POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Thesis

DATA FUSION LOCALIZATION FOR AN AUTONOMOUS MOBILE ROBOT

Supervisors
Prof. Marcello Chiaberge
Prof. Marina Mondin

Candidate
Andrea Benedetto

Academic Year 2019/2020

Abstract

Autonomous mobile robots are able to navigate in an environment by using merely the computing capabilities they have on-board. Among the aspects of autonomous navigation, localization plays a very important and challenging role. Without knowing its position the robot would not be able to perform tasks such as path planning and motion control. Map-based probabilistic approaches are a widely used solution to the problem of mobile robots localization. However, they are not efficient when the robot is deployed outdoors due to the lack of environmental features. The aim of this document is to experiment an approach where the localization provided by a map-based SLAM algorithm is fused with the position computed by a GNSS system by means of an Extended Kalman Filter to obtain a more robust and accurate estimate. This work is part of a broader project developed by InnoTech company in collaboration with California State University, Los Angeles.

Acknowledgements

First of all, I would like to thank Prof. Mondin and Prof. Daneshgaran for having given me the opportunity to work on this thesis and for their academic and moral support during my whole stay in Los Angeles. I also want to thank Kash Olia for devoting time to help me with my work and for the guidance he provided.
Reflecting on the past 5 years of university I can't help but wonder how quickly they have passed but at the same time how many memories they left me with. In this regard, I would like to express my gratitude to all my friends I had the pleasure to meet at Politecnico, with whom I have shared many wonderful experiences that helped to build and solidify our relationships, and that I will forever cherish. Special thanks to my friend Matteo, who from day one of my life at Politecnico has been my study partner and has become one of my best friends.
Last but not least I would like to thank my family, who has supported me every step of the way and I can always count on.

Table of Contents

List of Tables

List of Figures

Chapter 1

Autonomous Mobile Robots

Robotics is one of the fastest-growing research fields of engineering. It has built its success in the industrial manufacturing world with robotic arms, or *manipulators*, programmed to perform a variety of repetitive tasks with great accuracy and speed. Nowadays, thanks to the improvements in technology, robotics has expanded outside the field of industrial robots and embraced the service robotics sector. The shortcoming of industrial robots is the lack of mobility, due to the fact that they have a fixed base. With the introduction of mobility, the landscape of robotics and its applications has changed. Mobile robots have a mobile base that enables them to move freely in the environment they are put in. Their motion is limited by the presence of obstacles and by their ability to overcome them, which in turn depends on their mechanical structure and their "intelligence". Their ability to move makes them suitable for a variety of new applications where they can help or substitute humans in carrying out tasks. *Autonomous* mobile robots (AMRs) are a branch of mobile robots that are capable of autonomously moving within their environment without human intervention. Autonomous navigation is currently one of the most studied topics in robotics since robots capable of moving without the need for supervision can produce a large number of new applications in many

fields. The numbers confirm this trend: according to BusinessWire[1], "The global autonomous mobile robots market generated $29.3 billion revenue in 2019, which is expected to reach $220.6 billion in 2030".

The evolution of sensor technologies and artificial intelligence allowed robots to be employed for more complex tasks. The robot's ability to better perceive the surroundings is enhancing the human-robot interaction and the application of robots in less structured and safe environments. For example, robotic systems are more and more often being used in robot-assisted surgeries to perform very precise operations with minimal invasion. Autonomous robots are one of the contributors of the Industry 4.0[2]. The main factor driving the use of AMRs in the industry is the increasing demand for solutions to improve the performance of production systems in terms of costs, productivity, and flexibility. These robots have numerous sophisticated sensors that enable them to comprehend the surrounding environment, allowing them to perform tasks in the most efficient way and navigate around fixed and moving obstacles successfully without human intervention. They provide flexibility in the organization of the work since they are able to adapt to the dynamics of the work environment and do not have to follow predefined paths to perform their tasks. This is the reason why in industrial settings AMRs are replacing the automated guided vehicles (AGVs), whose motion is restricted to predefined routes.

Service robotics is also benefiting from the application of AMRs. Service robots relieve humans of performing tasks that may be dangerous, time-consuming, dull, or repetitive to let them focus on more cognitive functions. A general definition given by the ISO 8373 states that a service robot "performs useful tasks for humans or equipment excluding industrial automation applications". So the service robots category includes a wide spectrum of applications, ranging from vacuum cleaning and lawn-mowing robots to robots performing outer space and deep-sea exploration.

1.1 The interdisciplinarity of autonomous robots

The architecture of an autonomous robot is the result of the integration of different bodies of knowledge[3]. The foundations of autonomous mobile robotics comprise the fields of locomotion, perception, localization and navigation. The capability to move requires robots to have a suitable mechanical structure, whose design requirements depend on the environment they navigate and the applications they are utilized for. For example, a ground robot may be wheeled, if the motion is enabled through wheels, or legged, if it has legs. Wheels are suitable for environments where the ground is hard and homogeneous, while legged robots are better for navigating over non-smooth terrain and small obstacles. To design a locomotion system an evaluation of the motion capabilities of the robot is required, and its kinematic and dynamic behaviors have to be modeled. Perception is the ability of the robotic system to acquire information about its surroundings. The environment is perceived through sensors. Different types of sensors are employed for different working environments as well as for specific applications. Perception is not just sensing the environment, but also interpreting the sensor's measurements. Sensor data is processed and manipulated to extract relevant features that are fundamental for the robot's correct functioning. This is where artificial intelligence plays its role: signals analysis, computer vision and machine learning techniques are applied to analyze sensor measurements and retrieve information from them that are meaningful to the robot's task. Perception is a vital aspect of autonomous mobile robots. Without sensors, tasks such as localization and navigation would not be possible. Sensors are identified in two main categories:

- proprioceptive sensors, which measure variables internal to the robot system,

such as wheel speed and position, its orientation, joint angles, etc. Examples of proprioceptive sensors are encoders, gyroscopes, compasses, etc.

- exteroceptive sensors, which measure physical quantities relative to the environment that the robot interacts with, like the distance from an obstacle, the light intensity, and so on. Examples are radars, sonars, cameras, etc.

Autonomous navigation is currently one of the most studied topics of robotics. In order to autonomously move from one destination to another, the robot needs to know essentially two things: where it is at the present moment and how it can reach the goal position. To achieve this it relies on four pillars (depicted in figure 1.1): perception, localization, cognition and motion control. The aspect that is connected the most to the concept of navigation is cognition. The cognitive part of a mobile robot is the "brain" of the robot, which is the system in charge of taking decisions regarding the robot's motion to reach its destination. Decision-making involves identifying a trajectory for the robot to follow and computing the control inputs for the actuators. The problem of determining the optimal collision-free path from the robot starting position to the goal position is known as *path planning*. The challenge of navigation lies in the execution of a motion plan that is successful in spite of unexpected events that hinder the robot in its intent to reach the destination. Autonomous robots navigate in environments that are non-static and that change with time. Based on the knowledge of the surroundings provided by sensors, the cognition system must be able to react to environment changes and locally deviate from the robot's plan to still reach the goal. A motion control system coordinates the input data for the actuators to keep the robot's motion as close as possible to the one planned. Today, much focus is put into research and development of improved sensors and more intelligent control systems to make robots more autonomous. Sophisticated sensors can perceive details of reality that

allow handling more complex scenarios, provided that the intelligence of the robot is flexible enough to cope and adapt.

The main components constituting the architecture of autonomous mobile robots have been introduced except for localization, which will be discussed in more detail in the next section. In the following discussion we are going to consider ground

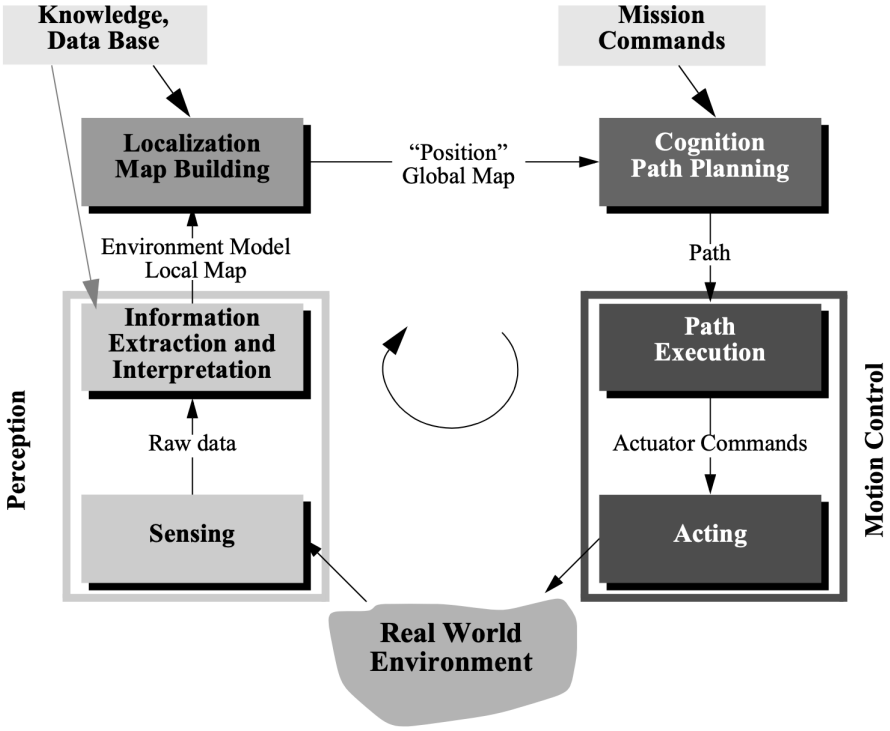


Figure 1.1: Building blocks of autonomous robots

wheeled mobile robots, since they are subject of this thesis.

1.2 Localization

An entire section is dedicated to localization since it is a topic of major importance in the development of this work. Localization is the problem of estimating the robot position and orientation inside an environment. The term estimation is

purposely used to indicate that the pose (position and orientation) of the robot cannot be measured directly, but has to be inferred from the sensors readings. To pursuit the trajectory computed by the path planner it is necessary to know in real time the configuration (position and orientation) of the robot with respect to a world-fixed frame. Due to its challenging nature, localization has been subject to a lot of research in the past decade, which lead to the development of several methodologies.

Pose estimation techniques can be categorized in several ways, but the most conventional one is probably to divide them between *relative localization* methods and *global localization* methods[4]. The relative localization approach, also known as *dead reckoning* or *deduced reckoning*, consists in the estimation of the current pose in relation to a known initial pose. At each time step the current pose is computed by adding to the previous pose the measured displacement from that pose. So, starting from an initial point, the pose at a certain time instant is the result of the integration (or summation in case of discrete time) over time of the motion increments that the robot performed up to that moment. The weakness of this localization approach is durability, because, due to the accumulation of measurement errors, the estimate becomes more and more unreliable over time. One of the relative navigation methods is *Odometry*. Odometry uses wheel encoders to measure the rotations of the wheels, from which the velocity of the robot is inferred. Wheels and robot velocities are in fact linked by the kinematic model of the robot. Once the velocity is computed, the movement of the robot between two consecutive locations is obtained through the relation $d = vt$, where t is the time length between successive measurements. If we consider a two dimensional case, the robot's pose is determined by three variables: its position coordinates (x,y) and its heading θ (see figure 1.2). Given the configuration of the robot at time $t-1$, the new configuration (x, y, θ) at the next time instant t is computed

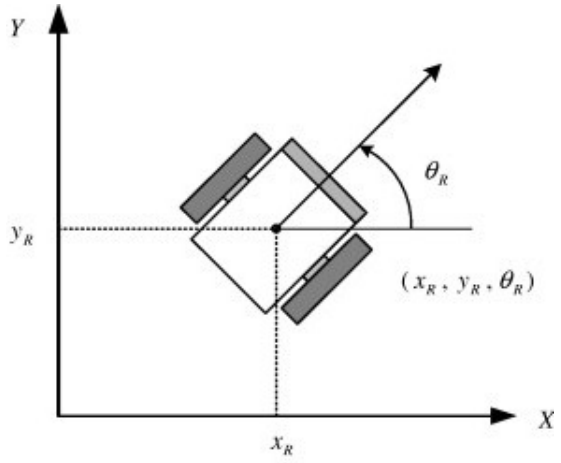


Figure 1.2: Configuration of a differential drive robot in two dimensions described by the coordinates (x_R, y_R, θ_R).

according to odometry as:

$$x_t = x_{t-1} + vtcos\theta_{t-1}$$

$$y_t = y_{t-1} + vtsin\theta_{t-1}$$

$$\theta_t = \theta_{t-1} + \omega t$$

Another relative localization method is the *Inertial Navigation*. Inertial navigation systems directly measure linear acceleration and angular velocity of the robot using accelerometers and gyroscopes. By double integration of linear acceleration and single integration of angular velocity, an estimate of the pose can be obtained.
As anticipated before, these methods suffer from errors accumulation which causes the estimated pose of the robot to drift over time from its true value. Individual estimates of the robot configuration are inaccurate because measurements are subject to systematic and non-systematic errors. These errors come from sources (ie. sensors, environment, etc) which have not been modeled entirely, may be because of the impossibility to create a model due to the unpredictable nature of the error, or because of the lack of knowledge about the source. Anyhow, this results in inaccuracies between the intended motion of the robot, the true one and the

one inferred from measurements. Systematic errors are usually due to incomplete kinematic models and sensor measurement unknown biases, while non-deterministic errors include wheels skidding, ground irregularities, sensor noise, etc. Because the current pose is obtained by integration of the measured displacements, and hence of their measurement errors, the pose estimate error accumulates and grows over time. Therefore, relative localization methods are applicable only for short-term pose estimation from an initial pose, before becoming too unrealiable.

In order to maintain a meaningful pose estimate in the long run, relative estimation methods are used in combination with absolute localization ones. Absolute localization methods determine their pose by observing their relative orientation and position with respect to external references known as *landmarks*. These "markers" may be natural if they are part of the environment, or artificial if they are placed on purpose for localization. Moreover, artificial landmarks may be active or passive. Active landmarks emit signals to be easily detected by the robot (e.g. infrared transmitters), while passive landmarks rely on the ability of the robot sensors to detect and recognise them (e.g. QR-codes). Of course natural markers are more sustainable since they do not require additional work, while artificial ones offer more robustness and fast localization but at the price of maintenance and operational costs. Examples of natural landmarks in the context of mobile robotics are lines on the ground, doors, ceiling lights, and so on.

In order to estimate the pose of the robot, the position of the landmarks has to be known a priori. By measuring angles and distances from multiple landmarks, the robot can determine its pose. The techniques that are used for this kind of localization are *triangulation* or *trilateration*[5]. Both of these methods find an estimate by solving a system of equations where the unknowns are the robot's position and orientation coordinates, while they differ in the formulation of the equations. To explain the two methods we consider a two-dimensional scenario, where the

robot pose is given by the set of coordinates (x, y, θ). The idea of triangulation is to measure the angles to at least three markers and use trigonometric laws to connect the position of the robot to the position of the landmarks. Figure 1.3 shows the case when the angles to exactly three landmarks are measured. The pose estimate can be found by solving the following system of three equations with three unknowns:

$$tan(\alpha_1 + \theta) = \frac{y_{m1} - y}{x_{m1} - x}$$
$$tan(\alpha_2 + \theta) = \frac{y_{m2} - y}{x_{m2} - x}$$
$$tan(\alpha_3 + \theta) = \frac{y_{m3} - y}{x_{m3} - x}$$

where the angles α_j (j = 1,2,3) are the measured angles to landmarks in position (x_{mj}, y_{mj}). Since these trigonometric equations are nonlinear, the algebraic system

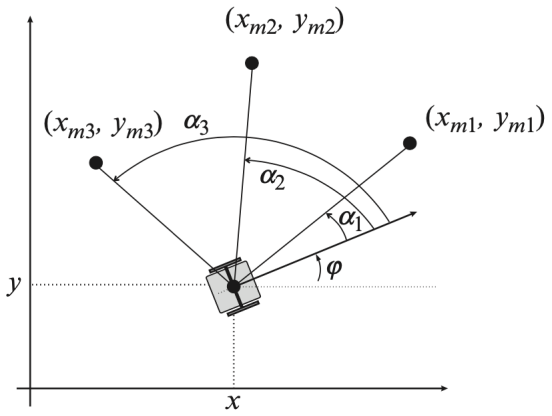


Figure 1.3: Robot localization using triangulation of three landmarks

can be solved either by linearization or by numerical techniques such as the Newton-Raphson technique. In practice three equations are never enough because measurements are noisy, and noise introduces further unknowns. In this case more landmarks are used to obtain an overdetermined system, which can be solved by iterative techniques like the least-squares method.

Triangulation is used in scenarios where it is not possible to measure distances accurately. If that is not the case, trilateration can be applied. Trilateration has the same working principle of triangulation, but uses distances instead of angles to relate the position of a marker to the position of the robot. Moreover, with this method it is only possible to compute the position of the robot. Measuring the distance d_i to three different landmarks, we can write the following system of equations:

$$d_1^2 = (x_{m1} - x)^2 + (y_{m1} - y)^2$$
$$d_2^2 = (x_{m2} - x)^2 + (y_{m2} - y)^2$$
$$d_3^2 = (x_{m3} - x)^2 + (y_{m3} - y)^2$$

This is again a determined non-linear system, graphically shown in figure 1.4. For

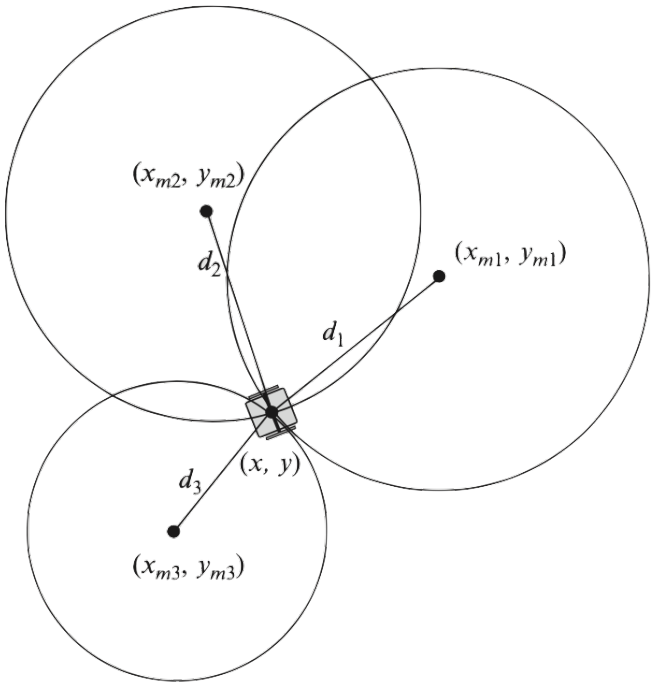


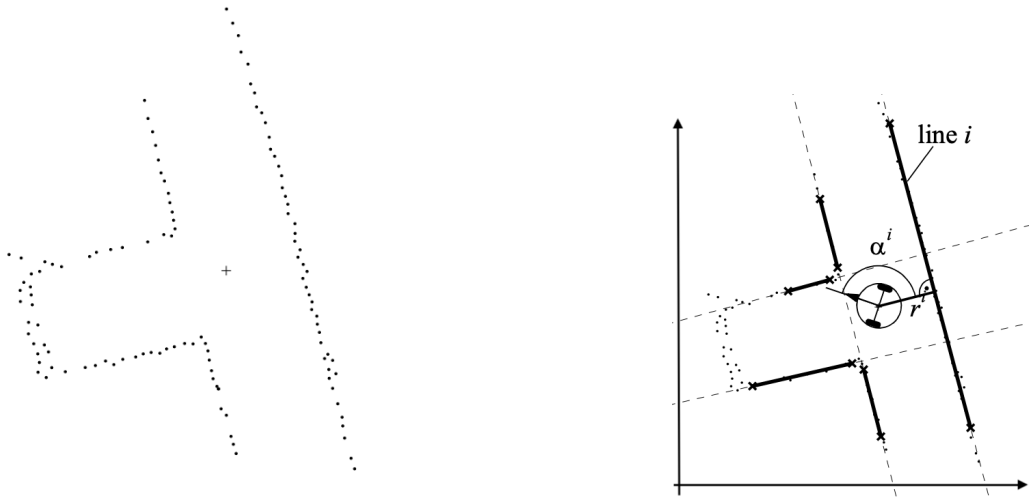
Figure 1.4: Robot localization using trilateration of three landmarks

the same reasons we discussed for trilateration approach, in reality this is never

the case and we will always need to solve an overdetermined system. A well known trilateration localization system is the *Global Navigation Satellite System* (GNSS), where the landmarks are artificial active landmarks such as satellites. GNSS is discussed in detail in chapter 2.

The last and most used absolute localization method is *Map Matching*, or *Map-based localization*, where the robot determines its pose in relation to a environmental map. The robot needs to have a global map stored in memory that contains the parts of the environment that it has already visited. By matching its map with the measurements coming from its exteroceptive sensors, the robot is able to figure out its pose. The map can be provided by an user or built by the robot. In the latter case, the generation of the map can be done either through an offline learning phase, where the robot is guided manually through the environment, or online while the robot is autonomously navigating. Autonomous map building is known as the *Simultaneous Localization and Mapping* (SLAM) problem, where the robot builds a map while simultaneously localizing inside it. This is a very challenging task and has been an intensively studied topic in mobile robotics as it allows the robot to navigate through an unknown environment while mapping it.

An environment is basically a collection of natural landmarks, or simply objects. A map is a representation of the environment based on some of its features that describe the present objects. By scanning the environment through its sensors, the robot is able to gather a collection of points (or vectors) representing the position of environmental features. Examples of features are lines (figure 1.5), segments, corners, or more specific patterns that can be extracted by processing sensor measurements such as distances and angles. The map is formed by the set of vectors representing the coordinates of the features detected in the environment in a global coordinate system. While moving, the robot collects data from its exteroceptive sensors and creates a local map of the environment. To find its



(a) **(b)**

Figure 1.5: Figure (a) depicts the raw measurements acquired by a laser scanner, while figure (b) shows the extracted lines features.

location, the robot compares the local map to its global map until it finds a suitable match. In other words, by comparing the features observed and the features of a known map, the robot pose can be estimated.

Map-based localization is widely used in mobile robotics because, in this context, localization implies more than knowing the robot absolute pose. Indeed, obtaining an absolute estimate of the robot pose may not be sufficient for navigation because the robot needs to find a path to reach its destination, and to do that it requires a map of the environment. Therefore, the problem of localization becomes not just the problem of determining the robot absolute pose in space, but of identifying its pose relative to a map of the environment.

1.2.1 Uncertainty

The difficulties of localization are due to the presence of uncertainty in the robot-environment system. The robot's sensors and actuators play a key role in all the localization techniques. However, they are also the cause of the challenges

of localization because of their inaccuracy and incompleteness[6]. The working environment of mobile robots is also affected by uncertainties, which are usually lower in structured environments and higher in dynamic unstructured ones environments. Therefore, the development of the autonomous mobile robots must take into consideration the issue of uncertainties coming from various sources.

Sensors inaccuracies are due to two phenomena: sensor noise and sensor aliasing. Sensor noise is an apparently random error introduced in sensor measurements, caused by environmental features that cannot be observed by the robot. For example, for a vision system environmental variations like illumination conditions, blooming, blurring, and so on are a source of noise; in sonar systems, reflective environments can cause multipath interference effects. Sensor noise limits the consistency of sensor readings in the same environmental state, and therefore sensors may at times provide irrelevant information about the measured quantity. The second source of imprecision of sensors is aliasing, that is the fact that sensor readings are not unique to a single environmental state. For example, an infrared range sensor measures the distance of an object in a certain direction, but does not provide information about the material of the object like color, texture or hardness. This means that there is not a one-to-one mapping between environmental states and sensor measurements. In other words, the same sensor values can be triggered by a multiplicity of environmental states. This implies that the robot is not able distinguish among these states through a single sensor measurement. Even in case of noise-free sensors, the available information from a single sensor reading is generally not sufficient to identify the robot pose. The solution to both problems is to take multiple readings into account and apply sensor fusion techniques in order to minimize the effects of noise and aliasing and increase the accuracy of the robot pose estimate.

Another source of inaccuracy comes from the non-deterministic motion of the

robot. An action performed by the robot may end up with different possible results. As already introduced in dead-reckoning localization, the act of moving increases the uncertainty on the pose estimate. This is because the actual motion of the robot is not exactly the planned one. The reasons lie in the non-idealities of the actuators, in the incompleteness of the kinematic and dynamic models of the robot, and in environmental factors that are not taken into account (e.g. slope of the road). What relative localization methods try to do is to use the information captured by sensors to compensate for the uncertainty introduced by noisy actions. Of course also sensors are affected by noise, so at the end the intended motion, the estimated motion and the actual motion are all slightly different from one another.

1.2.2 Probabilistic localization

The uncertainty associated with sensor measurements and robot motion affect the accuracy of the pose estimate, making the localization problem more difficult. The idea of probabilistic localization is to deal with uncertainty by modeling it through probability[6]. Instead of identifying a single best estimate for the robot pose, probabilistic approaches use a probability distribution over the space of all possible poses; in other words, they assign a probability to each possible robot pose. Therefore the goal of localization becomes not only the estimation of the robot configuration, but also of the uncertainty associated with the estimate, that is modeled by means of a probability density function known as *Belief*. The term belief indicates precisely that the robot believes to have a certain configuration. The most probable robot pose corresponds to the highest peak of the probability distribution.

Probabilistic estimation methods follow an iterative process (shown in figure 1.6) that is composed of two steps:

- *Prediction* step, where the robot uses relative localization methods to make a

prediction of its future configuration.

- *Measurement update* or *Correction* step, where the robot uses absolute localization methods to correct the configuration estimated in the prediction step.

Basically the robot makes use of information from its proprioceptive sensors to predict its future pose, and from exteroceptive sensors to update the predicted pose. In general the prediction phase increases the uncertainty of the robot belief about its pose, since the robot motion is to some extent non-deterministic and measurements errors are accumulated. On the contrary, the correction phase typically reduces the uncertainty because an absolute position measurement is provided by sensors.

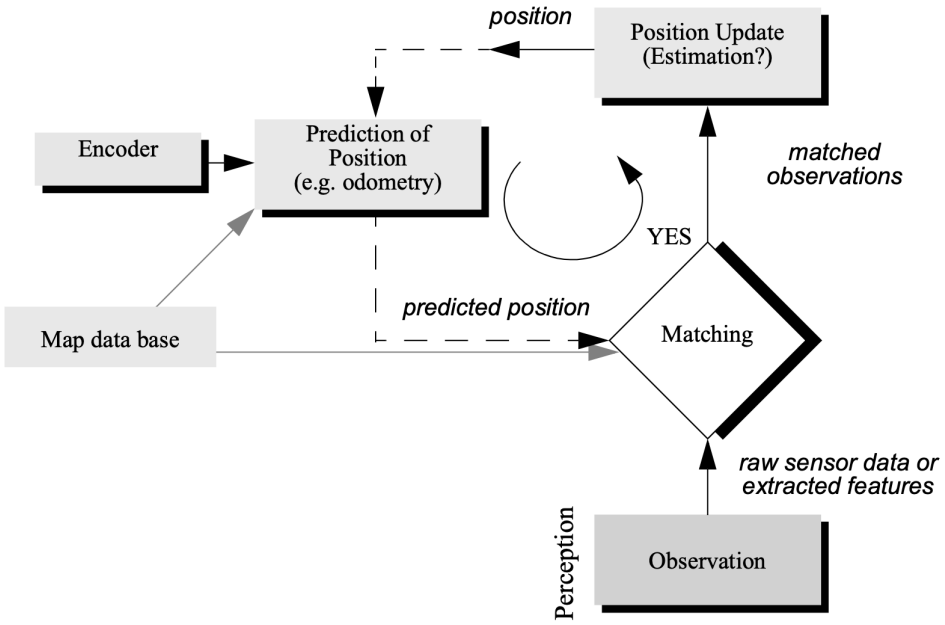


Figure 1.6: Robot localization process

The localization problem is part of the bigger family of state estimation problems. The most general probabilistic approach to solve state estimation problems is given by the *Bayes filter*. The Bayes filter calculates recursively the probability

distribution of the state from sensors observations. In the context of mobile robot localization, the state corresponds to the robot pose and probabilistic state estimation is performed by *Markov localization*, which is the straightforward application of the Bayes filter to the localization problem[7]. Probabilistic localization methods are described in chapter 3.

1.2.3 Sensor fusion

Integration of information coming from various sources, known as *sensor fusion* or *data fusion*, is a key factor for robust localization. Sensors are subject to noise and, occasionally, failures, which make the problem of perception critical. Employing multiple sensors provides reliability and redundancy, reducing the effects of noise and aliasing. Moreover, the use of heterogeneous sensors helps the robot grasp a more complete view of its surroundings[8]. Different sensors may detect different features of the environment, resulting in a richer body of information that allows the robot to detect and recognize objects better. Through the integration of multiple sensors data, the quality of localization is expected to improve in terms of accuracy and confidence about the result obtained. A robust sensor fusion technique should be able to provide consistent data, despite the presence of uncertainties of sensors and environment. Probabilistic localization techniques previously introduced allow to perform sensor fusion in the correction step, where data from multiple sensors can be used to update the prediction estimate. More about this matter is discussed in chapter 3.

Chapter 2

Satellite Positioning

Satellite positioning systems (like GPS) allow to easily determine the absolute location of a receiver anywhere on the earth. The development of this technology over the years has reached a point where the position can be determined with a high degree of accuracy, in some cases below the centimeter. Moreover, this localization method does not need previously acquired knowledge about the local environment, which makes it easy to use and applicable in any working environment. The use of a satellite navigation system may seem to be the perfect solution to the problem of robot localization, given also the fact that satellites nowadays give coverage of the whole earth's surface. However, few drawbacks make the use of satellite position systems alone not sufficient for autonomous navigation. First of all, mobile robots require a map of the environment in order to perform path planning, and they need to self-localize on that map. Furthermore, GPS-like technologies do not function well indoors and in obstructed areas, where the satellite signals are blocked by obstacles (e.g. buildings, trees, etc). For these reasons, in the framework of mobile robotics, satellite navigation systems are usually utilized in combination with other localization techniques. Particularly, they offer an advantage in large environments where landmarks run low, where they can serve the purpose of giving a measure of

the robot position in support of that given by map-based probabilistic algorithms, which may experience difficulties in converging to a solution.

2.1 Global Navigation Satellite System

When we talk about localization systems that make use satellites we usually mention the Global Positioning System (GPS). GPS, also known as NAVSTAR GPS, was the first satellite system launched in space, and it is owned by the United States government. However, it is not the only satellite navigation (SATNAV) system operating in the world. Indeed, different countries nowadays have their own SATNAV system.

The correct term to refer to any SATNAV system is *Global Navigation Satellite System* (GNSS). GNSS is used to indicate the collection of all satellite positioning systems as well as any individual NAVSAT system. Following the second definition, we can say that GPS was the first operational GNSS system; for this reason, the two terms are often used interchangeably in the common language. Besides GPS, GNSS currently includes other satellite navigation systems, such as the Russian GLONASS, the European Union's Galileo and China's Beidou.

2.1.1 GNSS architecture

Each GNSS system is a collection of satellites denominated *constellation*, which orbit at around 20,000 km in the atmosphere and are designed to cover specific regions of earth. Each satellite travels on a specific orbit and broadcasts signals towards earth which allow to identify it.

GNSS satellites are aware of their orbit ephemerides and clock very accurately. To manage the satellites' orbits and send communications to them, a network of control stations is present on earth. Stations receive the satellites' signals, analyze

them and transmit proper orbit and time corrections to the satellites. This is necessary to adjust their orbit's parameters and clocks to maintain the best possible accuracy. Ground stations are equipped with cesium clocks, which are even more accurate than satellite clocks: they have an accuracy of around ± 5 parts in 10^{12}, that is, they lose or gain 1 second every 100,000 years.
The user equipment is in charge of capturing the signals, process them, and derive the location information. The two components that make the job are antennas and receivers. An antenna receives the signals transmitted from a satellite and passes them on to the receiver. They come in different sizes and varying performance depending on the applications they are used for. The receiver analyzes the signal to compute its own position. Since every GNSS satellite uses a specific unique code to modulate the signal sent, receivers must know the code of every satellite they are interested in in order to understand their signals.

2.1.2 GNSS signals

GNSS systems use radio signals at frequencies between 1 and 2 GHz. The frequency band used by GNSS systems is called L-band; in particular L1 refers to 1 GHz and L2 to 2 GHz. GNSS signals are sinusoidal EM waves, called *carriers*, which are subject to a complex modulation that allows them to carry information. By definition, modulation is the variation of one or more properties of the carrier wave (frequency, amplitude and phase) through a modulating signal, which contains the information data to be transmitted. Since all constellations use the same frequency spectrum, the signals must not interfere with one another in order to be recognizable by the receiver. To do so, GNSS signals use a transmission scheme called CDMA (Code-division Multiple Access). This technique allows several transmitters to send informations simultaneously on the same band of frequencies without causing interference. To distinguish between signals of different satellites, carriers are

modulated by a pseudorandom code. This code is a binary code that repeats every millisecond and is composed of 1,023 bits, so has a transmission rate of 1.023 Mbps. The pseudorandom code, also known as *Coarse Acquisition* (C/A) code or *Pseudo random noise* (PRN) code, looks random but it is actually a deterministic sequence of bits. It is unique of each satellite, and allows receivers to distinguish between signals sent by different satellites.
A receiver must know the PRN of a satellite in order to understand the data it transmits. Receivers synchronize with the signal to be able to recover the information it contains through correlation of pseudorandom codes. Codes are created in such a way that they have very little cross correlation, and high auto-correlation. Thus, different codes will have low correlation, while same codes have very high correlation. This helps the receiver identify the satellite from which the signal was originated. The receiver creates locally a replica of the codes of the satellites it wants to track, and by comparing it with a received signal it finds the sender satellite. Obviously the receiver has to know in advance the codes of the tracked satellites in order to reproduce them.

Being carried on the carrier wave we have also the navigation data. This is the binary code that contains satellite information necessary to determine the position of the receiver. In particular, the time, orbit and status information of the transmitting satellite are encoded in a binary message which further modulates the carrier wave. A navigation message is composed of 1500 bit that are transmitted at a rate of 50 bps, so every bit takes a period of 20 milliseconds. This means that, for every navigation bit, 20 pseudorandom codes are transmitted.
Modulation of the carrier wave is produced using the Binary Phase-Shift Keying (BPSK) modulation, which associates to the binary values 1 and 0 two different phases of the carrier waves, separated by 180°. Since the pseudorandom noise is a binary code, the carrier wave is flipped by 180° every time there is a transition

between a 1 and a 0.

Figures 2.1 and 2.2 show the two modulations applied on the carried wave, one by the C/A code and one by the navigation data.

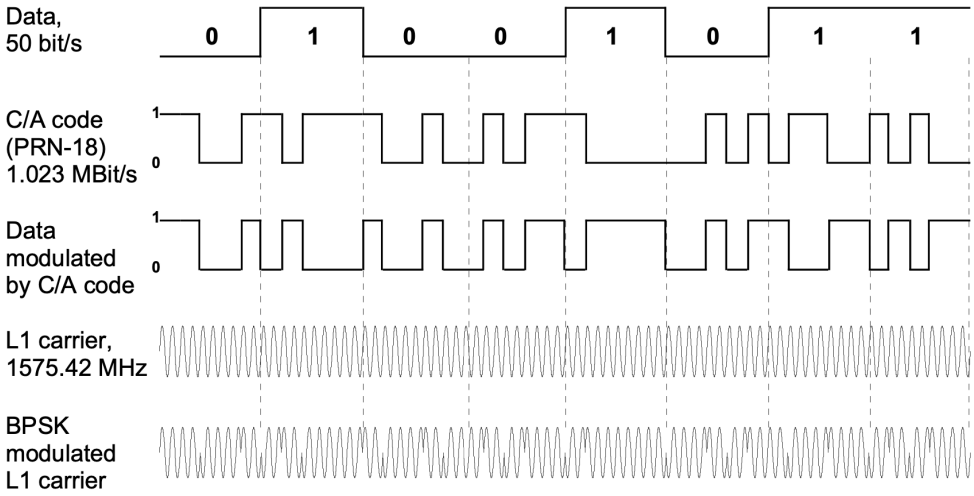


Figure 2.1: Structure of a GNSS signal

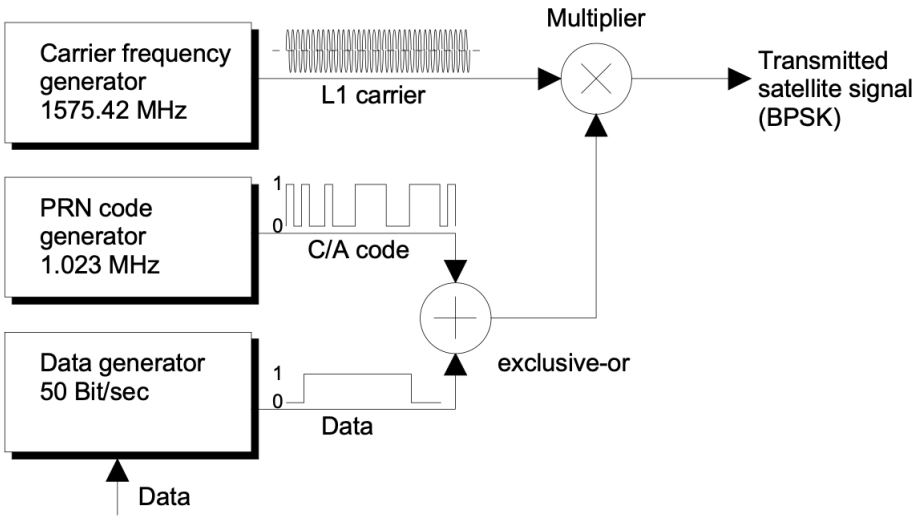


Figure 2.2: Block diagram of GNSS signal generation

2.1.3 Position computation

GNSS positioning is based on a technique called *Trilateration*. The position of the receiver is determined using the distance from known points in space. A minimum number of four satellites' signals is required by the receiver to determine its position (also know as *fix*). A higher number of tracked satellites will improve the fix solution. To explain the concept of trilateration we make a simple example in the 2-dimensional space. If we can measure the distance of the receiver from a satellite 1, of which we know the position in the reference coordinate frame, we can conclude that the receiver is somewhere on the circle with center in satellite A and radius equal to the measured distance. If we take a second measurement from a satellite 2, we now know that the receiver is somewhere on the circle centered in 2. But since the receiver has to be in points common to both circles, then it is in either of the two points where the circles intersect. Getting a third measurement from a satellite 3 we get a third circle. Supposing we are in an ideal error free situation, the three circle intersect in a single point (see figure 2.3). We have in this way identified the position of the receiver. Actually, even with only two measurements we could identify the correct position of the receiver, since one of the two points of intersection the circles can usually be discarded because we know it is not a feasible solution.

If we consider the real 3-dimensional case, we would need at least three satellites in an ideal error free setting. This is because, with three dimensions, a distance from a satellite generates a sphere of possible locations of the receiver. Two spheres intersect in a circle, and three spheres in two points, one of which can be discarded by making an heuristic reasoning. So, if we know the distance (which is referred to as *range*) from three satellites and we know their position, we can determine the position of the receiver anywhere on earth. However, we are not in an ideal world, and the taken measurements contain errors. For this reason, the number of

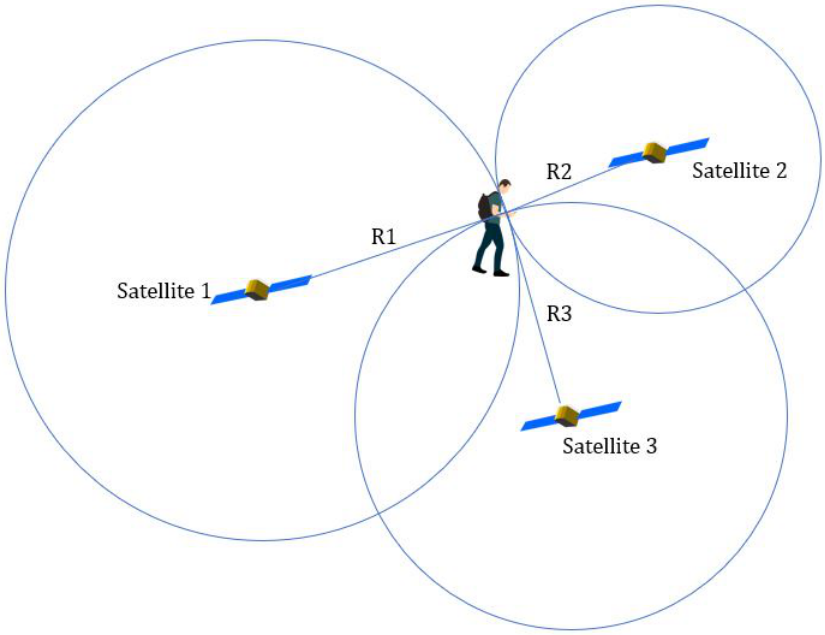


Figure 2.3: 2-dimensional trilateration

satellites a receiver needs to track is at least four, as shown in figure 2.4.

To understand why, we first need to understand how we measure the distance from a satellite, and how we know the position of the satellites. The computation of the range from a satellite is done by the receiver using a simple dynamic principle: space = velocity $\times$ time. The idea behind it is to compute the propagation time of the signal, that is the time that the signal takes to travel from the satellite to the receiver, and multiply it by the speed of light to obtain the distance travelled. Recall in fact that, since the signal is an EM wave, it moves at the speed of light. The propagation time is computed by taking the difference between the time the signal reached the receiver and the time it left from the satellite. The departure time is encoded inside the signal, and it is very accurate thanks to the high precision clocks onboard the satellite. Once the receiver captures a signal, it registers the time of arrival and computes the propagation time. The position of the satellite at the time of transmission is also carried in the signal.

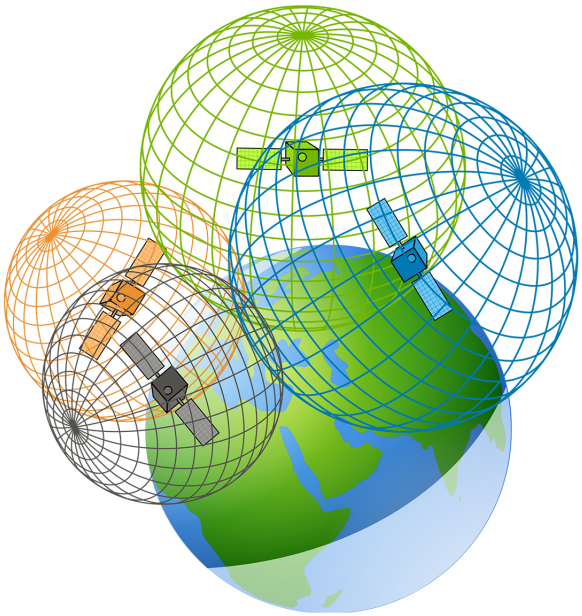


Figure 2.4: GNSS positioning trilateration

The arrival time is computed through correlation of PRN signals (see figure 2.5). For every signal the receiver captures, a correlation process takes place. The receiver knows the PRN code of a collection of satellites, and for each of them generates a replica. These codes are then shifted in time until one of them reaches the maximum correlation with the processed signal. Since the PRN signal is unique of a satellite, we have identified the transmitter. Now that we know the source, we can reconstruct the navigation message encoded in the signal. Moreover, the time shift of the PRN code replica that the receiver applied for the correlation procedure is used to calculate the arrival time of the signal. So the PRN code purpose is not only to identify the satellite from which the signal was generated, but also to allow the receiver to compute the travel time.

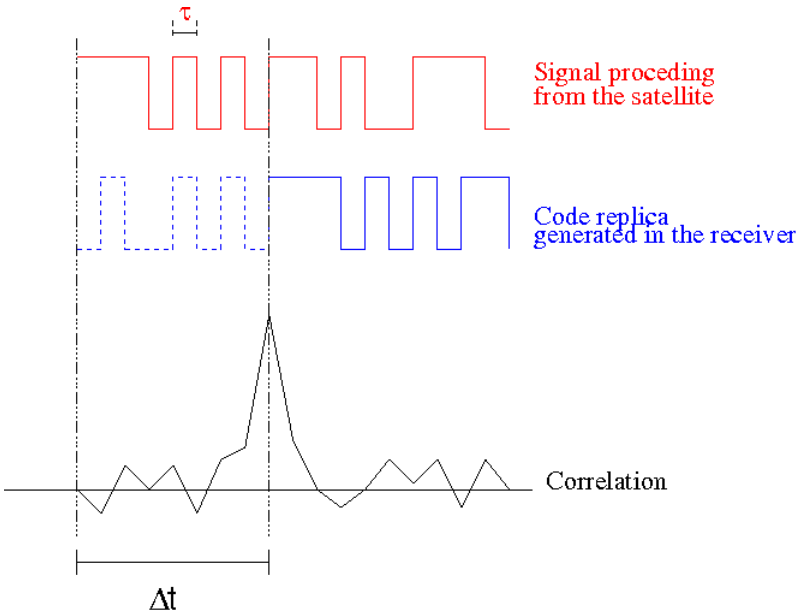


Figure 2.5: Correlation process of PRN codes

2.1.4 Pseudorange

As previously stated, the GNSS signal is subject to different sources of errors: atmospheric delays, clocks imprecision, multipath. For this reason the computed range from a satellite is not the true range, and is referred to as *pseudorange*. The fix computed through the trilateration procedure contains an error proportional to all the errors affecting the pseudorange. From a visual point of view, the three spheres computed do not intersect in the true position. This is easily visible in 2-dimensions: the position where two circles intersect is wrong by a quantity proportional to the pseudoranges errors with respect to the true ranges. If we now draw a third circle, we expect it to intersect the computed position, but it won't because both the position and the third pseudorange contain an error. Instead, the three circumferences determine an area of points where the true location lies, as shown in figure 2.6.

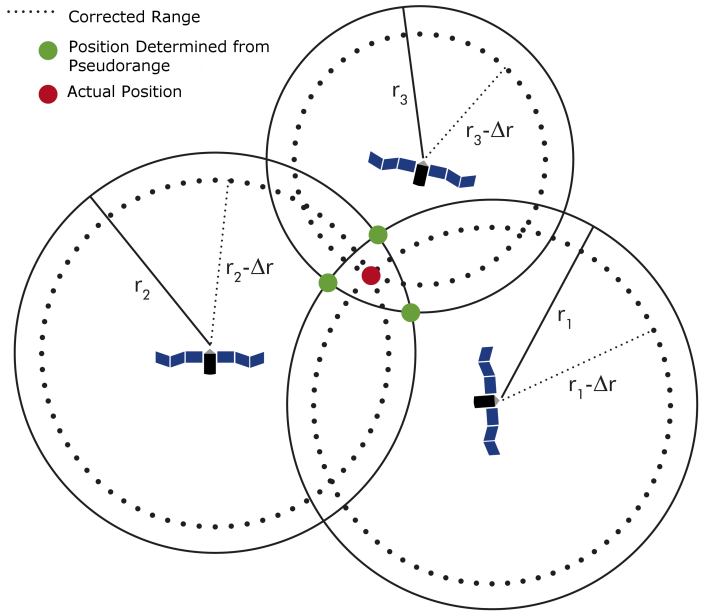


Figure 2.6: 2-dimensional pseudorange trilateration

A simple solution is to take into consideration only the error common to all pseudoranges, which is the time error coming from the non-synchronized satellite and receiver clocks, and neglect all other sources of error. Time has a key role in SATNAV systems. A universal timing reference is provided by the *Universal Coordinated Time* (UTC). Moreover, every SATNAV system has an internal time scale denoted as *system time*. The clocks onboard satellites of a particular constellation are synchronized with the system time of the SATNAV system they belong to. Even though satellites' clocks are very precise, they still may contain an error from the system time. For this reason, control stations continuously monitor satellites' clocks to correct them and keep them synchronized to the system time. On the other hand, the receiver clock is not synchronized with the system time.

System time is the reference time used by receivers tracking satellites of a certain SATNAV system to compute range measurements. Both receiver's time and satellite's time have a bias error from system time. For the following development,

however, we consider the satellite clocks' offsets compensated for by the correction sent by the SATNAV systems monitoring stations, so the only clock error we consider is the receiver's clock offset. Since this offset is the same for all the pseudorange measurements, we can introduce this error as unknown variable along with the three Cartesian coordinates of position. To solve the system of four unknowns we need four equations, therefore a fourth pseudorange measurement from another satellite. In such way, we obtain the following system of four unknowns and four equations:

$$
\begin{aligned}
d_1 &= \sqrt{(x_1 - x_p)^2 + (y_1 - y_p)^2 + (z_1 - z_p)^2} + c \cdot t_p \\
d_2 &= \sqrt{(x_2 - x_p)^2 + (y_2 - y_p)^2 + (z_2 - z_p)^2} + c \cdot t_p \\
d_3 &= \sqrt{(x_3 - x_p)^2 + (y_3 - y_p)^2 + (z_3 - z_p)^2} + c \cdot t_p \\
d_4 &= \sqrt{(x_4 - x_p)^2 + (y_4 - y_p)^2 + (z_4 - z_p)^2} + c \cdot t_p
\end{aligned}
$$

The equations of the system are non-linear, since they are quadratic functions of the unknown coordinates x_p, y_p, z_p. One way to solve the system is to linearize it around a point that we know to be close to the real solution. We denote the approximate position as $(\hat{x}_p, \hat{y}_p, \hat{z}_p)$ and we expand the four equations around this position with the first order Taylor expansion. What we obtain is a linear system in function of the coordinates position offset Δx_p, Δy_p and Δz_p:

$$
\begin{aligned}
\Delta d_1 &= a_{x1}\Delta x_p + a_{y1}\Delta y_p + a_{z1}\Delta z_p + c \cdot t_p \\
\Delta d_2 &= a_{x2}\Delta x_p + a_{y2}\Delta y_p + a_{z2}\Delta z_p + c \cdot t_p \\
\Delta d_3 &= a_{x3}\Delta x_p + a_{y3}\Delta y_p + a_{z3}\Delta z_p + c \cdot t_p \\
\Delta d_4 &= a_{x4}\Delta x_p + a_{y4}\Delta y_p + a_{z4}\Delta z_p + c \cdot t_p
\end{aligned}
$$

Once we compute the unknowns, the real position's coordinates are computed as:

$$x_p = \hat{x}_p + \Delta x_p$$
$$y_p = \hat{y}_p + \Delta y_p$$
$$z_p = \hat{z}_p + \Delta z_p$$

This solution is of course applicable in case we measure exactly four pseudoranges. If we are tracking more than four satellites, the solution can be found with the Least Squares (LS) method. LS finds the optimal solution by minimizing the sum of the squares of the residuals.

2.2 Errors

The propagation time is affected by environmental factors as well as system factors. Among the system errors are the satellite errors, which include clock and orbit. Even if these errors are very small, their effect on the position computation can be substantial: recall that EM waves travel at the speed of light, therefore an error of 1 nanosecond in the satellite clock corresponds to a 30 centimeters error in the pseudorange. The same reasoning can be done for orbit drifts. For this reason, control stations continuously monitor and correct the satellites clocks and ephemerides in order to keep the informations send by satellites very accurate thanks to the fact that they have more reliable information. In fact, ground stations have more accurate clocks, and know the exact orbits of satellites.
Other errors come from the signal crossing the layers of the atmosphere (figure 2.7). The non-homogeneous nature of the atmosphere causes the carrier waves to be refracted and take more time to arrive to the receiver. The layers of the atmosphere that influence the signal propagation are the Ionosphere and the Troposphere. The Ionosphere is the furthest layer from earth. Sun ray ionize gas molecules accumulated in this layer, creating electrically charged ions and free electrons

that delay the propagation time of the EM waves. The delay depends on the carrier frequency, so L1 and L2 GNSS signal will experience different errors. In the troposphere, instead, L-band signals are equally delayed. Their propagation time depends on the refraction index, which in turn varies according to pressure and temperature.

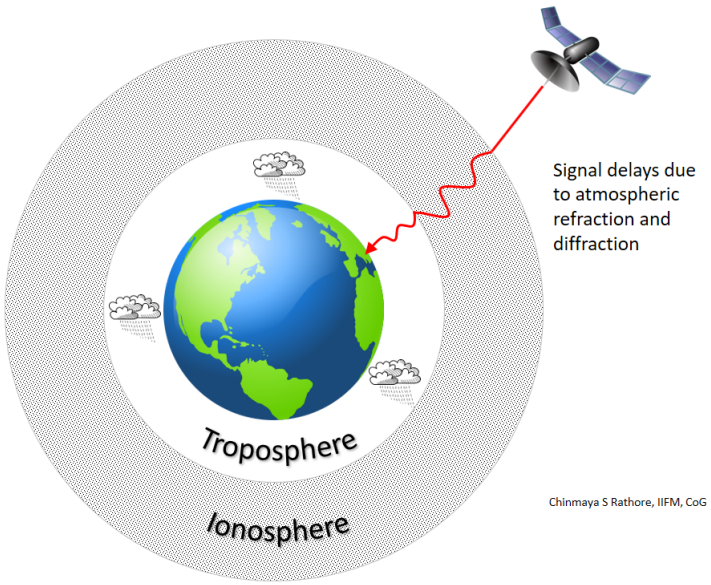


Figure 2.7: Refraction of a GNSS signal that causes a delay in propagation time

Another source of error is the multipath effect, shown in figure 2.8. This phenomenon is caused by reflection of GNSS signals off objects, such as buildings, resulting in a delayed arrival of the signal at the receiver. Multipath effects may be more or less mitigated depending on if the direct path signal is also received or not. Anyway, unless it is very large and thus is can be easily detected and neglected, the multipath delay introduces errors in the pseudorange measurements. Its effect on range errors also depends on the power of the reflected signal: the higher is its power, the larger is the error produced. Shadowing of the direct path can weaken the direct signal so severely that the multipath signal has a bigger influence on the pseudorange computation. It is important that the receiver has a clear view of the

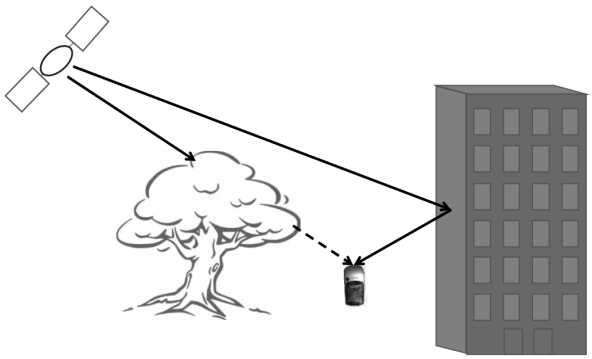


Figure 2.8: Shadowing of direct path signal and multipath occurrence

satellites and is far from possible sources of reflection. Table 2.1 shows a summary of the GNSS systems errors.

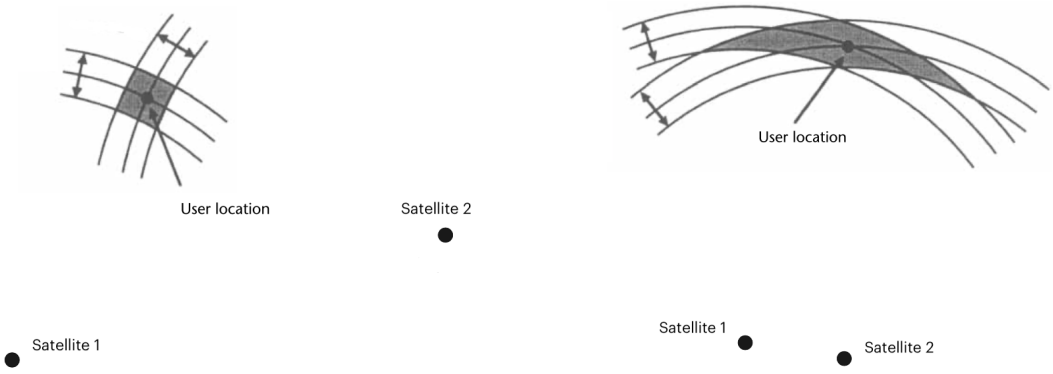
Source	**Error Range**
Satellite clocks	±2 m
Orbit errors	±2.5 m
Ionospheric delays	±5 m
Tropospheric delays	±0.5 m
Receiver noise	±0.3 m
Multipath	±1 m

Table 2.1: GNSS systems errors

2.2.1 Dilution of Precision

The satellites' geometry changes the way measurement errors propagate to the error in the position computation. This is known as the concept of *Dilution of Precision* (DOP). As we said before, the errors in the range measurements cause the pseudorange to not intersect in a point, but to determine an area of possible locations of the receiver. This area grows with the range inaccuracies, but also with the position of the tracked satellites. To visualize this, we again suppose a 2-dimensional setting. In figure 2.9a we see that two true ranges intersect in a point which is the correct location of the receiver. However, we do not actually know the

true ranges but only the pseudoranges, which are affected by an error. Supposing we can estimate an error bound, we can determine an area where the receiver might be. In figure 2.9b we can see another representation of the positioning computation where the two satellites that are being tracked have a different relative geometry with respect to the previous case. Their position in space creates a different intersection of pseudoranges, that ultimately highlights a bigger area of receiver position. Despite the fact that the pseudorange errors are the same, the two situations depict different error bounds in the GNSS solution. The factor that expresses the relationship between range errors and GNSS fix error is the DOP, and its value depends on the satellites' spacial geometry. The bigger the DOP, the higher the error in the computed position.



(a) low DOP **(b)** high DOP

Figure 2.9: Satellite geometry and dilution of precision

Estimating errors and reducing their effect on the computed position is fundamental to obtain a higher level of accuracy. One way to resolve errors is to model the phenomena that cause the error and predict a correction value. Another method is to use redundancy: take many measurements, use signals of different frequencies and exploit multiple constellations. Using multi-frequency GNSS signals

is a way for removing ionospheric errors, since they depend on the frequency of the signal. By comparing the delays of an L1 signal and an L2 signal, the receiver can mitigate the error. Multi-constellations GNSS receivers are able to reduce multipath and shadowing problems, and also obtain a better DOP. Usually this solutions result in obtaining an accuracy of few meters. Some applications, however, need higher levels of accuracy. For this purpose techniques that use augmentation are employed. Augmentation means to integrate other sources of information in the position calculation process. Until know we have discussed positioning of a *standalone* GNSS receiver, meaning that the receiver determines its position without the help of any other device. GNSS augmentation consists in using other systems that provide additional information that can be used to achieve a better position accuracy. Depending on how and what information they share with the receiver, there are different augmentation systems.

2.3 Differential GNSS

Differential GNSS (DGNSS) exploits auxiliary information to reduce the positioning errors introduced in the pseudorange computation. To do that, it uses so called *base stations*. A base station is simply a receiver whose position is fixed and known: through surveying techniques, its position can determined very accurately. As any receiver, it can calculate the pseudorange from satellites. Since its position is known, the base station can determine the exact distance from a tracking satellite. Once it computes the pseudorange, it can use it to derive the error introduced by the propagation of the signal by taking the difference with the true range. If the receiver and base station are close enough, we can consider the propagation error introduced in the pseudorange similar since satellite signals go though similar athmospheric conditions. Multipath errors and receiver noise are an exception, as they cannot be predicted to be similar. As a consequence, any receiver in

an area close eough to the base station can use that error information to adjust its computed pseudoranges. The concept of Differential GNSS is the following: base stations send pseudorange corrections for the visible satellites to the receiver requesting them (see figure 2.10). In this context, the receiver is called *rover* and the estimated errors are called *corrections*. This method allows the receiver to

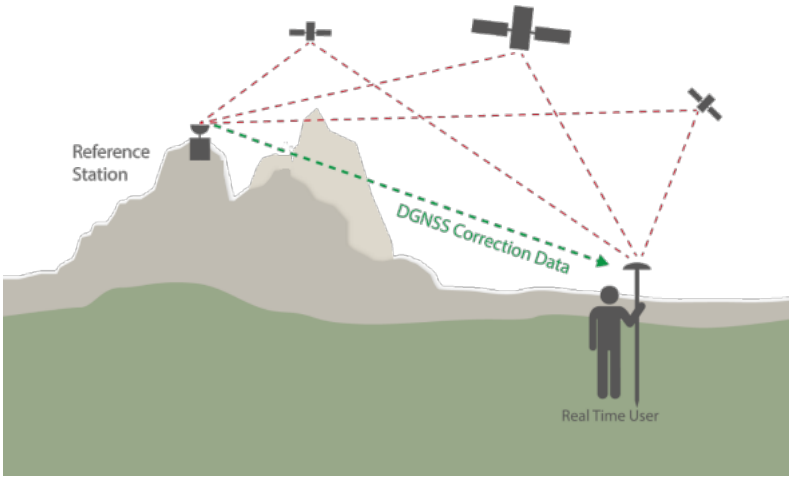


Figure 2.10: Differential GNSS correction mechanism

reach decimeter accuracy. Differencial GNSS requires base stations and rovers to communicate through a data link. The correlation between errors of two receivers is obviously higher for shorter distances: for this reason position accuracy improves with the vicinity of station and rover.

2.4 Real-time Kinematics

Another augmentation system is the *Real-time Kinematic* (RTK). The working principles of RTK are similar to Differential GNSS, as both make use of corrections sent by base stations to compute the GNSS fix. The advantage of RTK is that it uses a different technique for pseudorange computation that, in the end, provides

an accuracy of millimeter. RTK uses a technique called *carrier-phase* ranging. Until know we have talked about pseudorange calculation through correlation of PRN codes of a received GNSS signal and the receiver generated replica. This method is know as *code-phase* positioning. The difference between the two methods lies in the correlation procedure. The code-phase technique synchronizes the GNSS signal PRN code and its replica to compute the propagation time. Carrier-phase instead does correlation of carrier waves, which travel at a much faster rate, and so a good synchornization results in a very high accuracy. However, solving the carrier-phase the ambiguity problem is not trivial and requires more time.

As the name suggests, the code-phase technique evaluates the phase of the C/A code of the satellite signal by shifting the replica code in time until the two codes reach maximum correlation. The carrier-phase, instead, measures the phase of the carrier wave of the signal, which is the phase difference between the receiver-generated carrier signal and the one received from a satellite. Carrier-phase measurements are much more precise simply because the wavelength of the carrier is approximately 19 centimeters, which is much smaller than the 293 meters of pseudorandom code's wavelength (see figure 2.11). The bits of the C/A code are too wide to be perfectly synchronized, and the error we have in the phase measurement is of the meter level. Since the pulses of the carrier wave are much closer, the error we get from the synchronization of the signal and its replica is precise to the millimeter. The downside of the carrier-phase measurement method is that, unlike the code-phase, the measurement of the signal travel time is ambiguous, because the total number of cycles of the carrier between the satellite and the receiver is unknown, and resolving this ambiguity takes time.

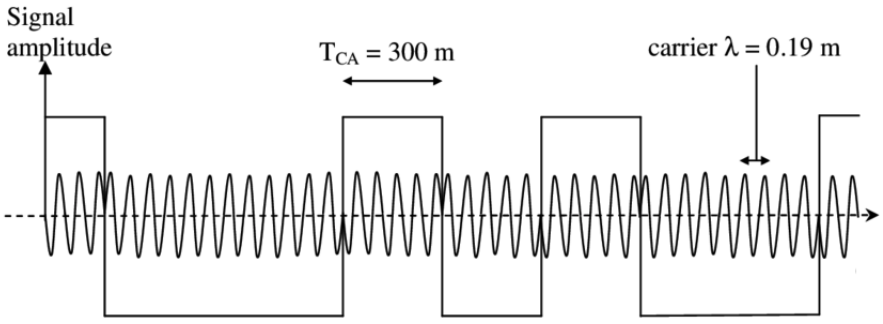


Figure 2.11: C/A code signal and carrier wave signal

2.5 Coordinate Systems

Before moving onto the next subject of this thesis, a brief introduction on coordinates systems is done to clarify how the different localization information can be merged into the same frame. In mobile robotics, the objects inside the robot environment are mapped into a two-dimensional frame denoted as *world frame*. This frame is the reference for localization data, where the map and the robot pose are expressed in. A GNSS system provides coordinates in Geodetic form, that is latitude, longitude and altitude, which is nothing else than spherical coordinates. In order to be useful for the localization process, these coordinates have to be converted into the world frame; this requires passing through different coordinate systems:

- the Geodetic coordinate system (or datum), which locates points on earth using Spherical coordinates, namely latitude, longitude and altitude.
- the ECEF (earth-centered, earth fixed) system, which has origin at the center of the earth (also named Geocentric system).
- a LTP (Local tangent plane) system, which is a family of Cartesian coordinate systems having the x-y plane tangent to the earth surface. One example is the ENU (east,north,up) frame, which has the y-axis is along the direction of a meridian and the z-axiz perperndicular to earth surface.

The world frame is an instance of an LTP system, since we can assume the robot environment to be small enough with respect to the earth surface to be considered as a flat two-dimensional tangent plane. Figure 2.12 shows the ECEF and ENU coordinate systems.

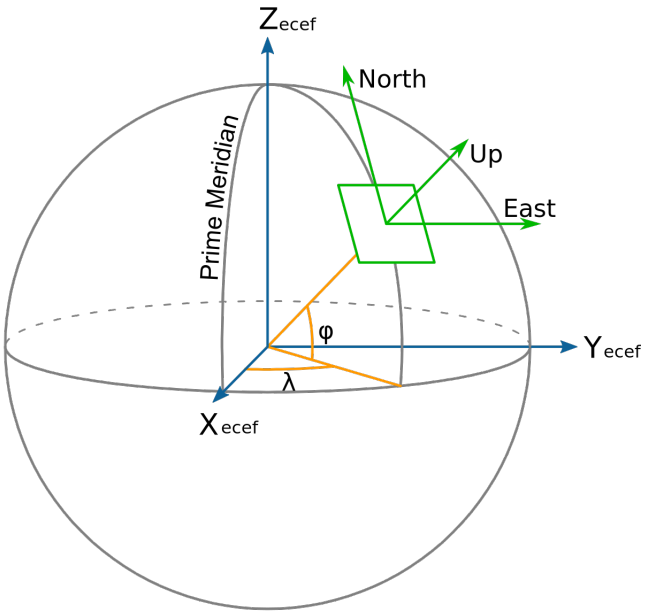


Figure 2.12: ENU and ECEF coordinate systems

2.6 Conclusions

GNSS systems can be used together with other sensors to provide robust navigation of a robot. As already discussed, GNSS performance is affected by signal obstruction, which makes the GNSS not reliable at all times for positioning. The use of other localization systems improves the localization process robustness against failures of the single positing devices, allowing a mobile robot to always have at least one functioning device. For this reason sensor fusion is applied: an accurate solution is available in all conditions by fusing the position estimates that each technology computed.

Chapter 3

Probabilistic robotics

The uncertainty present in robot perception and motion actions is the foundation of probabilistic robotics. In order to accommodate uncertainty explicitly, probabilistic algorithms represent the pose of the robot through probability distributions over the space of all possible poses[9]. Probability allows to represent the ambiguity that uncertainty introduces in the localization process, giving a sense of degree of belief about the pose of the robot. The goal of this chapter is to introduce the probabilistic framework on which localization algorithms are based. However, before delving into the details of probability, a general introduction to robot localization is done.

3.1 Robot localization

Mobile robot localization, also know as *tracking*, is the problem of determining the pose of a robot relative to a map of the environment. Since sensors for measuring the pose directly do not exist, the pose has to be recovered from sensor data. In state estimation, the non-observability of the pose is the reason why it is referred to as a hidden state. Localization algorithms make use of measurements (sometimes called *observations*) of measurable quantities to find the value of the pose that fits

them the most. Depending on the held knowledge about the robot pose, three types of localization problems are identified:

- **Position tracking**

  The robot current pose is computed based on the knowledge of its previous pose, and it is therefore assumed that the initial configuration is known. Position tracking is a local problem, meaning that the current pose of the robot will be somewhere around the previous pose, since the uncertainty is confined near the robot's true pose.

- **Global localization**

  In global localization the robot initial pose is unknown, so the robot could be anywhere in the environment.

- **Kidnapped robot problem**

  This problem is a version of the global localization where the robot is kidnapped and moved to another location. This problem is more difficult than the global localization because the robot does not know that it has been moved, and it keeps believing to have a certain pose. One might argue that robots are rarely kidnapped in practice, so solving this problem is not useful. However, the ability to solve the robot kidnapping problem is a measure of robustness of a localization algorithm.

Depending on the type of problem, different localization algorithms are applied. The basic working principles behind probabilistic localization methods are the same, and they all derive from the *Bayesian filtering* method, while they differ in the way pose distributions are modeled.

3.2 Concepts of probability

In probabilistic terms, quantities that are affected by uncertainty such as sensor readings, control actions, and robot and environment states can be modeled through random variables. A random variable is a variable that can assume multiple values as a result of an experiment (e.g. a measurement) according to a certain probability distribution. In this context, the robot pose can be defined as a random variable whose domain is the space of all possible poses. Since this space is continuous, the state (pose) is characterized by a probability density function (PDF) that defines with what likelihood the pose assumes any of the values in the space. In the context of mobile robotics, the robot control system will have to compute the estimate of the position in real time. This means that algorithms have to converge to a solution in a limited time frame. For this reason, assumptions and simplifications are often made to improve the speed of convergence at the cost of an approximate solution. Moreover, since computational systems are digital, they cannot take into account an infinite range of values, whatever is the context. These practical limits force the localization process to consider the robot state space as discrete and not continuous, so it is supposed that only a finite number of possible poses in the map can be assumed by the robot. The pose is thereby described by a discrete random variable.

The reason why the mobile robot localization problem is also known as the tracking problem is that the robot moves while performing its tasks, therefore the aim of the estimation process it to track the robot configuration over time. Given the stochastic nature of the robot pose at any moment in time, the evolution of the state is best described in probability by a stochastic process. Even though in reality the pose changes continuously, localization algorithms are only able to give estimates at specific time instants. As in any computational system, time is discretized and we indicate a generic time step as t. The goal of a localization

algorithm is to compute the pose at any discrete time instant t from the sensor measurements. As already discussed, sensors provide two types of information: proprioceptive, that regards the internal state of the robot, and exteroceptive, that regards the surroundings of the robot. The robot pose can be estimated from either of the two types of information, and probabilistic algorithm actually consider both.

In the following discussion we will denote with x_t the pose of the robot at a generic time instant t. Since it has some uncertainty, x_t is described by a probability distribution that in robotics is referred to as *belief*. To estimate the belief we are going to use the concept of *conditional probability*, which defines the probability of an event to happen conditioned on the fact that the outcome of another event is already known. We could say that the future robot pose x_t is conditioned on the values of all the previous poses $x_{0\to t-1}$, inputs $u_{0\to t-1}$ and measurements $z_{0\to t-1}$. In other words, the history of the robot may influence the stochastic evolution of its pose. For simplicity reasons, we assume that the pose at the next time step x_t depends only on the current pose x_{t-1} and, of course, on the future input u_t and measurement z_t. This assumption is known as the *Markovian assumption*, and a random process that satisfies this condition is called a *Markov chain*.

3.2.1 Motion model

The goal of localization is to estimate the robot pose x_t at a generic time step t. Between two consecutive time steps the robot performs the motion actuated by the input command. By knowing the input command u_t, the pose x_t can be found from the previous pose x_{t-1} through the kinematic model of the robot, which can be seen as a function $f(x_{t-1}, u_t)$. Ideally, if we knew the exact pose at time $t-1$, then we would find the exact pose at time t. However, the robot motion is not deterministic as it is affected by uncertainty (see figure 3.1), so the kinematic model can only give an estimate of the most likely future pose x_t. This uncertainty

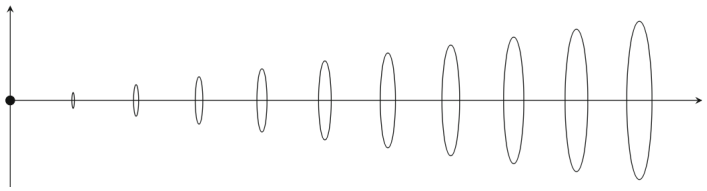


Figure 3.1: Growth of the pose uncertainty over time for a straight movement.

is caused by different factors, which may include environmental reasons, model approximation and motion actions. One could argue that the input itself is not carried out exactly due to the non-ideality of the actuators, and therefore it would be better to measure the motion of the robot using proprioceptive sensors instead of deriving it from the input. However, also measurements are affected by noise and are not exact, therefore they may not give reliable information about the motion. Thus, we will consider a generic input u_t without making any distinction between the source of this value. In practice u_t could be a sensor measurement like the reading of the robot's wheel encoders, or a control input like the one given to the motors (e.g. velocity). Since the motion is uncertain, the evolution of the pose needs to be modelled through a probability distribution which defines the *motion model*. This distribution gives the probability that, given a state x_{t-1} and an input u_t, the robot pose assumes at the next time step a certain pose x_t (see figure 3.2). The probability just described is known as *state transition probability* and, since it depends on the present input and the previous state, it is a conditional probability and is therefore indicated as $p(x_t|x_{t-1}, u_t)$.

3.2.2 Measurement model

Proprioceptive sensors measurements are used in the motion model to predict how the pose x_{t-1} evolves into x_t as function of the control input. While motion data gives information about the degree of change of the pose between consecutive time steps, measurements from exteroceptive sensors provide information about the

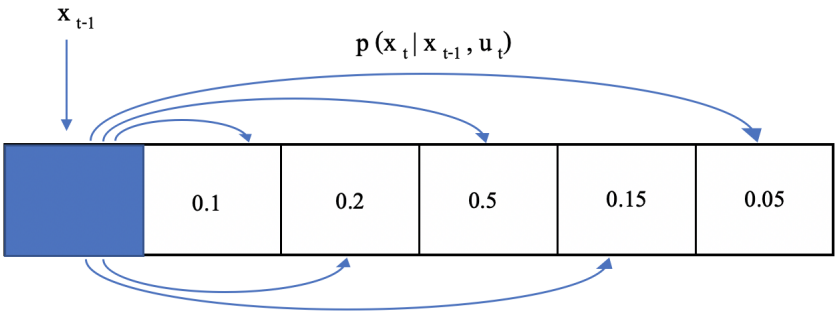


Figure 3.2: Example of transition probability of moving from a state x_{t-1} to x_t in one dimension

pose at given time instants. Given a specific state x_t, we would expect a certain measurements z_t coming from the exteroceptive sensors according to a function $g(x_t)$. However, because measurements are noisy, a sensor may provide a variety of different values which are not the one predicted. To take into account this uncertainty we need to build a *measurement* (or *observation*) *model* that defines the probability distribution of having a sensor reading z_t from a pose x_t. This model is derived directly from the error model of the sensor, which gives a description of the reliability of a measurement. Once a measurement is taken, we are able to select the corresponding *measurement probability* $p(z_t|x_t)$ and use it to update the probability of the estimate x_t, as we will see later.

3.2.3 Belief distribution

The belief of a robot is represented by a conditional probability distribution conditioned on the available information. Supposing to be at time $t-1$, the probability to be in a state x_{t-1} is given by $p(x_{t-1}|u_{t-1}, z_{t-1})$, which is conditioned on the data u_{t-1} and z_{t-1}. To simplify the notation, we will simply denote this probability as $p(x_{t-1})$. The goal is to estimate the pose belief at time instant t using the information given by the control input u_t and the measurement z_t; in

other words, we want to find the probability $p(x_t|u_t, z_t)$. To distinguish between the beliefs before and after taking into account the collected data, we denote the belief of $p(x_{t-1})$ as *prior* and the belief of $p(x_t|u_t, z_t)$ as *posterior*. A common notation used in robotics to denote the posterior belief probability is $bel(x_t)$, and correspondingly for the prior is $bel(x_{t-1})$. We also make a further distinction by denoting the probability $p(x_t|u_t)$ to indicate a posterior after applying the motion input u_t but before including the measurement z_t. This belief probability is also denoted as $\overline{bel}(x_t)$, and for reasons that will be explained later it is often referred to as *prediction*.

3.3 Bayes filter

The Bayes filter is the most general probabilistic algorithm for the computation of the probability density function of a system state. In the context of robotics it can be used to estimate the belief in the presence of system and measurement uncertainties. The Bayes filter is an iterative algorithm which computes the belief $bel(x_t)$ at time t from the belief $bel(x_{t-1})$ at the previous instant $t-1$ using the most recent control action u_t and the most recent measurement z_t. The information needed at each new iteration includes:

- prior probability $p(x_{t-1})$ (or $bel(x_{t-1})$)
- motion model probability $p(x_t|x_{t-1}, u_t)$
- observation model probability $p(z_t|x_t)$
- the observation z_t and control input u_t
- an initial belief $bel(x0)$

The idea of the filter is first to predict the future pose x_t based on the knowledge of the previous pose x_{t-1} and the control input, and then to adjust the obtained pose by combining it with the sensor measurement. The two steps just described

make up one cycle of the Bayes filter (figure 3.3), and are denoted respectively as *prediction* step and *correction* step.

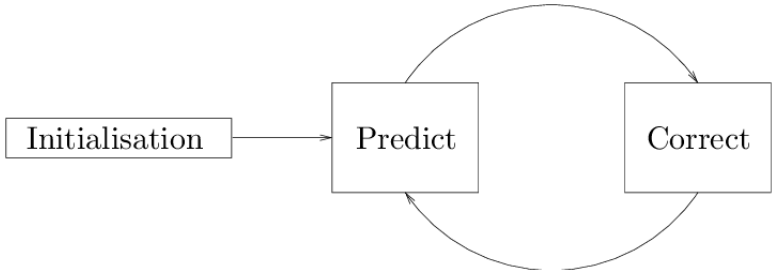


Figure 3.3: Estimation cycle of the Bayes filter

Prediction

In the prediction step, the robot calculates the probability $p(x_t|u_t)$ to end up in a state x_t after a motion command has been issued. To do that it exploits the motion model, which gives the probability $p(x_t|x_{t-1}, u_t)$ that the control action u_t generates a transition from pose x_{t-1} to x_t. The belief $p(x_t|u_t)$ is calculated using the *law of total probability*:

$$p(x_t|u_t) = \int p(x_t|x_{t-1}, u_t)p(x_{t-1})\, dx_{t-1} \tag{3.1}$$

This equation has an intuitive explanation: to compute the probability that the pose at time t assumes a value x_t we have to consider all the possible ways in which the robot can reach that pose, that is, we have to sum the probabilities to reach x_t from any possible previous pose x_{t-1}. In mathematical terms, we have to compute the integral (or sum in case of discrete state) over every possible prior pose of the product between the probability $p(x_{t-1})$ of being in a certain previous pose and the probability $p(x_t|x_{t-1}, u_t)$ of transitioning from that pose to x_t.

Correction

The correction step starts from the knowledge of the posterior belief given by the prediction step. At time t a new measurement z_t is taken by the robot sensors and it has to be incorporated in the belief estimation to update the predicted belief $p(x_t|u_t)$. The updated probability is computed according to the *Bayes rule*:

$$p(x_t|z_t, u_t) = \frac{p(z_t|x_t, u_t)p(x_t|u_t)}{p(z_t|u_t)} \tag{3.2}$$

Actually, z_t does not depend on u_t, therefore the formula could be re-written as:

$$p(x_t|z_t, u_t) = \frac{p(z_t|x_t)p(x_t|u_t)}{p(z_t)} \tag{3.3}$$

The idea of the correction step is to update the probability to be in a certain state x_t predicted by the motion model using the sensor measurement and the probability of observing that measurement. The Bayes rule is a convenient tool to compute the corrected posterior probability from the inverse probability $p(z_t|x_t)$, which is given by the measurement model. The measurement model defines the likelihood of observing any measurement z_t from a pose x_t. When the actual measurement is observed, the corresponding probability $p(z_t|x_t)$ is selected and it is applied for the correction. If the sensor measurement value is close to the one expected by the measurement model, then the probability of being in the pose x_t increases since the validity of that estimate is somehow confirmed by the sensor reading. On the other hand, if the measurement is far from the expected value, the probability of being in state x_t given by the prediction step decreases. The denominator $p(z_t)$ does not depend on the state x_t, therefore it is usually assumed as a normalization factor η determined knowing that the integral of the probability distribution of x_t given the measurement z_t has to sum up to one. The Bayes filter in this case can

be re-written as:

$$p(x_t|z_t, u_t) = \eta \cdot p(z_t|x_t)p(x_t|u_t) \tag{3.4}$$

Recalling that the prediction probability $p(x_t|u_t)$ is denoted as $\overline{bel}(x_t)$, and that the posterior probability $p(x_t|z_t, u_t)$ is denoted as $bel(x_t)$, and also putting together the prediction step and the correction step, we obtain a final general formula of the Bayes filter:

$$\begin{aligned} bel(x_t) &= \eta p(z_t|x_t)\overline{bel}(x_t) \\ &= \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})\, dx_{t-1} \end{aligned} \tag{3.5}$$

The pseudocode for the Bayes filter is shown in table 3.1.

1: **Bayes Filter** ($bel(x_{t-1}), z_t, u_t$)**:**
2: **for** all x_t **do**
3: $\quad \overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})\, dx_{t-1}$
4: $\quad bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$

Table 3.1: Bayes filter algorithm pseudocode

3.3.1 Bayes filter for localization

In the context of mobile robot localization, the implementation of the Bayes filter for pose estimation is called *Markov localization*. Markov localization addresses the problems of global localization, position tracking and the kidnapped robot problem in static environments. Recall that, since robots require a map to navigate, the Markov localization algorithm uses also the knowledge of a map to estimate the posterior. The pseudocode of Markov localization (see table 3.2) slightly differs from

the Bayes code as the observation model takes into account the features of a map m to compute the measurement probability, which thereby becomes $p(z_t|x_t, m)$. A

1: **Markov localization** ($bel(x_{t-1}), z_t, u_t$)**:**
2: **for** all x_t **do**
3: $\quad \overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t) bel(x_{t-1})\, dx_{t-1}$
4: $\quad bel(x_t) = \eta p(z_t|x_t, m)\overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$

Table 3.2: Markov localization algorithm pseudocode

simple one-dimensional example of the Markov localization algorithm is depicted in figure 3.4. The robot starts from an unknown pose, thus it is reasonable to assume that the initial belief $bel(x_0)$ is described by a uniform distribution over all possible poses in the map. Supposing that the sensors detect a door in front of the robot (figure 3.4b), the belief can be updated according to the Bayes rule multiplying $bel(x_0)$ by the measurement probability $p(z_t|x_t, m)$. Since a door has been observed, the belief distribution grows in density around the position of the three present doors: this comes directly from the fact that the probability $p(z_t|x_t, m)$ is higher when x_t is close to a door. Then at the next time step (figure 3.4c) the robot moves to the right. The new belief is obtained as result of the convolution (as in the law of total probability) of the robot's previous belief with the motion model probability $p(x_t|x_{t-1}, u_t)$, which, due to the uncertainty that introduces, causes the belief distribution to flatten out. At this point the distribution function has gained uncertainty from the previous belief. After observing the next measurement, the correction step is executed again. Finally, as another door is detected, the product of the prediction with the perceptual probability $p(z_t|x_t, m)$ determines a new belief (illustrated in figure 3.4d) that has a distinguishable peak near the correct pose of the robot. As the robot keeps moving and observing the environment, its belief should improve giving a quite confident estimate of its pose.

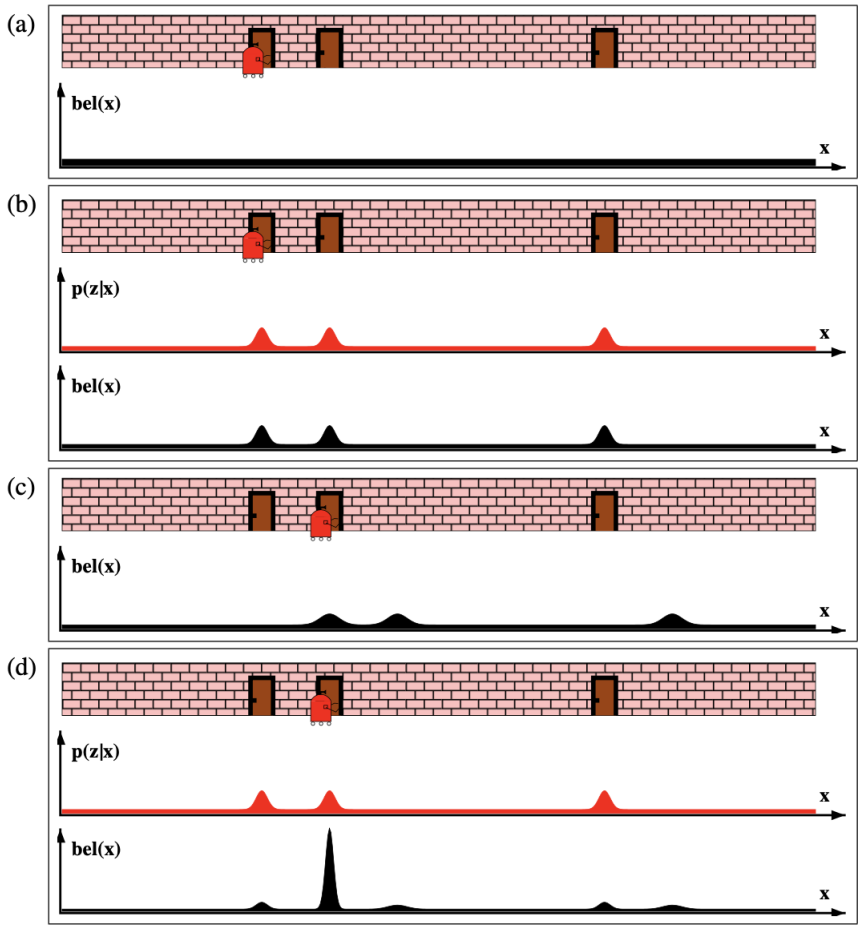


Figure 3.4: Illustration of the belief computation in Markov localization algorithm

Markov localization, and implicitly also the Bayes filter, represents the robot's belief through an arbitrary probability density function, thus it constitutes the most generic form of probabilistic localization algorithm. In practice, however, all Markov localization derived systems use simplified belief representations in order to ease the computational load. This comes at the cost of an approximate solution, given the fact that the assumed distribution may not represent accurately the pose. One of the most efficient and used algorithms for estimation is the *Kalman filter*, which represents the belief through a Gaussian function. Another approach to

simplify the belief representation is to discretize or decompose the state space and approximate the pose distribution using a discrete number of values. This approach is adopted by a family of Bayes-derived methods called *non-parametric filters*. Some of them decompose the state space into regions and assign to each region a single probability value that represents the cumulative probability to assume any of the poses of that region; no information on the probability distribution within a region is kept. An example is the *Histogram filter*, which represents the belief with a piecewise constant PDF where a uniform probability is assigned to each pose within a region. Others instead approximate the pose space through a finite number of samples, therefore considering only a discrete number of poses in the map; this is the case of the *Particle filter*. The quality of these non-parametric techniques depends on the number of parameters used to approximate the belief distribution: the higher this number, the more reliable the posterior approximation. *Grid localization* is an alternative to Markov localization that uses a histogram filter (figure 3.5) to represent the belief over a grid decomposition of the space, shown in figure 3.6. Another popular localization algorithm in robotics is the

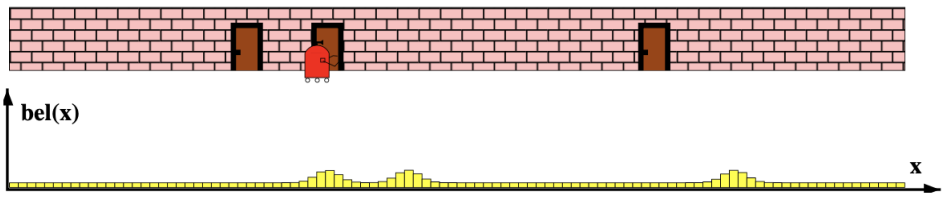


Figure 3.5: One-dimensional grid-based localization: the belief is represented by a histogram over a grid.

Monte Carlo localization, that uses a particle filter to estimate the belief over the robot possible poses (figure 3.7). In the following two sections, the Kalman filter and the particle filter are discussed, since they are an important part of this work.

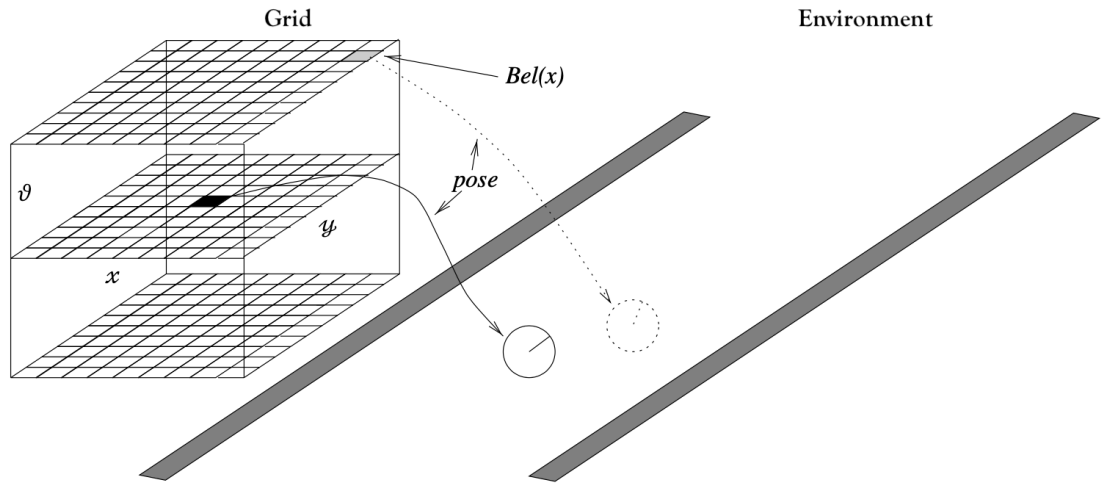


Figure 3.6: Fixed grid representation of the 2-dimensional pose (x, y, θ). Each grid represents a robot pose in the environment. The orientation is represented on the z-axis, and each layer represents all the possible poses with the same orientation.

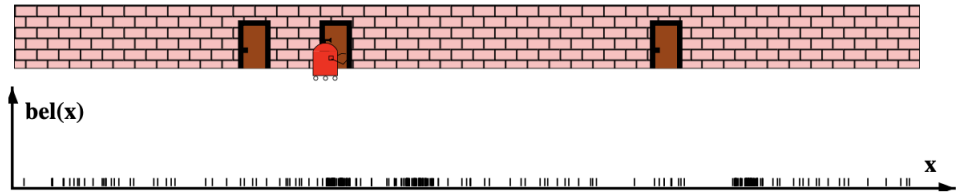


Figure 3.7: Monte Carlo localization applied to the one-dimensional case. The belief is represented by a finite number of possible positions.

3.4 Kalman filter

The Kalman filter is an implementation of the Bayes filter for continuous state spaces where the robot's belief is represented by a single Gaussian probability density function. The advantage of using this representation is that it simplifies significantly the computations, resulting in a very efficient algorithm in comparison to Markov localization. This simplification comes from the fact that a multivariate normal distribution is uniquely determined by its mean vector μ and its covariance matrix Σ, therefore at runtime the algorithm needs to update only these two parameters.

Gaussian functions are unimodal, that is they have a single maximum. The advantage of having a single-hypothesis belief is that there is no pose ambiguity: the most probable pose corresponds to the mean of the distribution. Such distributions depict the scenario of many localization problems, where we approximately know where the true pose is but with some uncertainty. However, the assumption of a single hypothesis belief, such as the one made by using a normal distribution, implies that the initial belief $bel(x_0)$ must be a Gaussian, thus the robot initial pose must be known with a certain degree of uncertainty. This makes the Kalman filter not suitable for localization problems in which multiple pose hypotheses exist, as the global localization problem and the kidnapped robot problem, and useful only for tracking problems.

The Kalman filter starts from the assumption that the robot initial belief $bel(x_0)$, the motion model and the measurement model can be represented by normal distributions: this guarantees that the posterior belief $bel(x_t)$ computed by the algorithm remains always a Gaussian function. Moreover, it assumes that the overall system is linear, thus both state transitions and measurements are linear functions of their variables. In particular:

- the motion model $f(x_{t-1}, u_t)$ is linear, so the prediction of x_t is based on a linear function of the previous belief x_{t-1} and of the input u_t, and it is affected by white Gaussian noise (known as *process noise*), which introduces uncertainty in the motion, represented by a random variable ϵ_t:

$$x_t = A_t \; x_{t-1} + B_t \; u_t + \epsilon_t \tag{3.6}$$

  This equation defines the transition probability $p(x_t|x_{t-1}, u_t)$, which has a Gaussian PDF $\mathcal{N}(A_t x_{t-1} + B_t u_t, R_t)$ with mean $A_t x_{t-1} + B_t u_t$ and covariance R_t (that is exactly the covariance of the distribution of ϵ_t).

- the measurement model $g(x_t)$ is also a linear function affected by a Gaussian measurement noise δ_t, which comes from the sensor error model:

$$z_t = C_t \; x_t + \delta_t \tag{3.7}$$

The measurement probability $p(z_t|x_t)$ is determined by a normal distribution $\mathcal{N}(C_t x_t, Q_t)$ with mean $C_t x_t$ and covariance Q_t, where Q_t is the covariance matrix of δ_t's probability distribution.

1: **Kalman Filter** $(\mu_{t-1}, \Sigma_{t-1}, z_t, u_t)$**:**
2: $\overline{\mu}_t = A_t \; \mu_{t-1} + B_t \; u_t$
3: $\overline{\Sigma}_t = A_t \; \Sigma_{t-1} \; A_t^T + R_t$
4: $K = \overline{\Sigma}_t \; C_t^T \; (C_t \; \overline{\Sigma}_t \; C_t^T + Q_t)^{-1}$
5: $\mu_t = \overline{\mu}_t + K_t \; (z_t - C_t \; \overline{\mu}_t)$
6: $\Sigma_t = (I - K_t \; C_t) \; \overline{\Sigma}_t$
7: **return** μ_t, Σ_t

Table 3.3: Kalman filter algorithm pseudocode

Table 3.3 shows the pseudocode of the Kalman filter. It is an iterative algorithm that computes at each time step the mean μ_t and its covariance Σ_t of the pose belief. As any Bayes-like filter, the Kalman algorithm is a two-step process that comprises a prediction and a correction, as shown in figure 3.8. The prediction step computes the belief $\overline{bel}_t$ before incorporating the measurement z_t. From equation 3.6 we can compute this belief by making the following reasoning: the value of x_{t-1} is represented, according to the Kalman filter assumptions, by a Gaussian function $\mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$; if we consider ϵ_t to be the noise of the input product $B_t u_t$, we can represent also the input as a normal distribution $\mathcal{N}(B_t u_t, R_t)$. Then, according to the motion equation, the distribution $\overline{bel}(x_t)$ is obtained by a linear combination of the two Gaussian distributions of $bel(x_{t-1})$ and u_t. Considering the input and

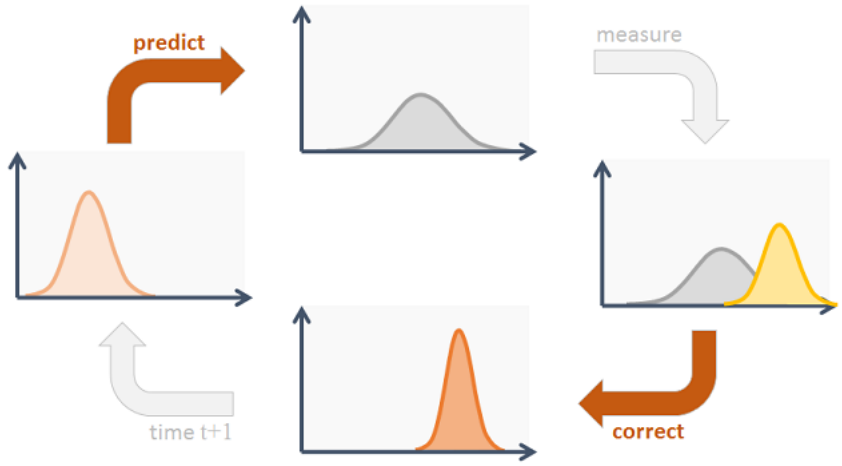


Figure 3.8: Kalman filter cycle

the belief of x_{t-1} independent, the result is still a Gaussian distribution with mean and covariance shown in lines 2-3 of the algorithm. As depicted in figure 3.9, the uncertainty on the robot pose after the motion is larger, and this is a direct consequence of the stochasticity of the pose transition.

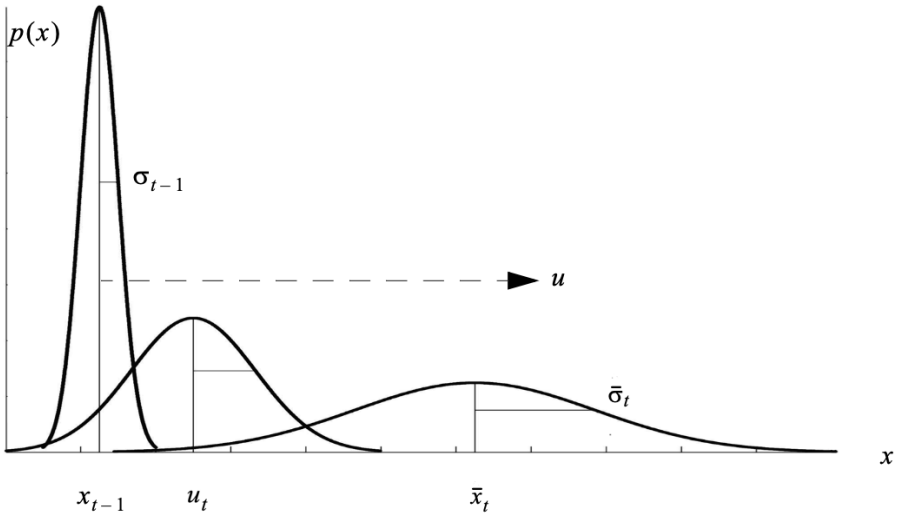


Figure 3.9: Propagation of the robot's belief in one dimension

The correction step is done applying the Bayes rule. Once the measurement z_t is returned by the sensors, the distribution of the probability $p(z_t|x_t)$ is known. As stated by Bayes rule, $bel(x_t)$ is given by the product of the probabilities $\overline{bel}(x_t)$ and $p(z_t|x_t)$, which are both represented by normal distributions. The resulting probability density function of $p(x_t|z_t, u_t)$ is still a Gaussian with the mean and covariance shown is lines 5-6 of the Kalman algorithm. The updated mean pose is obtained by adding to the predicted mean $\overline{\mu}_t$ a quantity proportional to the error between the expected measurement and the actual one. This trade-off is given by the matrix K_t, which is known as the *Kalman gain*. An intuitive interpretation of the correction update can be easily observed from the application of the Kalman filter in one-dimension, where the covariance matrix is substituted with a variance and the multiplication matrices are substituted with scalars. By unfolding the equation in line 5 of the algorithm, we can re-write it as:

$$\mu_t = \frac{\overline{\mu}_t \frac{\sigma_c^2}{c^2} + \frac{z_t}{c}(\overline{\sigma}_t)^2}{\frac{\sigma_c^2}{c^2} + (\overline{\sigma}_t)^2} \tag{3.8}$$

From this form of the equation it is clear that the mean pose of the posterior belief is a weighted average of two quantities: $\frac{z_t}{c}$ and $\overline{\mu}_t$. The latter is the mean of the predicted distribution, while the former is the mean of the pose guess retrieved from the sensor measurement. Moreover, the weights are represented by the variances of the two quantities just described. This cannot be seen directly from line 5 of the algorithm because the weights are hidden inside the kalman filter. Thus, the kalman filter performs a trade-off of importance to give between the prediction and the pose guessed from the measurement, based on the respective variances. In particular, the lower is the uncertainty on one of the two quantities, the more it contributes to the posterior mean. As a result of the correction step, the posterior mean lies in between the predicted mean and the mean of pose guessed from

the measurement, and the overall uncertainty is smaller than the one of the two contributing Gaussians (see figure 3.10). This is a consequence of the mathematics

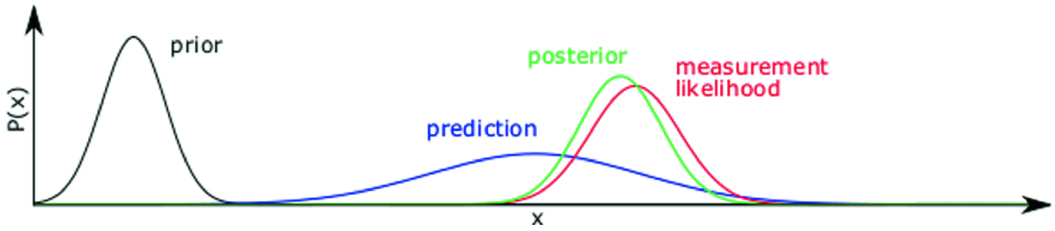


Figure 3.10: Pose distribution curves in a one-dimensional Kalman filter cycle

describing operations between Gaussians.

Figure 3.11 shows an example of position tracking with a Kalman filter in a two-dimensional scenario. The figure shows one iteration of the algorithm, starting with the robot having a belief (represented by a circle) about its position. After the robot moves, the prediction step makes the position estimate mean shift in space and the uncertainty grow (the circle is wider). Afterwards, a measurement is taken which suggests that the position has an offset from the mean of the prediction and has a lower covariance. Performing the correction step, the new posterior belief has a mean which is somewhere between the mean of the prediction and the mean of the measured position, and has a covariance which is significantly reduced.

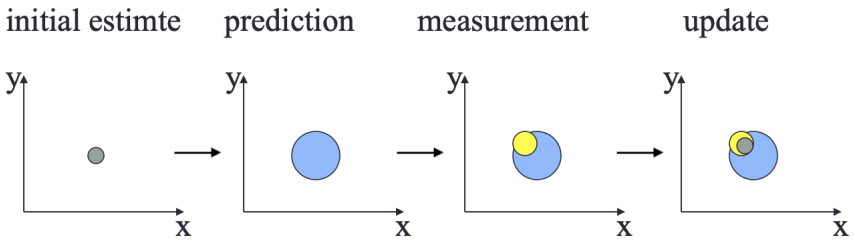


Figure 3.11: Evolution of position belief computed in one Kalman filter cycle in a two-dimensional scenario

The limit of the Kalman filter is that it can be applied only when the assumptions

on linear state transitions, linear measurements and Gaussian representation hold. This is rarely the case in practice, indeed the robot is likely to move on trajectories that are non-linear, for example on circular paths. A solution is offered by the *Extended Kalman Filter* (EKF), which is a version of the standard Kalman filter for the non-linear case. Moreover, the unimodal belief representation only works when the robot initial pose is approximately known. In all other cases, other algorithms like the particle filter are preferred.

3.5 Particle filter

Particle filters represent the posterior belief using a finite set of pose samples called *particles* instead of a continuous distribution, as depicted in figure 3.12. This is

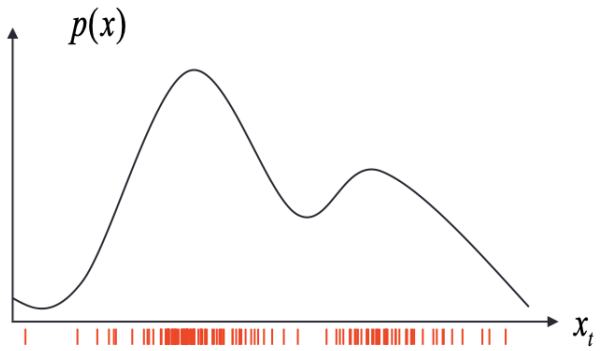


Figure 3.12: Distribution of particles in particle filter belief representation

very powerful because it does not limit the use of this estimation technique to particular analytic distributions, like Kalman filters do. The probability density of the belief is represented by the particles distribution, thus the probability that the pose assumes some value x_t is given by the probability to draw out the particle closest to that value. In other words, the more are particles that populate a region of the space, the more likely it is that the true pose belongs to that region.

The particle filter, like the Bayes filter, is an iterative algorithm that builds the belief $bel(x_t)$ from the previous belief $bel(x_{t-1})$. However, since it describes distributions using particles, what it really does is to estimate a posterior particle set $\mathcal{X}_t$ from the previous set $\mathcal{X}_{t-1}$. As shown by the pseudocode in table 3.4, in each cycle the algorithm starts from a set $\mathcal{X}_{t-1}$ of N particles which all have the same weight. Even though a prediction and a correction step can be recognized, the algorithm is better visualized through three steps:

1. For each particle $x_{t-1}^{[i]}$ in the set $\mathcal{X}_{t-1}$, a prediction about its pose at the next time instant is made based on the control input u_t. Since the motion model is stochastic, from a single particle we obtain a distribution of possible poses x_t. Thus, the transition probability distribution $p(x_t|x_{t-1}, u_t)$ has to be sampled in order to obtain only one predicted particle.
2. Once the predicted set of particles $\bar{\mathcal{X}}_t$ has been computed, the particles are weighted using the measurement likelihood model and the measurement z_t. Weighting allows to update the probability of a particle to represent the real pose according to the sensor readings values.
3. Since the goal is to obtain a set of unweighted particles $\mathcal{X}_t$ from which to start another cycle of the particle filter, a *re-sampling step* is performed: particles are drawn with replacement from the weighted set $\bar{\mathcal{X}}_t$ with a probability proportional to their weight, and then they are placed without weight in the posterior set $\mathcal{X}_t$.

Step 1 and step 2 are performed in lines 4-6, while step three in lines 9-10. With this method the posterior accumulates duplicates of the same particles, since the re-sampling process is done with replacement. The presence of duplicates indicates that the probability density is higher around those particles. An alternative is to consider only the first two steps of the algorithm, which implement the prediction

1: **Particle Filter** $(\mathcal{X}_{t-1}, z_t, u_t)$**:**
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = 0$
3: **for** $i = 1$ *to* N **do**
4: sample $x_t^{[i]} \sim p(x_t | x_{t-1}^{[i]}, u_t)$
5: $\omega_t^{[i]} = p(z_t | x_t^{[i]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \left\langle x_t^{[i]}, \omega_t^{[i]} \right\rangle$
7: **end for**
8: **for** $i = 1$ *to* N **do**
9: draw $x_t^{[i]}$ from $\bar{\mathcal{X}}_t$ with probability $\propto \omega_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: **end for**
12: **return** $\mathcal{X}_t$

Table 3.4: Particle filter algorithm pseudocode

and the correction update. Let alone they would be enough, since we could choose to represent the probability of a particle through its weight instead of through its multiplicity. However, the power of re-sampling is to discard low-probability poses and maintain high-probability ones. If the algorithm would not re-sample, the particle set would contain poses with very low probability which are not useful to the pose estimation process. Instead, it would be better if those less likely particles were discarded (not drawn), so that the N present particles always represent the most probable locations. With re-sampling, the less realistic particles will eventually go away because they are inconsistent with the measurements, therefore they are weighted less and during the re-sampling step they have less chance to be drawn. Only the samples that are consistent with the measurements survive. After a while that the algorithm is running, the remaining particles may be grouped in clusters distant from each other: this is the case when there is an ambiguity in the position and the robot cannot find features of the environment that uniquely determine its location. This, though, is another advantage of the particles filter which makes it suitable for global localization and kidnapped robot problems.

A graphical illustration of the particle filter is shown in figure 3.13. The

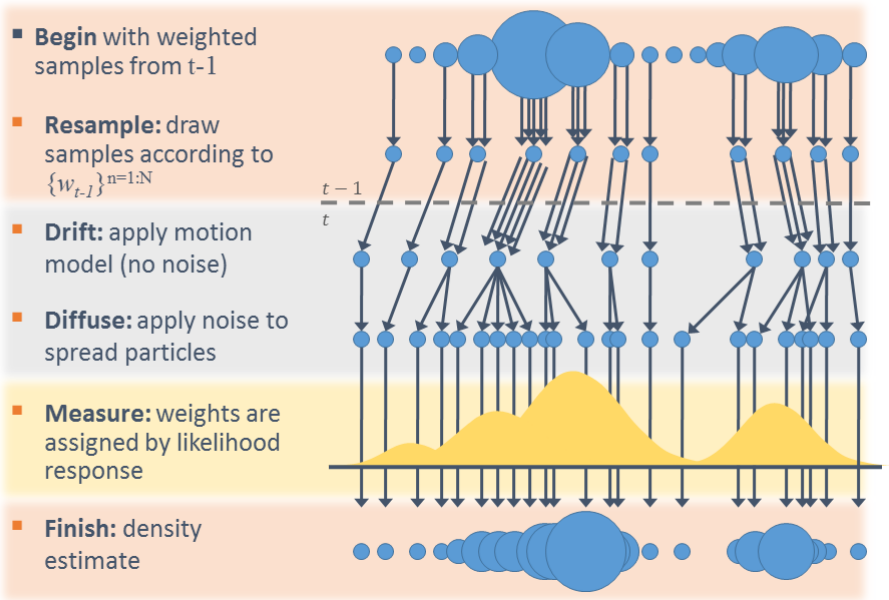
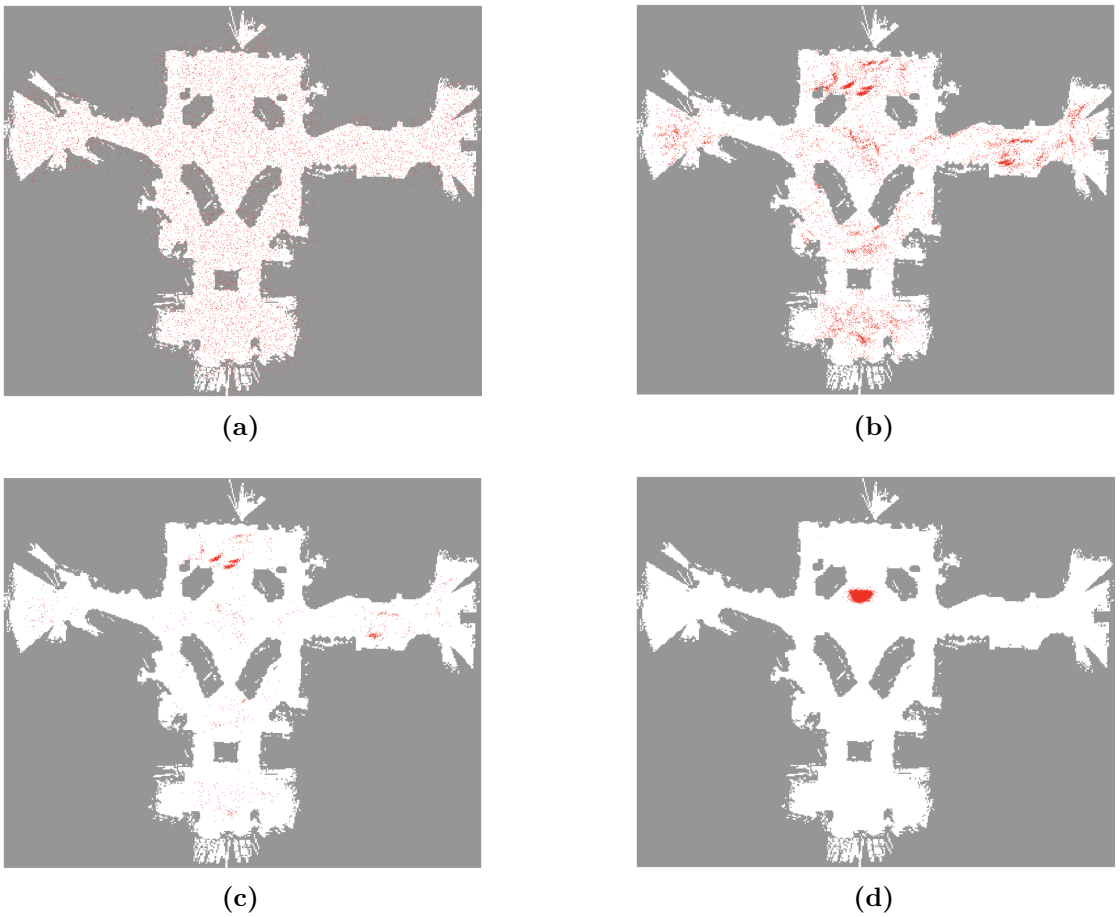


Figure 3.13: Particle filter cycle

example starts from step 3 of the algorithm, where re-sampling occurs and particles with higher weights are likely to be drawn more than once, while the ones with lower weights may not be drawn at all. Then, drift and diffusion are applied in accordance to the motion model and a new set of particles $\bar{\mathcal{X}}_t$ is defined. Finally the measurement z_t is taken and the particles are weighted accordingly, so to give a better chance to be re-sampled to the particles that reflect the most the measured value.

As already discussed, the particle filter is widely used in map-based robot localization problems, where it is known as Monte Carlo Localization[10] (MCL), as it is simple to implement and it works across a broad range of localization problems. In figure 3.14 we can see how in MCL localization the particles distribution evolves over time from subfigure 3.14a, where they are uniformly distributed over the entire map, to subfigure 3.14d, where they are clustered in a small area that clearly

identifies the robot position.



(a) **(b)** **(c)** **(d)**

Figure 3.14: Monte Carlo Localization

Chapter 4

Project development

4.1 Project overview

This project has been developed under the supervision of the InnoTech Systems company in collaboration with the California State University of Los Angeles. InnoTech Systems was founded in 2018 with the purpose of providing new solutions in the field of Service Robotics. In 2019 the company started a new project involving the development of an autonomous mobile robot that provides assistance and services in various ways to airports and other transportation centers. The use of robotic platforms allows to significantly reduce operational costs and increase the efficiency of the workplace. The work carried out in this thesis aims at enriching the robot's capabilities to localize inside an environment. The robot is equipped with Simultaneous Localization and Mapping (SLAM) technology that allows for autonomous navigation in either known or unknown environments. As discussed in the next subsection, SLAM has some limitations that hinder the efficiency of robot navigation. The focus is on the topic of robot localization, that is an essential step for path planning and monitoring of the trajectory. To improve the accuracy and robustness of localization, a GNSS system can be added to support the SLAM

algorithm in those situations in which it is the most vulnerable. Most of the focus in the development of this work has been dedicated to searching for and understanding the working principles of localization methods and of data fusion techniques that would provide a more robust localization outcome.

4.1.1 The problem of SLAM

The goal of autonomous navigation is to move from a starting point to a target point inside an environment. In order to generate a motion path, the robot needs an environment map and at the same time its current location. When the map is not known a priori, the robot needs to build it by itself. The autonomous creation of a map involves the use of on-board sensors to perceive the environment and of techniques for place recognition and data association[11]. Of course sensors have a limited range and resolution, so the robot must explore the environment to build a map. This problem is known as Simultaneous Localization and Mapping, and involves the construction of a map of the environment while trying to localize inside it. Localization, as previously discussed, is the problem of estimating the robot pose (and implicitly its path) inside a map. On the other hand, Mapping aims at building a map of the environment knowing the position of the robot in the map. Thus, it is clear that SLAM is a chicken-and-egg problem, since to perform mapping the robot requires localization and vice versa. One of the approaches to solve this problem is represented by probabilistic algorithms based on Bayesian filtering theory, like the Kalman filter or the Particle filter, which are the same methods used for the problem of localization. These techniques estimate both the map and the position of the robot exploiting two main sensor technologies: vision sensors (cameras) and laser range sensors (LiDAR)[12]. The power of SLAM algorithms is to allow the robot to move autonomously in unknown and unstructured environments, therefore making a robot truly autonomous; it provides an alternative to user-built maps, showing

that robot operation is possible in the absence of any type of previously built localization infrastructure[13]. However, few downsides limit the application and efficiency of SLAM-based navigation. First of all, the technologies used have their limitations, for example a 2D-LiDAR that uses laser pulses to detect obstacles may encounter objects that do not reflect laser scans very well. Laser scanners are still widely employed for SLAM, however cameras are becoming more appealing because they are cheaper and provide much richer data, which facilitates the recognition of loop closures. On the other hand cameras do not provide depth information, which instead is provided by LiDAR. Regardless of the technology, the complex nature of the problem makes the implementation of a SLAM algorithm very time consuming and computationally expensive. In autonomous robot navigation, Simultaneous Localization and Mapping can be used in two ways: the robot could first use SLAM to build a map of its surroundings which then uses for autonomous navigation, or it could use SLAM directly for navigation in the environment while constructing the map. In the first case, building a map beforehand could be a huge task depending on the dimension of the environment. In contrast, planning in an unknown and not yet mapped environment would mean planning using a local and incomplete map, which implies not finding the optimal trajectory. This represents one downside. Another problem is that map-based probabilistic algorithms used for implementing localization may encounter an ambiguity in the estimated pose due to the inability to recognize the differences between two or more similar areas, and therefore the robot may get lost. Moreover, in outdoor environments and open areas where the presence of landmarks is scarce, feature-based localization is more difficult.

4.1.2 Solution

A step towards the improvement of SLAM localization can be obtained by integrating a GNSS system. The strength of GNSS is that it does not need any prior

knowledge about the navigated environment to provide a position estimate. This allows to avoid mapping before being able to localize efficiently, plus the position estimate given by the GNSS can aid the mapping process. As discussed in chapter 2, satellite positioning does not work in indoor environments because the signals are obstructed, but it is the perfect solution outdoors. It is thus complementary to map-based probabilistic localization, which functions well indoors where the environment is usually relatively small and rich of features that allow fast localization, while it suffers in open areas because of the lack of landmarks. Moreover, it can prevent the kidnapped robot problem. By fusing the pose computed by the SLAM algorithm and the one provided by the GNSS module, a more accurate and reliable estimate can be obtained. Recall that autonomous navigation requires a very accurate pose estimate, around few centimeters. Regular GNSS is not able to provide such precision as it averages an error of 1-5 meters, therefore the RTK (real-time kinematics) functionality or Differential GNSS must be used.

4.2 GNSS technology

After evaluating various options for RTK-capable GNSS systems, we chose to use for this project the ArduSimple simpleRTK2B board (depicted in figure 4.1), which contains a U-Blox ZED-F9P GNSS module. This module supports the RTK functionality and can reach an accuracy of 1 centimeter. Recall that in order to use the RTK feature, a base station that transmits coordinates corrections is required. Some companies offer a subscription correction service, but to avoid extra costs we decided to use our own base station. This was simply achieved by buying two sets of GNSS receiver and antenna, and using one set as base station. Through a survey-in procedure, the receiver placed in a stationary position can determine with high precision its location, which then uses to compute the correction data. This process involves a minimum observation time during which a number of fixes are

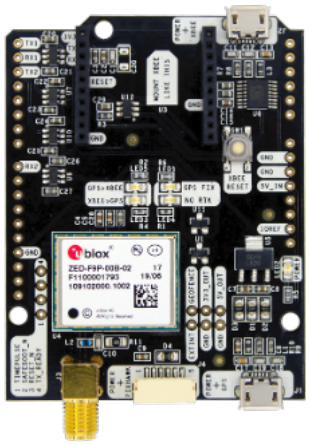


Figure 4.1: SimpleRTK2B board from ArduSimple

accumulated and then combined through a weighted average to obtain an estimate of the receiver position. The protocol used to send correction information is called RTCM. In order to compute the RTK position, both receivers need to have visibility on at least one common GNSS constellation. For this reason the RTCM corrections are only valid up to a certain distance from the Base Station. For units which are very close to each other, corrections can be send through a cable connecting them. This is obviously not applicable in case of mobile robots, therefore radio-frequency wireless communication is commonly used. The peculiarity of using radio links is that a direct line of sight between base and rover is needed in order to work, and this might be a problem depending on the environment. A different solution would be to send RTK corrections over the IP network, which however would require an internet connection. For the time being, we decided to use a radio communication to send the corrections.

Once the base station is set up, it starts to send RTCM corrections to the rover, which is placed on the robot. If we put the base station in a position such that there is a direct line connecting the rover receiver to the stationary receiver, then the robot is able to get the corrections and compute the position with a centimeter accuracy.

As explained in section 2.4, RTK positioning needs to resolve an ambiguity related to the carrier-phase of the satellite signal. The F9P GNSS module has two RTK modes available:

- RTK float, in which the rover will estimate the ambiguities as float but will make not try to fix them.
- RTK fixed, where the rover will attempt to fix ambiguities.

The fixed ambiguity solution has a much higher precision than the float solution, but the convergence time is worse[14]. The ZED-F9P receiver will attempt to provide the best positioning accuracy depending on the received correction data. As soon as it receives an input stream of RTCM corrections, it enters the RTK float mode. Once the rover resolves the carrier-phase ambiguities, it enters RTK fixed mode. In this mode, the relative position accuracy between base and rover can be expected to be correct to the centimeter level; therefore, if the base station surveyed position is accurate to the centimeter, also the position computed by the rover will have an uncertainty of few centimeters.
After reading the F9P module manual, the rover board has been configured to output the computed position coordinates on the USB port through a NMEA message. The NMEA protocol is a textual serial communication protocol used to output satellite data. The output NMEA message we are interested in is the GGA message type, which outputs the geodetic coordinates (latitude, longitude, altitude) computed by the receiver. The message is processed by the robot's software that transforms the geodetic data in coordinates local to the robot's world frame (see section 2.5), so that the GNSS position estimate can be used together with the data from other localization sources.

4.3 Data fusion

Recall that the objective of this project is to combine the pose estimate of the SLAM algorithm with the estimate given by the GNSS. After reviewing the state of the art about data fusion techniques[15], we have opted for the application of an Extended Kalman Filter. Indeed, fusion of absolute poses can be done through standard Bayesian filters considering measurements from multiple sensors in the measurement update step, as for example has been carried out in some scientific papers [16, 17, 18]. These probabilistic estimation methods exploit uncertainty to obtain a more reliable estimate, so they benefit from the use of multiple observations, since probability theory proves that the uncertainty of the fusion estimate can only be smaller than the original values. Recall that the Kalman filter assumes that the pose can be approximated by a Gaussian probability distribution. In our case this assumption is reasonable since the GNSS gives one position estimate, which we can assume to have a normal error distribution, and the SLAM's pose distribution can be assumed to be a single-hypothesis Gaussian function around the algorithm's best guess.

Since the SLAM algorithm and the GNSS device might output the estimated positions asynchronously and at different frequencies, they cannot be fused together at the same time instant; instead, they have to be integrated over time. To do so, we have implemented a Kalman filter that updates its current estimate every time a new pose is provided by either SLAM or GNSS, regardless of their order. The working of the filter, shown in figure 4.2, is fairly simple: when a new pose z_t is measured, the filter makes a prediction $\hat{x}_t$ starting from the previous fusion pose x_{t-1} based on the time interval between the two values, and then updates it with the measured pose. Thus, the fusion of GNSS and SLAM estimates occurs over time, as a Kalman filter cycle is run for each measured pose, regardless of the source. The implementation of the used filter, described in [19], assumes an

omni-directional motion model derived from Newtonian mechanics, which uses the velocity measurements to compute the prediction. The measurement model, instead, assumes that sensors provide measurements directly of the pose, therefore there is no transform function between state and measurement space.

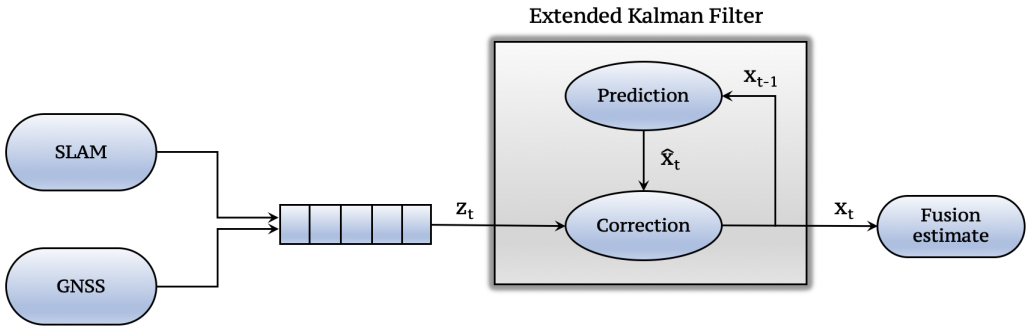


Figure 4.2: Fusion diagram: when at time t a new measurement pose z_t is observed, a prediction $\hat{x}_t$ is made from the previous fusion estimate x_{t-1} based on the time interval between time t and time $t-1$, and on the robot's previous velocity. Then the pose z_t is used to update the prediction and produce the current fusion estimate x_t.

4.4 Development framework

The code for this project was developed in order to work on the *Robot Operating System*[20] (ROS). ROS is an open-source framework running on Unix-like platforms that is widely adopted for robotic applications. It is a collection of software libraries and tools that aims at simplifying the development of robotic systems. Similar to what an operating system does, it implements an inter-process communication infrastructure based on a publish-subscribe message-passing model. ROS executables, called *nodes*, communicate publishing predefined messages and subscribing to communication channels named *topics*. Thus, ROS serves as an interface to help the various subsystems of a robot to communicate easily according to known schemes, and greatly simplifies the construction of a robotic system, which in general is a

hard process. Moreover, ROS tools are language and platform independent, even though the vast majority of open-source code and the main client libraries are implemented in C++ and Python. As a result, it allows to integrate software developed by other programmers into a robot system with little effort. For this reason, ROS-based code has been developed and, as mentioned in chapter 4.6, many open-source ROS packages have been used for the simulation of the localization solution.

4.5 Hardware and software schematics

The hardware components needed for localization include the GNSS device and the camera or LiDAR sensor used by the SLAM algorithm. The developed software is run on a NVIDIA Jetson Nano[21] board, to which the localization devices are attached. The small dimensions of the board and its computational power are the reasons behind the choice of employing this board on the mobile robot.
From the software point of view, other than the already present SLAM module, two other modules have been developed:

- a *GNSS localization* module, which is in charge of reading the GNSS data output by the receiver on a USB port of the Jetson Nano, transforming the coordinates into the robot's frame and publishing the result on the designated ROS topic.
- a *fusion* module, which waits for new pose data from either SLAM or GNSS software components to be published on the corresponding ROS topics, and then performs an EKF cycle to produce the fusion estimate.

Figures 4.3 and 4.4 show the software and hardware schematics regarding localization.

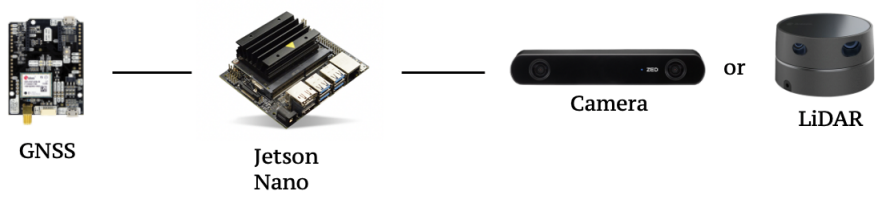


Figure 4.3: Hardware schematics

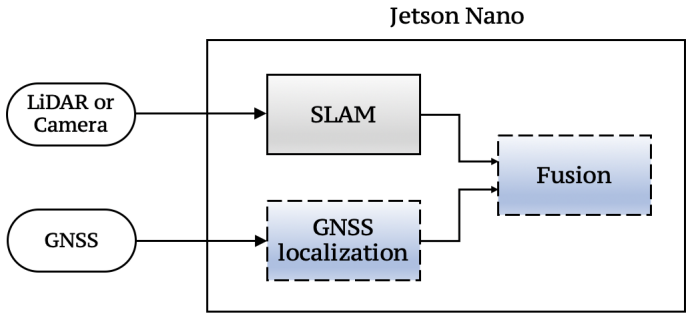


Figure 4.4: Software architecture of the components utilized for localization. The modules with the blue background are the novel components introduced in the robot system, while the grey module (SLAM) was already present.

4.6 Testing

4.6.1 Simulation setup

To test the effectiveness of the proposed approach, a software simulation has been carried out. For this purpose the Gazebo simulator[22] and the Rviz[23] tool have been used. Gazebo allows to create virtual 3D worlds, with robots, obstacles and many other objects, where physical laws gravity, inertia, etc are adhered. It offers an alternative to test the developed software on a model of the robot before proceeding to the actual implementation on the robot hardware. Rviz, instead, is

a visualization software for ROS that allows to visualize either simulated data (like sensor readings) or real data coming from a robot hardware. For this project both tools have been used. In particular, two simulations have been carried out:

- a simulation in Gazebo was done using models of the robot and the environment. The goal was to test the autonomous navigation exploiting the pose estimate provided by the fusion approach.
- a simulation using real data was run to test the operation of the localization provided by the fusion algorithm in a real-life scenario. In this case only Rviz was used to visualize the results on the environment map.

For the sake of the first simulation, a world has been created in gazebo (shown in picture 4.5) with relatively enough obstacles so that the map-based localization software always has a good estimate of the pose. The modeled robot is a differential

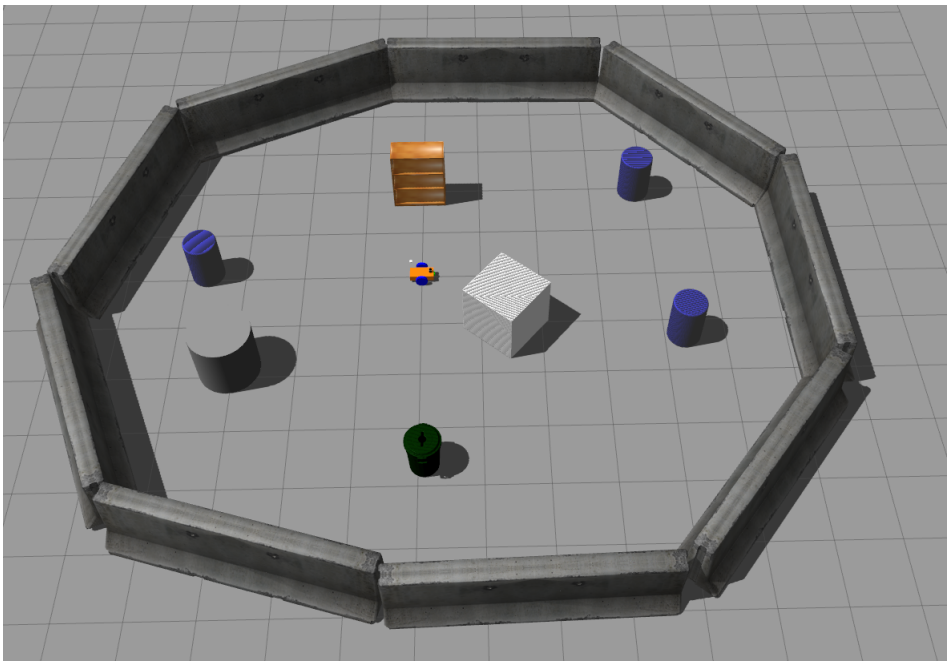


Figure 4.5: Gazebo world created for the first simulation

drive robot with two castor wheels that keep it in balance. For the simulation

few assumptions about the robot sensors and software have been made. First of all, we supposed that the robot already has a map of the environment, previously built using a laser-based SLAM algorithm called *gmapping*[24],[25]. This algorithm produces a grid map of the environment (see figure 4.6) which is used by the robot to autonomously navigate. In order to provide localization we used a laser-based

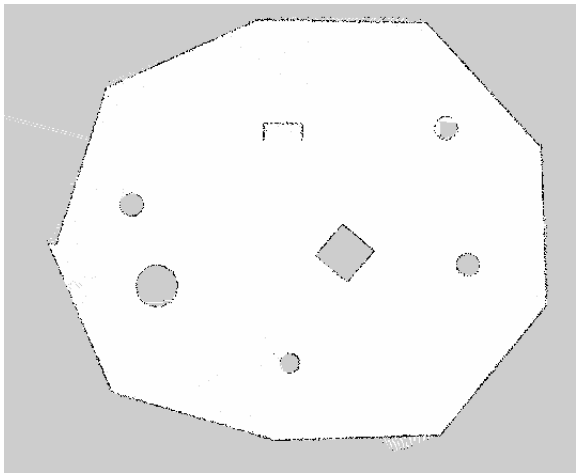


Figure 4.6: Grid map of the Gazebo world: black lines represent occupied areas, while white spaces are free.

Adaptive Monte Carlo Localization[26],[27] (AMCL) method, which estimates the path of the robot using a particle filter, as gmapping does. This choice was made for simplicity reasons: the only ROS implementation of the gmapping algorithm found did not output an estimate of the pose, but only of the map, therefore it could not be used for navigation. Obviously, since the environment is already mapped, at the start of the simulation the AMCL is more effective than the localization provided by a laser-based SLAM, however in open environments with few map features its accuracy decreases too. The choice of using a laser sensor to provide observations was encouraged by the fact that implementing a visual SLAM, as the one used on the real robot, was too complicated and was not necessary to the objective of this work, which is to demonstrate that GNSS can aid a map-based localization

method (like a SLAM algorithm) in the navigation task. In addition to the laser sensor, AMCL exploits an Inertial Measurement Unit (IMU) to have additional orientation data and wheel encoders to predict the robot motion. To reproduce GNSS data in the Gazebo environment we employed a plugin[28]. While in the simulation the GNSS coordinates are directly output on a ROS topic by the plugin, in reality this job is done by a software module that reads the coordinates from the USB port connected to the GNSS device and then publishes the read data on the corresponding ROS topic. As specified in section 4.2 of this chapter, the GNSS position estimate is given in geodetic coordinates and has to be converted into the robot's world frame. To perform the coordinates transformation we used the ROS *robot_localization*[29] package, which is the same package used to implement the EKF for the pose fusion. The last component for the autonomous navigation simulation is provided by the ROS Navigation Stack[30] package. The Navigation Stack takes as input the pose estimate from the EKF fusion and performs path planning using the map of the environment. For this purpose it constructs two costmaps (see figure 4.7), that is it assigns a cost to each grid of the map based on the presence of obstacles. A global costmap is used to compute the optimal cost collision-free path to destination, while a local costmap serves for local planning and obstacle avoidance. On the basis of the planned motion, it sends the velocity commands to the wheels controller.

In the second simulation the fusion method was tested using the KITTI Dataset[31]. This dataset contains raw data recordings captured in real-world driving situations, which include 3D point clouds, GPS, IMU and cameras information. Since this simulation takes real data, we do not need any other software other than the localization modules. For our application, we used the data acquired from navigation in a residential area, where the GNSS signal is accurate, whose map is shown in figure 4.8. As was done in [32], the ground truth position was

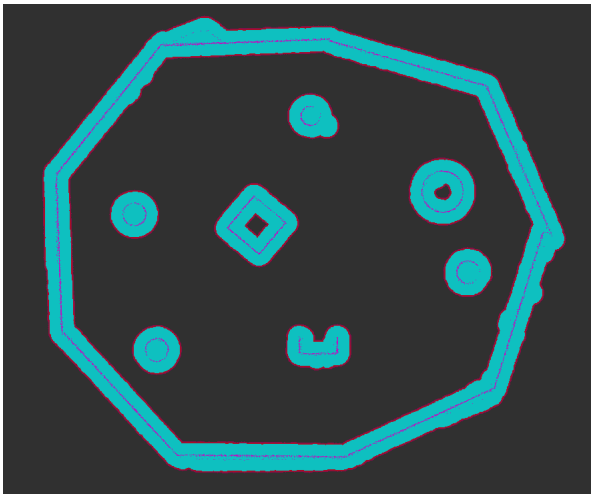


Figure 4.7: Global costmap: each cell of the map is assigned a cost value, represented in the picture by different colors. In this map the black areas are are free, while colored ones are either occupied by a real obstacle or close to it.

assumed to be the one provided by the high precision GPS module used in the recorded driving session. In order to simulate GNSS data, we added white Gaussian noise to the ground truth values. As we have done for the Gazebo simulation, the SLAM pose estimate is provided by an AMCL algorithm. The 2D laser scan measurements, used by AMCL, are obtained from the 3D point cloud raw data using the *pointcloud_to_laserscan*[33] ROS package.

4.6.2 Results

The first simulation involved autonomous navigation of the robot in a virtual world created in Gazebo using as input the fusion pose. The created environment is small (the dimension of the map is 10x10 meters), so the AMCL always has a good estimate of the robot pose and can quickly compute it. The purpose of this simulation was to test the functioning of the used EKF in a situation where GNSS and AMCL estimates have similar values; in this way we can really observe how the fusion result changes in dependence of the fused positions uncertainties. In

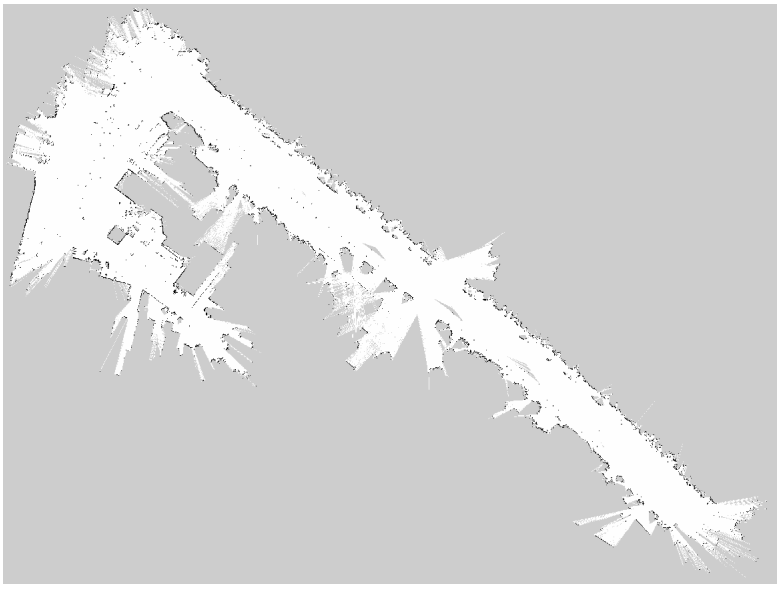


Figure 4.8: Map generated from one of the KITTI scenarios

particular, several consecutive tests have been run, making the robot autonomously move around to the same destinations in the map (see figure 4.9), while changing the position variance of the GNSS fix for each test. As expected from the Kalman

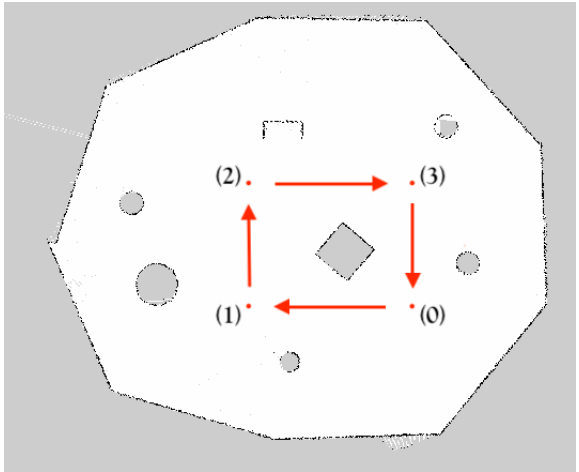


Figure 4.9: Predefined destinations for the robot navigation in test 1. The robot starts from and returns to position (0) passing through (1),(2) and (3) in order.

filter functioning, by increasing the variance from 0.001 to 0.1 m^2 we observed a decrease in error between the fusion pose and the AMCL estimate, which averages a position variance of 0.03 m^2 along the predefined path. This means that fusion pose shifts towards the AMCL estimate and movs away from the GNSS value, which we are confident to be less accurate since we increased its variance. In figure 4.10 are shown the observed average errors between the fusion result and the AMCL estimate as well as between the fusion value and the GNSS fix, where each dot represents the error obtained in a certain test. The size of the errors is small (around centimeter) because the map itself is small, thus allowing the AMCL to be very accurate.

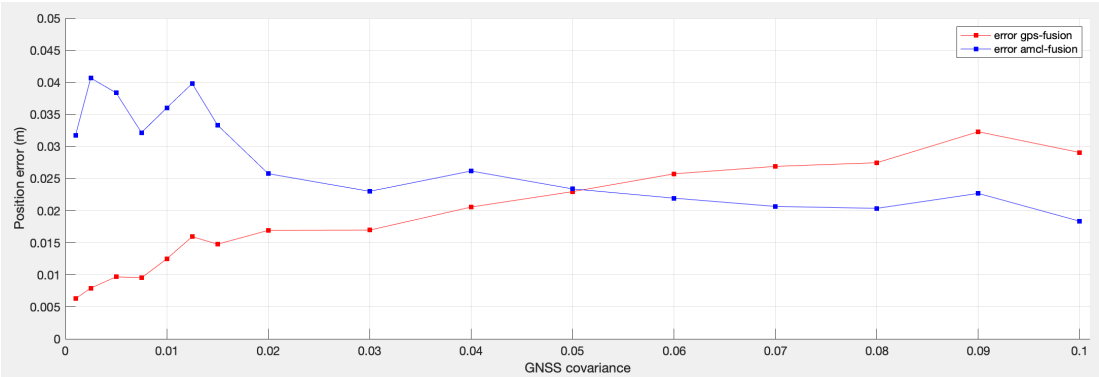


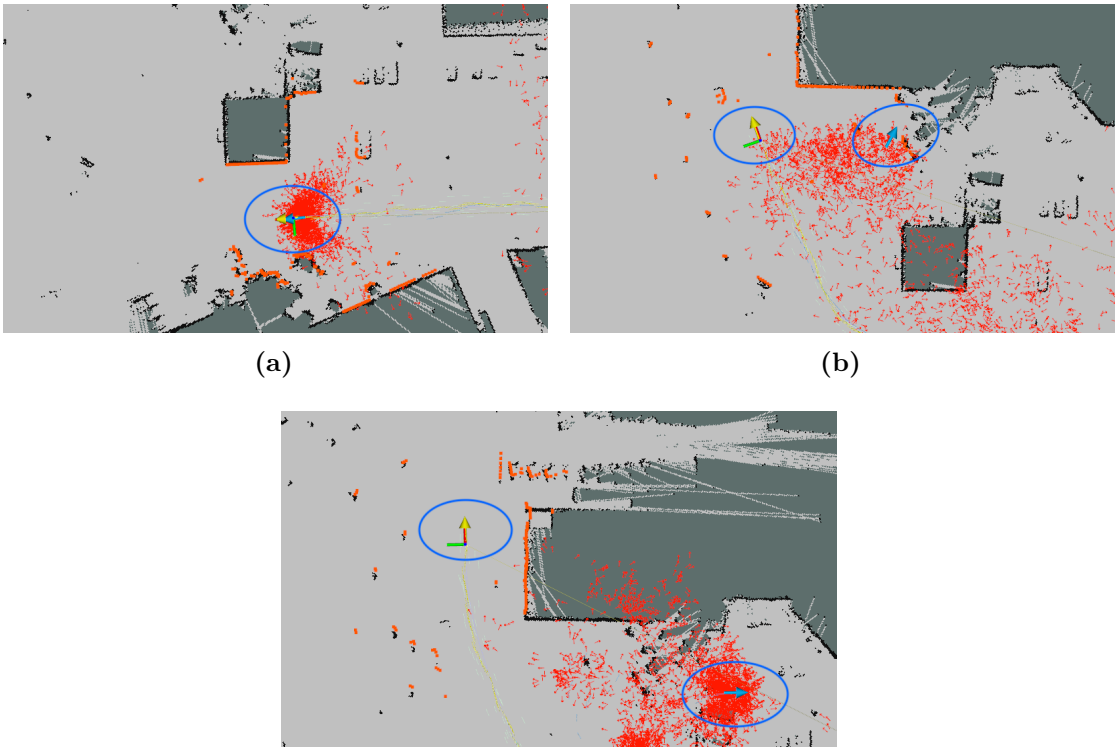
Figure 4.10: Comparison in term of position error with respect to the fusion result between AMCL estimate and GNSS estimate

Figure 4.11 shows a sequence of screenshots of the localization process taken during the second simulation. As previously mentioned, map-based probabilistic localization methods, like AMCL, compute the pose estimate by comparing and matching the map with the laser readings. The test carried out demonstrates that this kind of algorithms have difficulties in estimating accurately the pose in open environments where map features are scarce, sometimes resulting in a total loss of the robot position. At the beginning of the test, depicted in figure 4.11a, the

map contains enough obstacles to allow the AMCL algorithm to produce a reliable estimate. At this point of the simulation the current AMCL best guess (the blue arrow) and the fusion pose (the yellow arrow) are relatively close to each other compared to the dimension of the map, which is much bigger than the map used for the first simulation: the two estimates differ of about half a meter. As the simulation goes on, the robot enters an area where natural landmarks are very few and not easily distinguishable. This causes the AMCL particle cloud to consider a bigger area of possible poses, eventually loosing completely the robot true pose as shown in 4.11c. On the contrary, the fusion pose is still near the ground truth, thus proving that the fusion method provides a robust localization even when the AMCL does not work properly. The following result is obtained thanks to the logic of the Kalman filter. In fact, as discussed in section 3.4, the Kalman filter performs a trade-off among the fused values based on their uncertainty: the smaller the uncertainty, the higher a value's influence on the fusion result. Being the uncertainty of the AMCL much higher than the uncertainty of the GNSS solution (as can be see in figure 4.11c), the AMCL pose is basically not considered in the fusion; therefore, the final pose is just the result of the fusion of motion prediction and GNSS fix.

4.7 Conclusions

In the presented work we have introduced the state of the art of localization methods adopted in mobile robotics, with a special regard towards autonomous navigation. Among the various methods, much focus has been given to probabilistic techniques and to satellite navigation systems (GNSS) due to their importance in this work. Finally, it has been presented a simple solution to the problem of mobile robot localization with SLAM in featureless environments like outdoor spaces using a GNSS receiver. The implemented fusion method based on Kalman filtering has



(a) **(b)**

(c)

Figure 4.11: Localization sequence in test 2. The robot true position is represented by its x and y axis (red and green in the figures), the AMCL estimate is given by the blue arrow and the fusion pose by the yellow arrow. The GNSS fix is not visible in the figures because it has a negative z value, therefore it is hidden. In red are depicted both the AMCL particle cloud and the laser sensor data.

been tested using the KITTI dataset, which contains sensor data gathered in the real world, as well as in a software simulation run using the Gazebo tool. The results obtained from the simulations are satifying since we reached the goal of being able to localize the robot in case the SLAM alogorithm (implemented through AMCL in the simulation) is inaccurate. Future steps involve integrating the proposed solution with the present visual-SLAM algorithm and testing it together with the F9P GNSS module on the physical robot in scenarios typical to its application.

Bibliography

[1] *Global $220+ Billion Autonomous Mobile Robots Markets, 2020-2030*. Apr. 2020. URL: https://www.businesswire.com/news/home/20200409005313/en/Global-220-Billion-Autonomous-Mobile-Robots-Markets-2020-2030---ResearchAndMarkets.com.

[2] *Industry 4.0, Smart Factories and Autonomous Mobile Robots*. Mar. 27, 2017. URL: https://milvusrobotics.com/blog/industry40-smart-factories-and-autonomous-mobile-robots.

[3] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. "A review of mobile robots: Concepts, methods, theoretical framework, and applications". In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596. DOI: 10.1177/1729881419839596. URL: https://doi.org/10.1177/1729881419839596.

[4] M. B. Alatise and G. P. Hancke. "A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods". In: *IEEE Access* 8 (2020), pp. 39830–39846. DOI: 10.1109/ACCESS.2020.2975643.

[5] G. Klancar, A. Zdesar, S. Blazic, and I. Skrjanc. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Elsevier Science, 2017. ISBN: 9780128042380.

[6] R. Siegwart, I.R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2011. ISBN: 9780262015356.

[7] Salvador Malagon-Soldara, Manuel Toledano-Ayala, Genaro Soto-Zarazua, R.V. Carrillo-Serrano, and Edgar Rivas-Araiza. "Mobile Robot Localization: A Review of Probabilistic Map-Based Techniques". In: *IAES International Journal of Robotics and Automation (IJRA)* 4 (Mar. 2015). DOI: 10.11591/ijra.v4i1.6548.

[8] M. L. Fung, M. Z. Q. Chen, and Y. H. Chen. "Sensor fusion: A review of methods and applications". In: *2017 29th Chinese Control And Decision Conference (CCDC)*. 2017, pp. 3853–3860. DOI: 10.1109/CCDC.2017.7979175.

[9] S. Thrun, W. Burgard, D. Fox, and R.C. Arkin. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005. ISBN: 9780262201629.

[10] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: Jan. 1999, pp. 343–349.

[11] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". In: Nov. 2002.

[12] M. Zaffar, S. Ehsan, R. Stolkin, and K. M. Maier. "Sensors, SLAM and Long-term Autonomy: A Review". In: *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2018, pp. 285–290. DOI: 10.1109/AHS.2018.8541483.

[13] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754.

[14] P. Joosten and C. C. J. M. Tiberius. "Fixing the ambiguities: are you sure they're right?" In: 2000.

[15] D. Ursino, Y. Takama, and Federico Castanedo. "A Review of Data Fusion Techniques". In: *The Scientific World Journal* 2013 (2013), p. 704504. DOI: 10.1155/2013/704504. URL: https://doi.org/10.1155/2013/704504.

[16] Y. Deng, Y. Shan, Z. Gong, and L. Chen. "Large-Scale Navigation Method for Autonomous Mobile Robot Based on Fusion of GPS and Lidar SLAM". In: *2018 Chinese Automation Congress (CAC)*. 2018, pp. 3145–3148. DOI: 10.1109/CAC.2018.8623646.

[17] L. Wei, C. Cappelle, and Y. Ruichek. "Unscented information filter based multi-sensor data fusion using stereo camera, laser range finder and GPS receiver for vehicle localization". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2011, pp. 1923–1928. DOI: 10.1109/ITSC.2011.6082990.

[18] Daniel Perea, Javier Hernaindez-Aceituno, Antonio Morell, Jonay Toledo, Alberto Hamilton, and Leandro Acosta. "MCL with sensor fusion based on a weighting mechanism versus a particle generation approach". In: Oct. 2013. DOI: 10.1109/ITSC.2013.6728228.

[19] Thomas Moore and Daniel Stouch. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Intelligent Autonomous Systems 13*. Ed. by Emanuele Menegatti, Nathan Michael, Karsten Berns, and

Hiroaki Yamaguchi. Cham: Springer International Publishing, 2016, pp. 335–348. ISBN: 978-3-319-08338-4.

[20] *Robot Operating System*. 2020. URL: http://wiki.ros.org/ROS/Introduction.

[21] *NVIDIA Jetson Nano*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson%20-nano/.

[22] *GAZEBO Robot simulation made easy*. 2020. URL: http://gazebosim.org.

[23] *rviz*. 2020. URL: http://wiki.ros.org/rviz.

[24] B. L. E. A. Balasuriya, B. A. H. Chathuranga, B. H. M. D. Jayasundara, N. R. A. C. Napagoda, S. P. Kumarawadu, D. P. Chandima, and A. G. B. P. Jayasekara. "Outdoor robot navigation using Gmapping based SLAM algorithm". In: *2016 Moratuwa Engineering Research Conference (MERCon)*. 2016, pp. 403–408. DOI: 10.1109/MERCon.2016.7480175.

[25] Brian Gerkey. *gmapping*. 2020. URL: http://wiki.ros.org/gmapping.

[26] Dieter Fox. "KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localization". In: (Oct. 2001).

[27] Brian Gerkey. *amcl*. 2020. URL: http://wiki.ros.org/amcl.

[28] Stefan Kohlbrecher and Johannes Meyer. *hector_gazebo_plugins*. 2020. URL: http://wiki.ros.org/hector_gazebo_plugins.

[29] *robot_localization*. 2020. URL: http://docs.ros.org/en/melodic/api/robot_localization/html/index.html.

[30] *ROS Navigation Stack*. 2020. URL: http://wiki.ros.org/navigation.

[31] A Geiger, P Lenz, C Stiller, and R Urtasun. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. DOI: 10.1177/0278364913491297. eprint: https://doi.org/10.1177/0278364913491297. URL: https://doi.org/10.1177/0278364913491297.

[32] Miguel Angel de Miguel, Fernando Garcia, and J.M. Armingol. "Improved LiDAR Probabilistic Localization for Autonomous Vehicles Using GNSS". In: *Sensors* 20 (June 2020), p. 3145. DOI: 10.3390/s20113145.

[33] *pointcloud_to_laserscan*. 2020. URL: http://wiki.ros.org/pointcloud_to_laserscan.