{q}) = \begin{bmatrix} F_{rx}(\dot{q}) & F_{ry}(\dot{q}) & M_r(\dot{q}) \end{bmatrix}^T \tag{A.15}$$

where F_{rx} and F_{ry} are resultant forces expressed in the inertial reference frame and can be defined as follows:

$$F_{rx}(\dot{q}) = cos\theta \sum_{i=1}^{4} F_{si}(v_{xi}) - sin\theta \sum_{i=1}^{4} F_{li}(v_{yi})$$
$$F_{ry}(\dot{q}) = sin\theta \sum_{i=1}^{4} F_{si}(v_{xi}) + cos\theta \sum_{i=1}^{4} F_{li}(v_{yi})$$

and M_r is the resistant moment around the COM and can be computed as follows:

$$M_r(\dot{q}) = -a \sum_{i=1,4} F_{li}(v_{yi}) + b \sum_{i=2,3} F_{li}(v_{yi}) - c \sum_{i=1,2} F_{si}(v_{xi}) + c \sum_{i=3,4} F_{si}(v_{xi})$$

In the above equations, F_{li} and F_{si} are the lateral and longitudinal friction forces, respectively, for the ith wheel. They can be computed using the friction coefficients and some considerations on the normal forces applied from the plane to the robot's wheels.
"The active forces generated by the actuators which make the robot move can be expressed in the inertial frame as follows"([38]):

$$F_x = cos\theta \sum_{i=1}^{4} F_i = \frac{1}{r} cos\theta \sum_{i=1}^{4} \tau_i \tag{A.16}$$

$$F_y = sin\theta \sum_{i=1}^{4} F_i = \frac{1}{r} sin\theta \sum_{i=1}^{4} \tau_i \tag{A.17}$$

while, the active torque around the center of mass can be computed as:

$$M = c(-F_1 - F_2 + F_3 + F_4) = \frac{1}{r} c(-\tau_1 - \tau_2 + \tau_3 + \tau_4) \tag{A.18}$$

where τ_i is the torque applied by the actuator on the ith wheel. The active forces and the active torque can be grouped in the following vector:

$$F = \begin{bmatrix} F_x & F_y & M \end{bmatrix}^T \tag{A.19}$$

and, defining τ_L and τ_R as "the torques produced by the wheels on the left and right sides of the vehicle, respectively"([38]), it is possible to write:

$$\tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} = \begin{bmatrix} \tau_1 + \tau_2 \\ \tau_3 + \tau_4 \end{bmatrix} \tag{A.20}$$

Combining the vector F and the vector τ, the following relationship has been obtained:

$$F = B(q)\tau \tag{A.21}$$

where B "is the input transformation matrix defined as"([38]):

$$B(q) = \frac{1}{r}\begin{bmatrix} cos\theta & cos\theta \\ sin\theta & sin\theta \\ -c & c \end{bmatrix}$$

All considered, combining the equations A.14, A.15, A.21 and using a vector of Lagrange multipliers (λ), in order to take into account the non-holonomic constraint (defined previously), it is possible to obtain the following dynamical model:

$$M(q)\ddot{q} + R(\dot{q}) = B(q)\tau + A^T(q)\lambda \tag{A.22}$$

Appendix B

Sensors

In the following part there will be described the technical specifications of the used sensors.

B.1 GPS receiver

The FlexPak6 (shown in Fig. B.1) by Novatel[1] is "capable of tracking all present and upcoming Global Navigation Satellite System (GNSS) constellations and satellite signals including GPS L1/L2/L2C/L5, GLONASS L1/L2/L2C, Galileo E1/E5a/E5b/AltBOC and BeiDou B1/B2 signals"([39]). It provides several customization options to meet the user needs, for instance the accuracy of positioning can range from metre to centimeter-level. As written in [39], it is "compact, lightweight, easy to integrate and ideal for low payload UAV and robotics applications". Moreover, it provides "navigation output support for NMEA 0183"([39]) communication protocol, that is deeply described in B.1.2 and necessary for the integration with ROS.
The main technical characteristics of the receiver are summed up in the Table B.1.

B.1.1 GPS receiver testing

The verification of the GPS receiver has been carried out in the following way: the antenna has been placed on a point with known GPS coordinates, then, after an initial phase of signal tracking, the position solutions provided by the receiver has been stored on a file. The known point has the following GPS coordinates,

[1] https://novatel.com/

Figure B.1: FlexPak6 module by Novatel

Table B.1: Main technical characteristics of FlexPak6 module

Features	**Description**
Dimensions	$147 \times 113 \times 45mm$
Weight	$337g$
Operating temperature	-40°C to $+75$°C
Single Point L1 horizontal position accuracy	$1.5m$(RMS)
Single Point L1/L2 horizontal position accuracy	$1.2m$(RMS)
SBAS horizontal position accuracy	$0.6m$(RMS)
DGPS horizontal position accuracy	$0.4m$(RMS)
RTK horizontal position accuracy	$1cm + 1ppm$(RMS)
Time to first fix	$< 50s$(cold start),$< 35s$(hot start)
Maximum data rate	$100Hz$

expressed in decimal degrees:

$$Latitude = 45.0620986N \qquad Longitude = 7.663334E$$

The GPS signals have been tracked for about 15 minutes, storing 839 points and the position solutions provided by the receiver have been computed employing the differential corrections coming from a Satellite-based Augmentation System

(SBAS). The obtained results are shown in Fig. B.2 expressed in latitude and longitude coordinates, while in Fig. B.3 the same results have been transformed in kilometers, in order to have a metre of comparison. Eventually, in Fig. B.4, all the points have been transformed in meters relatively to the known point, in order to check for the maximum and minimum Euclidean distance from that point, obtaining:

$$maximum\_distance = 1.2519m \qquad minimum\_distance = 0.2359m$$

The shown results are quite good taking into account that 15 minutes is a relatively

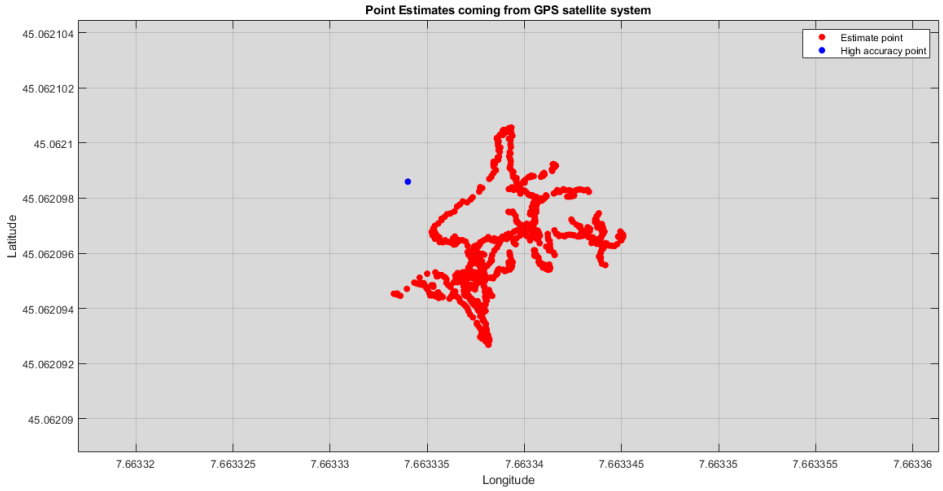


Figure B.2: Latitude and longitude coordinates

short time to obtain very accurate measurements also exploiting the differential corrections.

B.1.2 NMEA protocol

The National Marine Electronics Association (NMEA) has defined and, currently mantains, the NMEA standard of communication. It has been developed and improved through different versions. Actually, the most recent and used version is the NMEA 0183. This standard "is a combined electrical and data specification for communication between marine electronics such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments", as stated in [40]. "The NMEA 0183 standard uses a simple ASCII, serial communications protocol"([40]), in order to establish how data can be transmitted in a sequence of characters, called sentence, from one device (a talker), to multiple

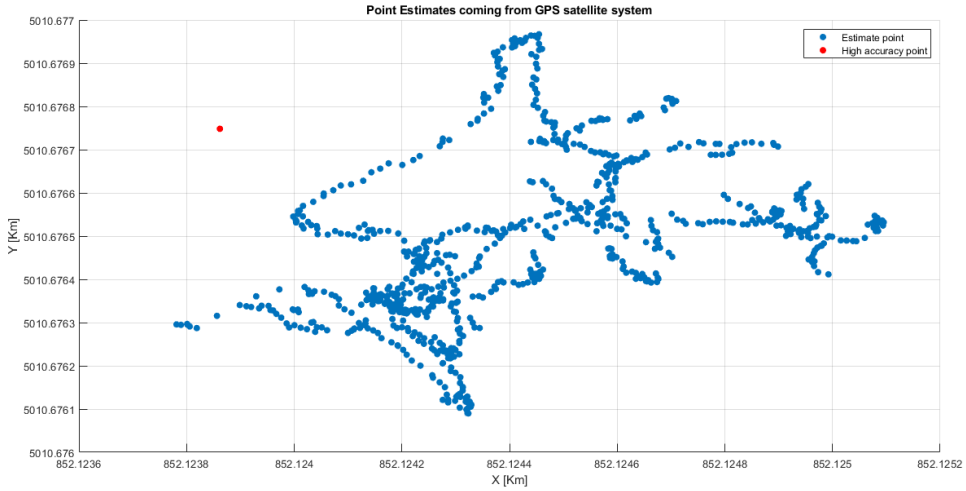


Figure B.3: Results expressed in kilometers

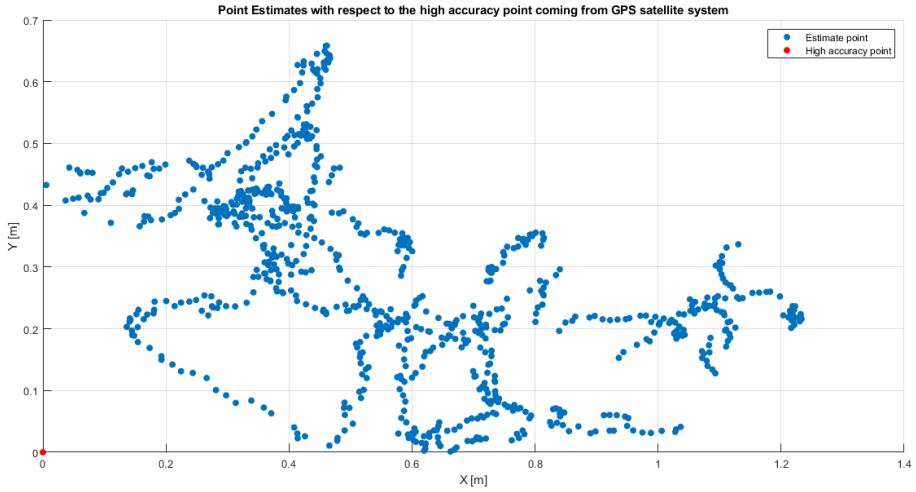


Figure B.4: Relative distance from the known point

receivers (listeners), at a time. While, the used electrical standard is EIA-422[2], also known as RS-422. Moreover, at the user level, the transmitted sentences have a well defined structure and contents, as shown in Fig. B.5, that can be summed up as follows:

[2]https://en.wikipedia.org/wiki/RS-422

- Length: each message has a maximum length of 82 characters, including the starting and ending characters.
- Start character: "the start character for each message can be either a $ (For conventional field delimited messages) or ! (for messages that have special encapsulation in them)", as written in [40].
- Talker identifier: the two characters after the starting one defines the type of communicating devices.
- Message type: it is identify by three characters after the first three ones.
- Data fields: they are comma separated, in case of unavailable data the corresponding field is leaved blank.
- Checksum: it is "represented as a two-digit hexadecimal number"([40]) and obtained through "the bitwise exclusive OR of ASCII codes of all characters between the $ and *, not inclusive"([40]). It "is optional for most data sentences, but is compulsory for"([40]) some others. When the checksum is present, the last data field character is followed by an asterisk, that delimits the starting point of the checksum.
- Ending characters: each message is ended by <CR><LF>.

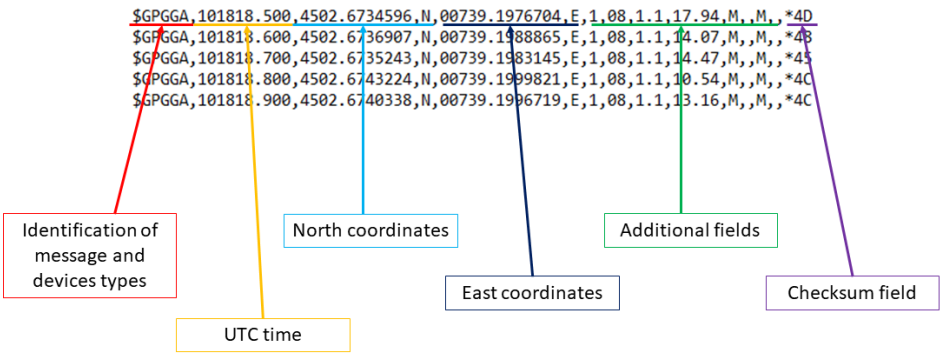


Figure B.5: Example of some NMEA messages

B.2 Camera

The Intel RealSense D435i is a depth camera with an integrated IMU, in order to "refine its depth awareness in any situation where the camera moves"([41]). It is the preferred solution for object recognition and robotic navigation, "thank to the combination of a wide field of view and global shutter sensor"([41]). Fig. B.6 shows the main components of the depth camera, while in Table B.2 are described the main technical characteristics.

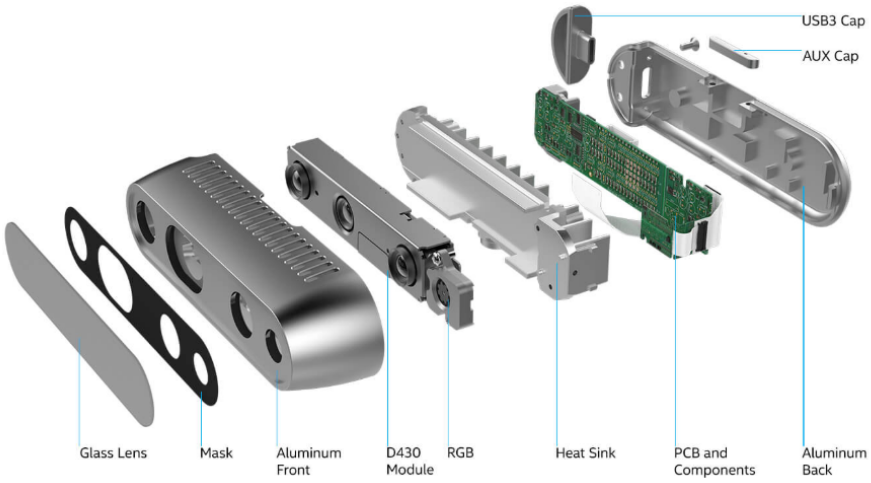


Figure B.6: The main components of the RealSense D435i ([41]).

Figure B.7: The Intel RealSense D435i

Table B.2: Technical characteristics of the Intel RealSense D435i

Features	**Description**
Length × Depth × Height	$90mm \times 25mm \times 25mm$
Use environment	Indoor and Outdoor
Image sensor technology	Global shutter, $3\mu m \times 3\mu m$ pixel size
Maximum range	Approximately 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
Depth technology	Active IR stereo
Minimum depth distance	$0.105m$
Depth Field of View (FOV)	$86° \times 57°(\pm 3°)$
Depth output resolution	up to 1280×720
Depth frame rate	up to 90 fps
RGB sensor resolution	1920×1080
RGB frame rate	30 fps
RGB sensor FOV (H × V × D)	$69.4° \times 42.5° \times 77°(\pm 3°)$

Appendix C

Coordinate frames and *tf* ROS package

In this section, there will be described how ROS is aware of the robot's position and orientation in the space.

C.1 Coordinate frames in ROS

A physical system (e.g. a robot) is usually very complex and composed by a lot of rigid components (e.g. wheels, shafts, etc.). Each system's element has well defined position and orientation in the space, that can be easily represented by a coordinate frame, which is virtually attached to the considered component. Furthermore, it is necessary to fixed a common reference frame to which the different coordinate frames should be referred. Eventually, making use of transformations between frames, it is possible to define the relationships among them.
ROS exploits the above described approach, in order to represent ROS-based robots in the space. The robot model is defined by means of an URDF file, which contain the mechanical characteristics, the used sensors and the coordinate frames attached to each component.
The two main fixed reference frames used in ROS are: *odom* and *map*. The former is mainly used for local navigation, while the latter is commonly used for global navigation purposes. Eventually, the Fig. C.1 shows an example of visual representation of frames.

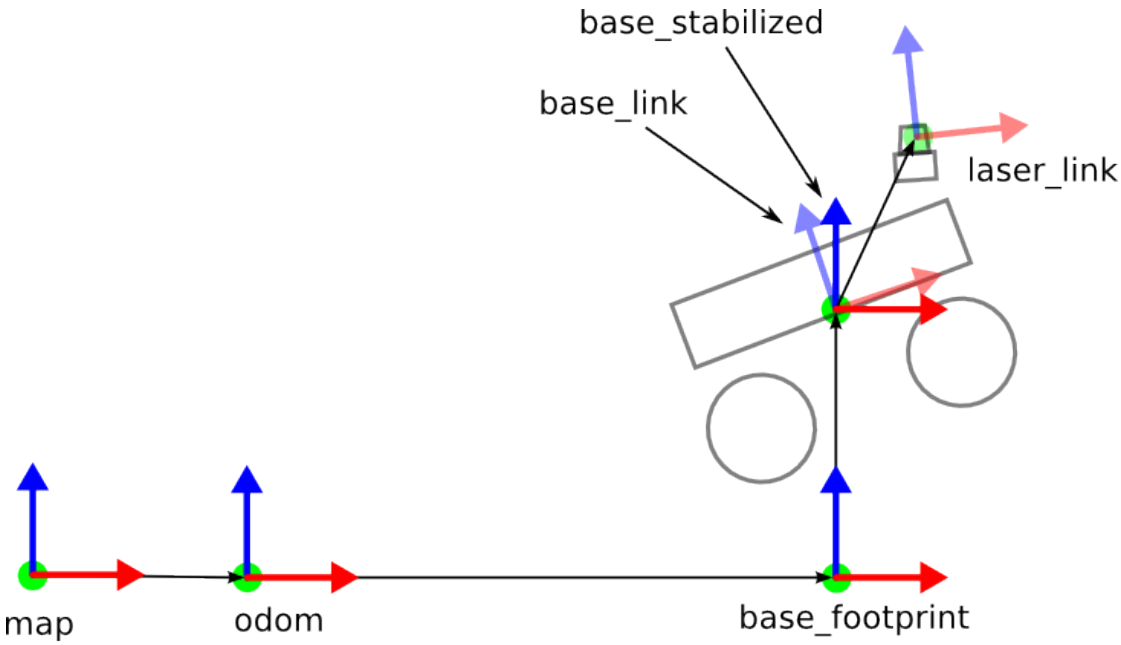


Figure C.1: Example of ROS frames

C.2 *tf* ROS package

The *tf* ROS package is highly employed in every ROS-based robot applications, in order to keep track of the relationship among the frames attached to each robot's component or sensor, exploiting the information in the URDF files. As the robot moves, the coordinate frames of components or sensors change in position and orientation and the *tf* package computes the new transformations among them. The up to date relationship among coordinate frames are represented in a tree-like structure, as can be seen in Fig. C.2.
A ROS node may obtain a particular transformation between two coordinate frames by listening on a specific topic, on which the *tf* package publishes the transformations.

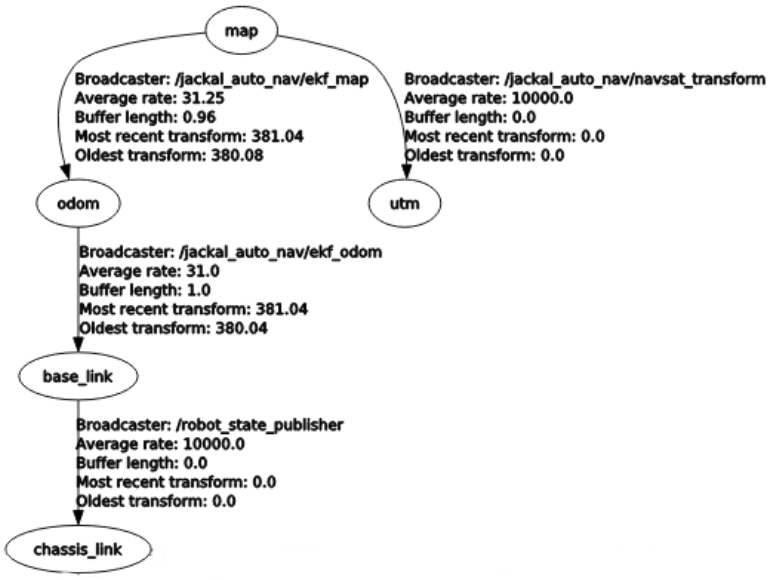


Figure C.2: Partial transformation tree of Jackal model

Bibliography

[1] Australian Centre for Precision Agriculture. *A General Introduction to Precision Agriculture*. URL: http://www.agriprecisione.it/wp-content/uploads/2010/11/general_introduction_to_precision_agriculture.pdf (cit. on p. 1).

[2] Wikipedia. *Precision agriculture*. URL: https://en.wikipedia.org/wiki/Precision_agriculture (cit. on pp. 1–3).

[3] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. «Autonomous Navigation in Vineyards with Deep Learning at the Edge». In: *Advances in Service and Industrial Robotics*. Ed. by Saïd Zeghloul, Med Amine Laribi, and Juan Sebastian Sandoval Arevalo. Cham: Springer International Publishing, 2020, pp. 479–486 (cit. on p. 3).

[4] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. «Local Motion Planner for Autonomous Navigation in Vineyards with a RGB-D Camera-Based Algorithm and Deep Learning Synergy». In: *Machines* 8.2 (May 2020), p. 27. ISSN: 2075-1702. DOI: 10.3390/machines8020027. URL: http://dx.doi.org/10.3390/machines8020027 (cit. on pp. 3, 4).

[5] Jurgen Zoto, Maria Angela Musci, Aleem Khaliq, Marcello Chiaberge, and Irene Aicardi. «Automatic Path Planning for Unmanned Ground Vehicle Using UAV Imagery». In: Jan. 2020, pp. 223–230. ISBN: 978-3-030-05800-5. DOI: 10.1007/978-3-030-19648-6_26 (cit. on pp. 4, 5).

[6] Chuck Thorpe and Hugh Durrant-whyte. «Field Robots». In: *International Journal of Pattern Recognition and Artificial Intelligence*. 2001, pp. 39–7 (cit. on pp. 6, 7).

[7] ROS.org. *ROS/Introduction*. URL: http://wiki.ros.org/ROS/Introduction (cit. on pp. 10, 11).

[8] ROS.org. *ROS/Concepts*. URL: http://wiki.ros.org/ROS/Concepts (cit. on pp. 11–13).

[9] Ami Woo, Baris Fidan, W.W. Melek, Seyed Zekavat, and R. Buehrer. «Localization for Autonomous Driving». In: (Jan. 2019), pp. 1051–1087. DOI: 10.1002/9781119434610.ch29 (cit. on pp. 14, 22, 24, 25).

[10] wikipedia.org. *Computer stereo vision*. URL: https://en.wikipedia.org/wiki/Computer_stereo_vision (cit. on pp. 18, 19).

[11] wikipedia.org. *Distortion(optics)*. URL: https://en.wikipedia.org/wiki/Distortion_(optics) (cit. on p. 18).

[12] David Marimon. «Advances in top-down and bottom-up approaches to video-based camera tracking». In: (Jan. 2007). DOI: 10.5075/epfl-thesis-3970 (cit. on p. 20).

[13] Hyunchul Lee and Okkyung Choi. «An efficient parameter update method of 360-degree VR image model». In: *International Journal of Engineering Business Management* 11 (Apr. 2019). DOI: 10.1177/1847979019835993 (cit. on p. 21).

[14] Liang Wang, Yihuan Zhang, and Jun Wang. «Map-Based Localization Method for Autonomous Vehicles Using 3D-LIDAR **This work is supported in part by the National Natural Science Foundation of China under Grant No. 61473209.» In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 276–281. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.046. URL: http://www.sciencedirect.com/science/article/pii/S2405896317300630 (cit. on p. 25).

[15] Jared Giesbrecht. «Global Path Planning for Unmanned Ground Vehicles». In: (Dec. 2004), p. 56 (cit. on pp. 25, 26, 28–30).

[16] Mehdi Mekni and Phil Graniero. «A Multiagent Geosimulation Approach for Intelligent Sensor Web Management». In: *IJDSN* 2010 (Nov. 2010). DOI: 10.1155/2010/846820 (cit. on p. 27).

[17] Fares Abu-Dakka. «Trajectory planning for industrial robot using genetic algorithms». PhD thesis. Mar. 2011 (cit. on p. 27).

[18] Rasoul Mojtahedzadeh. «Robot Obstacle Avoidance using the Kinect». PhD thesis. Aug. 2011 (cit. on p. 29).

[19] Milos Seda and Václav Pich. «Robot motion planning using generalised voronoi diagrams». In: 2008 (cit. on p. 29).

[20] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. «Path Planning for the Mobile Robot: A Review». In: *Symmetry* 10.10 (Oct. 2018), p. 450. ISSN: 2073-8994. DOI: 10.3390/sym10100450. URL: http://dx.doi.org/10.3390/sym10100450 (cit. on pp. 29, 33, 34).

[21] wikipedia.org. *D**. URL: https://en.wikipedia.org/wiki/D* (cit. on pp. 32, 33).

[22] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. «The Dynamic Window Approach to Collision Avoidance». In: *Robotics and Automation Magazine, IEEE* 4 (Apr. 1997), pp. 23–33. DOI: 10.1109/100.580977 (cit. on pp. 35, 36).

[23] David Bina Siassipour Portugal. «A study on local planning techniques for mobile robot navigation». In: (2010) (cit. on p. 37).

[24] wikipedia.org. *Global Positioning System*. URL: https://en.wikipedia.org/wiki/Global_Positioning_System (cit. on pp. 37–40, 43, 44).

[25] wikipedia.org. *Galileo (satellite navigation)*. URL: https://en.wikipedia.org/wiki/Galileo_(satellite_navigation) (cit. on p. 37).

[26] wikipedia.org. *Indian Regional Navigation Satellite System*. URL: https://en.wikipedia.org/wiki/Indian_Regional_Navigation_Satellite_System (cit. on p. 37).

[27] wikipedia.org. *Error analysis for the Global Positioning System*. URL: https://en.wikipedia.org/wiki/Error_analysis_for_the_Global_Positioning_System (cit. on pp. 41, 42).

[28] wikipedia.org. *Real-time kinematic*. URL: https://en.wikipedia.org/wiki/Real-time_kinematic (cit. on pp. 46, 47).

[29] Eric Wan and Rudolph Merwe. «The Unscented Kalman Filter». In: *Kalman Filtering and Neural Networks* (Apr. 2009), pp. 221–280. DOI: 10.1002/0471221546.ch7 (cit. on p. 53).

[30] Mohamad Shahab and Ahmad Alrefai. *Robot Self-Localization in a Known Environment*. June 2008. DOI: 10.13140/RG.2.2.30296.96001 (cit. on p. 57).

[31] Flavio Callegati, Alessandro Samorì, Roberto Tazzari, Nicola Mimmo, and Lorenzo Marconi. «Autonomous Tracked Agricultural UGV Configuration and Navigation Experimental Results». In: 2018 (cit. on p. 58).

[32] Mark A. Post, Alessandro Bianco, and Xiu T. Yan. «Autonomous navigation with ROS for a mobile robot in agricultural fields». English. In: 14th International Conference on Informatics in Control, Automation and Robotics (ICINCO) ; Conference date: 26-07-2017 Through 28-07-2017. July 2017 (cit. on pp. 58, 59).

[33] Pietro Astolfi, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci. «Vineyard Autonomous Navigation in the Echord++ GRAPE Experiment∗∗This work has been conducted under the "Ground Robot for vineyArdMonitoring and ProtEction (GRAPE)" Experiment funded by the European Commission under the ECHORD++ project (FP7-601116). http://echord.eu/grape/». In: *IFAC-PapersOnLine* 51.11 (2018). 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018, pp. 704–709. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.08.401. URL: http://www.sciencedirect.com/science/article/pii/S2405896318315271 (cit. on p. 59).

[34] ROS.org. *robot_localization wiki*. URL: http://docs.ros.org/melodic/api/robot_localization/html/index.html (cit. on p. 61).

[35] ROS.org. *move_base*. URL: http://wiki.ros.org/move_base (cit. on p. 65).

[36] ROS.org. *urdf*. URL: http://wiki.ros.org/urdf (cit. on p. 72).

[37] clearpathrobotics. *Jackal*. URL: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (cit. on p. 79).

[38] Krzysztof Kozłowski and Dariusz Pazderski. «Modeling and control of a 4-wheel skid-steering mobile robot». In: *International Journal of Applied Mathematics and Computer Science* 14 (Jan. 2004) (cit. on pp. 80–86).

[39] Novatel. *Enclosures FlexPak6*. URL: https://hexagondownloads.blob.core.windows.net/public/Novatel/assets/Documents/Papers/FlexPak6/FlexPak6.pdf (cit. on p. 87).

[40] Wikipedia. *NMEA 0183*. URL: https://en.wikipedia.org/wiki/NMEA_0183 (cit. on pp. 89, 91).

[41] Intel. *Intel RealSense Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (cit. on p. 92).