UNIVERSITAT POLITÈCNICA DE CATALUNYA

Programa de Doctorat:

AUTOMÀTICA, ROBÒTICA I VISIÓ

Tesi Doctoral

MAP-BASED LOCALIZATION FOR URBAN SERVICE MOBILE ROBOTICS

Andreu Corominas Murtra

Directors:
Josep M. Mirats Tur
Alberto Sanfeliu Cortés

Maig 2011

Preface

This thesis is the written result after 5 years of work at Institut de Robòtica i Informàtica Industrial, a joint UPC and CSIC research institute located at Barcelona, Catalunya. The work reports the research carried out on mobile robot map-based localization in urban pedestrian environments. During this period, the author has also participate actively in some research projects involving technology, engineering and integration issues.

The author would thanks to:

Josep M. Mirats, specially for theoric discussions, thesis guiding and experimentation.

Alberto Sanfeliu, specially for looking for funding and to steer my work to an international context.

Eduard Trulls, specially for 3D model edition, experimentation and video creation.

Joan Perez, specially for framework programming, experimentation and video creation.

and to:

Juan Andrade, Josep M. Porta, Gonzalo Ferrer, Sergi Hernández, Joan Solà, Miquel Ferrer, Dizan Vázquez, Francisco Real, Luís Riazuelo, Emanuele Menegatti and Cyrill Stachniss.

Finnally, last but not least, I would thank to my wife, Maragda Estany, my parents Agustí Corominas and Rosa Murtra and my brother Bernat Corominas.

Abstract (English)

Mobile robotics research is currently interested on exporting autonomous navigation results achieved in indoor environments, to more challenging environments, such as, for instance, *urban* pedestrian areas. Developing mobile robots with autonomous navigation capabilities in such urban environments supposes a basic requirement for a upper-level *service* set that could be provided to an users community. However, exporting indoor techniques to outdoor urban pedestrian scenarios is not evident due to the larger size of the environment, the dynamism of the scene due to pedestrians and other moving obstacles, the sunlight conditions, and the high presence of three dimensional elements such as ramps, steps, curbs or holes. Moreover, GPS-based mobile robot localization has demonstrated insufficient performance for robust long-term navigation in urban environments.

One of the key modules within autonomous navigation is *localization*. If localization supposes an a priori map, even if it is not a complete model of the environment, localization is called *map-based*. This assumption is realistic since current trends of city councils are on building precise maps of their cities, specially of the most interesting places such as city downtowns. Having robots localized within a map allows for a high-level planning and monitoring, so that robots can achieve goal points expressed on the map, by following in a deliberative way a previously planned route.

This thesis deals with the mobile robot map-based localization issue in urban pedestrian areas. The thesis approach uses the particle filter algorithm, a well-known and widely used probabilistic and recursive method for data fusion and state estimation. The main contributions of the thesis are divided on four aspects: (1) long-term experiments of mobile robot 2D and 3D position tracking in real urban pedestrian scenarios within a full autonomous navigation framework, (2) developing a fast and accurate technique to compute on-line range observation models in 3D environments, a basic step required by the real-time performance of the developed particle filter, (3) formulation of a particle filter that integrates asynchronous data streams and (4) a theoretical proposal to solve the global localization problem in an active and cooperative way, defining cooperation as either information sharing among the robots or planning joint actions to solve a common goal.

Resum (Català)

Actualment, la recerca en robòtica mòbil té un interés creixent en exportar els resultats de navegació autònoma aconseguits en entorns interiors cap a d'altres tipus d'entorns més exigents, com, per exemple, les àrees urbanes peatonals. Desenvolupar capacitats de navegació autònoma en aquests entorns urbans és un requisit bàsic per poder proporcionar un conjunt de serveis de més alt nivell a una comunitat d'usuaris. Malgrat tot, exportar les tècniques d'interiors cap a entorns exteriors peatonals no és evident, a causa de la major dimensió de l'entorn, del dinamisme de l'escena provocada pels peatons i per altres obstacles en moviment, de la resposta de certs sensors a la il.luminació natural, i de la constant presència d'elements tridimensionals tals com rampes, escales, voreres o forats. D'altra banda, la localització de robots mòbils basada en GPS ha demostrat uns resultats insuficients de cara a una navegació robusta i de llarga durada en entorns urbans.

Una de les peces clau en la navegació autònoma és la localització. En el cas que la localització consideri un mapa conegut a priori, encara que no sigui un model complet de l'entorn, parlem d'una localització basada en un mapa. Aquesta assumpció és realista ja que la tendència actual de les administracions locals és de construir mapes precisos de les ciutats, especialment dels llocs d'interés tals com les zones més cèntriques. El fet de tenir els robots localitzats en un mapa permet una planificació i una monitorització d'alt nivell, i així els robots poden arribar a destinacions indicades sobre el mapa, tot seguint de forma deliberativa una ruta prèviament planificada.

Aquesta tesi tracta el tema de la localització de robots mòbils, basada en un mapa i per entorns urbans peatonals. La proposta de la tesi utilitza el filtre de partícules, un mètode probabilístic i recursiu, ben conegut i àmpliament utilitzat per la fusió de dades i l'estimació d'estats. Les principals contribucions de la tesi queden dividides en quatre aspectes: (1) experimentació de llarga durada del seguiment de la posició, tant en 2D com en 3D, d'un robot mòbil en entorns urbans reals, en el context de la navegació autònoma, (2) desenvolupament d'una tècnica ràpida i precisa per calcular en temps d'execució els models d'observació de distàncies en entorns 3D, un requisit bàsic pel rendiment del filtre de partícules a temps real, (3) formulació d'un filtre de partícules que integra conjunts de dades asíncrones i (4) proposta teòrica per solucionar la localització global d'una manera activa i cooperativa, entenent la cooperació com el fet de compartir informació, o bé com el de planificar accions conjuntes per solucionar un objectiu comú.

Contents

Chapter 1

Introduction

This chapter introduces and motivates the reader in the subject of mobile robot localization in urban pedestrian environments. After a general motivation section, the chapter places localization in the process diagram of autonomous navigation for mobile robots and introduces the urban pedestrian environment and its particularities for mobile robot localization. A fourth section outlines the thesis approach and the working context. The introduction concludes with a thesis overview showing, chapter by chapter the associated scientific publications issued from the research work of this thesis.

1.1 Motivation

Quality of live in urban environments is becoming a major issue for city governments. After industrial revolution of XIX c. and XX c., nowadays we see how cities are being reinvented and yet evolving, taking great attention on some issues provoked by population density such as pollution, mobility or trash management. Moreover, economical trends are moving to a production with a key importance of technology and knowledge, while human force work devaluates since it rarely provides an added value. In this context, robots appear as a good alternative to substitue humans in hard and tedious tasks. Industrial robots were introduced in production plants since some years ago and they have successfully replaced humans in some operations requiring high power, extreme precision or long endurance. Today, at XXI c., mobile robots are being prepared to arrive at city downtowns to work on cleaning, mailing, good deliverance, taxi, guiding, surveillance, monitoring and so on.

However, current mobile robots are not yet ready to be deployed in an urban area. Beyond some important legal issues, they are not fully operational in such environments, mainly due to the fact that it doesn't exist yet a solution for a *safe* and *reliable autonomous navigation* in urban pedestrian settings. Safe navigation stands for keeping integrity of the robot itself, pedestrians and other static and dynamic objects around the robot. Reliable navigation refers to the ability of the robot to reach target positions, or tracking a given path, with a high confidence level, enough for an useful operative high-level planning. While some robot prototypes over the world have achieved successful results in terms of reliability and safety, technology is yet quite far from a closed product ready to navigate in urban pedestrian areas.

1.2 Map-based autonomous navigation

So, autonomous navigation is a fundamental ability for mobile robots trying to provide a set of services in urban pedestrian areas. High level applications such as goods transportation, guiding, taxi or surveillance are based on a robust navigation service. Since urban environments are usually well mapped areas, specially interest zones of downtowns, this thesis is developed in the context of *map-based* autonomous navigation framework. Map-based navigation considers that the navigation system is provided with some previous model of the environment, also called a map. The map-based autonomous navigation framework considered in this thesis is shown in figure 1.1. It is organized in two control loops, a *reactive* and a *deliberative* one.

The reactive loop is in charge of reaching goal positions expressed in terms of the robot local frame while avoiding obstacles perceived by the on board sensors. To this end, typically two processes are developed: a local planner and a motion controller. The first one, the local planner, gets a local goal in terms of robot coordinates and a set of real time data coming from sensor acquisition and perception modules. With these inputs, the local planner should provide to the motion controller an obstacle-free local goal point. With this obstacle-free local goal, the motion controller computes the appropriate motion commands to reach it, typically translational and rotational speeds, and sends them to the platform.

The deliberative loop is in charge of guiding the robot through a series of waypoints expressed with respect to the map coordinate frame, forming the path computed by the path planning module after a *go to* request sent by some upper-level process, or directly sent by an operator. Therefore, localization is a fundamental capability of a mobile robot trying to navigate in a way beyond a simple *reactive* behavior. Localization is the process that estimates the position of the robot in a given reference coordinate frame. The localization estimate is critical since it closes the *deliberative* navigation loop, allowing the platform driving processes (reactive loop) to take decisions on the commands to send to the motors with the aim of following a given path expressed in terms of the reference coordinate frame. A mobile robot not computing a localization estimate of its position would be able, at most, to navigate in a reactive way to, for instance, follow a person in the robot's field of view, go to a given environment feature within the sensor horizons or move randomly while avoiding obstacles.

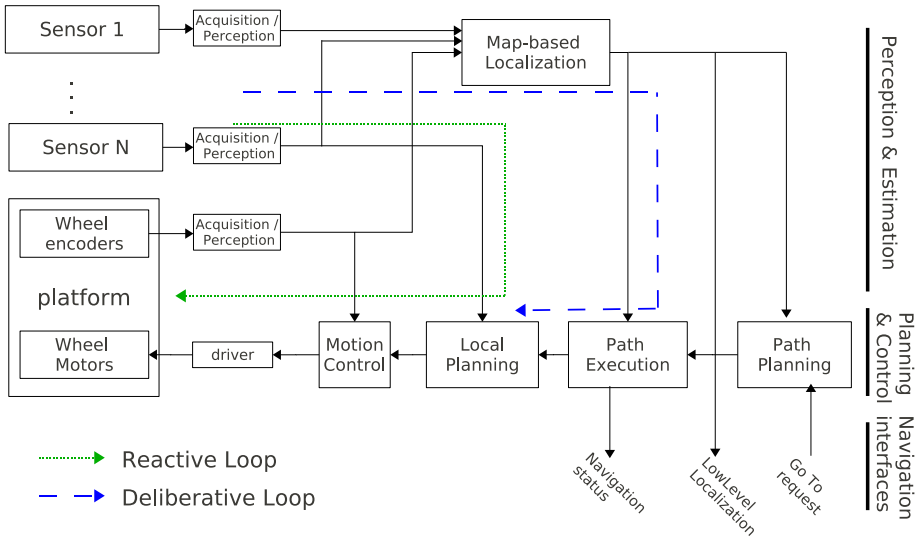


Figure 1.1: Process diagram of map-based autonomous navigation.

1.3 Mobile Robot Localization in Urban Pedestrian Environments

Previous section has explained the important role of localization in the navigation deliberative loop. Localization is a high level perception process that fuses information from a set of sensors while using the environment model, i.e. the map, to check for the likely of those sensor readings. Localization in urban pedestrian environments is challenging due to the following reasons: the outdoor nature of the environment, the dynamics caused by pedestrian traffic and other moving objects and the partial availability and poor accuracy of GPS in such areas. These issues constrain the localization problem and make research on mobile robot localization for urban pedestrian areas an open scientific and

technological field. The following paragraphes describe how these three items difficult the localization task.

Outdoor environments Outdoor environments impose challenging issues to robotic systems, specially due to the fact that sensors could be out of the range of their operative environmental conditions. This is specially critical for camera sensors since variations of lighting conditions cause difficulties on computer vision methods for localization and navigation. Laser scanner devices can be also affected by fog conditions, humidity, dust or even by direct or reflected incidence of sun light rays.

Besides these environment conditions, urban areas have usually elements such as ramps, steps, vegetation, or others with curved line edges which makes necessary to navigate and localize in 3D. This *semi-structured* geometry has to be taken into account when designing environment models or when evaluating possibilities of feature extraction from sensor readings. Moreover, a wide set of material textures are find in such environments, from vegetation to glass surfaces passing through feature-less walls, and common ranging or vision sensors can experience troubles in such variety of conditions.

Dynamic environment Dynamics of an urban pedestrian area is given by the ubiquitous presence of people, and other kind of objects such as bikes, animals such as birds or dogs, and so on. In this context dynamics does not refers only to all these moving objects but also to all objects that are not modelled in the map, thus their presence is considered as temporary, so they are also considered as part of the dynamism of the environment. Vegetation, for instance, is only partially modelled but they geometry and appearance evolves through the days, so that it is also part of the dynamism of the environment with respect to a static model.

Limitations of GPS reception GPS receivers have become popular devices due to their use in marine, mountain hiking and specially for vehicle localization. All these applications require a localization accuracy of some meters since there is always a human interpreting the GPS localization data provided by the receiver. Moreover, these devices work much better in conditions where a large part of the sky can be seen by the receiver, such as on the sea, on a highway or crossing mountain peaks. However, map-based mobile robot autonomous navigation in urban pedestrian areas requires a localization accuracy of some decimeters up to few meters, since urban environment has relevant changes in that spatial scale. A fail on the localization of two meters could cause a navigation failure if a robot should enter to a narrow passage. Moreover, in city pedestrian areas there are usually the so called *urban canyons*, passages with buildings in each side where GPS direct signal arrives weaker that the one reflected on the building surfaces. In the worst case this causes that GPS localization is not available due to receiving not enough satellite signals to triangulate. Otherwise this leads to a multipath signal reception resulting in a poor localization accuracy. Therefore, localization methods in urban pedestrian areas should consider alternative techniques to the GPS-based one.

1.4 Thesis Objectives and Approach

The main purpose of this thesis is to investigate and experiment techniques and methods for map-based localization of a mobile robot in urban pedestrian environments. Localization methods will be thought in the context of the autonomous navigation system proposed in figure 1.1. The final goal is to have a real platform with autonomous navigation capabilities to reach given points within a pedestrian area. This upper-level *navigation service* is the base to build up a mobile robot system that could offer a set of mobile services to a user community, such as cleaning, transportation of people and goods, surveillance and so on.

So this thesis addresses the map-based localization problem in urban pedestrian environments. The following list summarizes the main working principles and working context of this thesis:

- Take into account the localization role in the autonomous navigation context described above in section 1.2, so that integration of localization in a full navigation system is specially considered, taking care, for instance, of timing issues, such as real-time, latencies or time complexity.

- Use the well stablished particle filter localization approach for data fusion.

- Consider environment models that can be understandable by humans, such as geometric representations. Therefore there is no aditional design of a world representation adapted to a given robot / sensing requirements.

- Assume that the map is incomplete, although it models the most important part of the environment.

- Avoid discretization of either state or observation spaces to deal with large environments.

- In urban environments, a mobile robot is pretended to be an entity of an upper-level system, consisting on other robots, a sensor network and a communication network. Therefore, data coming from remote entities has to be considered in such cooperative environment.

- Avoid complex feature extraction steps and sensor data manipulation and give importance to data fusion algorithms.

- Perform experiments with real robots in real scenarios as far as being possible.

- Involve the research work within the projects carried out in the Institut de Robòtica i Informàtica Industrial, specially the URUS european project, to take benefit from the international consortium environment created for the project.

- Do not forget that the main goal of a PhD student is to learn.

1.5 Chapter overview and reading guide

This thesis is organized through nine chapters and six appendices. Next paragraphes outline the content of each chapter and appendix, and they guide the reader if some reading order exist. Paragraphes also list the related scientific publications of the author's thesis.

Chapter 1 is the introduction. It contains a motivation section, places the localization problem into the context of a map-based autonomous navigation system and introduces the urban pedestrian environment.

Chapter 2 reviews the state of the art on mobile robot localization, giving special attention to map-based techniques in urban environments.

Chapter 3 defines *sensor reading* and *real observation* and lists sensor devices used through this thesis detailing the data they provide.

Chapter 4 describes the environment models and how they are used to compute *expected observations*. A technical report, [19], and one publication, [24], are related to this chapter.

Chapter 5 overviews the *particle filter*, a well stablished probabilistic technique for data fusion. This chapter does not introduce original work but it sets the notation and the algorithmic framework used in the following chapters.

Chapter 6 presents research and engineering work on *position tracking* considering both 2D and 3D cases. The chapter is based on formulation and concept definitions previously made at chapters 3, 4 and 5. Four publications are related to this chapter, [70, 22, 25, 111].

Chapter 7 formulates theoretical work on *active and cooperative global localization*, based on formulation and concept definitions presented through chapters 3, 4 and 5. Two publications are related to this chapter, [21, 23].

Chapter 8 documents the *software framework* used in this thesis for simulations and real-world executions. One publication is related to this chapter, [20].

Chapter 9 concludes the work highlighting the original contributions of this thesis. The chapter also points out future research lines identified as key trends in the near future for mobile robot localization in urban pedestrian scenarios.

Appendix A shows the two robots used in most of the experimental sessions of chapter 6, listing the on board sensors and devices and showing some particularities of two-wheeled self-balancing platforms.

Appendix B details mathematical equations and methods of a C++ class that implements a frame/position in a 3D environment.

Appendix C explains how to transform GPS data, referenced to a global world frame, to be used in a given map coordinate frame.

Appendix D lists input and derived parameters of range observation models associated to actual laser scanner devices mounted on board the mobile platforms used during experimental sessions.

Appendix E outlines the main videos showing some thesis results.

Appendix F tells something about pictures placed at the start of each chapter of this thesis, detailing the place where the pictures were taken and in which sense they are related with the chapter content. This appendix is non-technical and it has been written basically for the pleasure of the author.

Chapter 2

State of the Art on Mobile Robot Localization

Due to the key role of localization in mobile robot autonomous navigation, the scientific community has paid great attention to this topic, resulting in an enormous amount of work. Organizing this vast and heterogeneous *mobile robot localization library* is a challenging task. Firstly, this chapter identifies the key issues of mobile robot localization approaches. Then, a classification of different mobile robot localization implementations is proposed, in order to better understand them and with the aim of putting the work presented in this thesis in the correct place within the state of the art of mobile robot localization.

2.1 Key aspects of mobile robot localization

Six key aspects have been identified to play an important role during the design and development of a mobile robot localization method. These key points are: (1) the nature of the environment where the robot operates, (2) how the environment is modelled to build the so called spatial representation, environment model or map, (3) the sensor setup on board the mobile platform, (4) the kind of mobile platform from the point of view of its kinematics and dynamics, (5) the representation of the uncertainty, strongly linked with the data fusion method used, and finally, (6) the cooperation arised when a robot can share information with other robots or sensors in the environment or when a set of robots can plan and execute joint actions to solve a common goal.

2.1.1 Nature of the Environment

The nature of the environment where the robot navigates is a very basic start point to design a mobile robot localization system. Although researchers want to propose generic solutions, they need to identify main properties of this environment as the size, the possibility to modify the environment placing artificial landmarks, its indoor or outdoor nature, whether it is structured or not and if there are dynamic objects moving on it.

The real *size* of the environment is a critical aspect to have into account. Environment size directly defines the size of the state space, that is the algebraic space where the estimated position is found. Moreover, large environments like urban areas have to be represented in a way taking care about memory and computational resources, so that the size of the environment also affects on how the environment can be modelled.

Unmodified environments offer the most challenging field of research since autonomous mobile robots have to navigate without aid of artificial landmarks that could act as reference beacons. Deploying mobile robot systems on unmodified areas has economical benefits since no landmark installation and maintenance is required. For instance, this is of special interest for city councils planning to deploy such systems in urban areas, thus the most research efforts are put on localization approaches that do not modify the environment.

The *indoor* or *outdoor* nature of the environment is also a key point, specially when taking decisions about which sensors embark onto the mobile platform. For instance, GPS availability and hard illumination changes for vision systems are two aspects to take into account in outdoor scenarios.

Structured or *unstructured* environments will force to design spatial representations adapted to them, or to design perception steps to detect some kind of environment features. For instance, office environments can be usually described with accuracy using only 2D straight segments, but modelling a forest, or even a city, with only 2D straight segments can be a hard task. Semistructured environments refer, for instance, to outdoor urban areas, where an important presence of straight lines can be found (buildings, streets) but other elements escaping of this "cartesian" structure can be also found, like trees, ramps, steps, flowerpots, monuments, open areas and so on.

Finally, the *dynamic* part of the environment also offers a great challenge. In map-based implementations, dynamic environments force the assumption of the incompletness of the map, since robot perception can detect objects that do

not exist in the model, and moreover these unmodelled objects can be occluding other modelled regions of the map.

There exist a lot of examples in the literature proposing localization approaches in a variety of environments. Among them, it can be found methods for indoor office or home environments [77, 104, 13, 17, 34, 43, 66, 115, 119, 54, 106, 63], for highly dynamic indoor environments like an exhibition pavilion [4], a museum [14, 108, 35, 110], a shopping mall [39] or a robot soccer field [65], for outdoor university campus areas [107, 52, 117, 25], for urban pedestrian settings [38, 64, 71, 111], for urban roads [55, 95, 60], for outdoor industrial environments [78], for forestry roads [58] and, finally, for planetary environments [81, 51, 62].

2.1.2 Environment Model

In the localization context, the environment model, also called *the map*, is the representation of the real world used by the localization algorithm to compute which observations are expected given a position in that environment or to compute some physical constraints such as elevation or expected collisions. Whether the localization is based on an already created map or a map is built while navigating, localization methods have to address how the environment is modelled.

Grid models were among the first models used in mobile robotics. They divide the physical space in a grid and label each cell following some criteria. The most common are the occupancy grids that label each cell as being occupied or not, or with a probability of occupancy [30, 54]. Other grids represent a terrain by its elevation such as [51, 50], building the so called elevation maps. Other grids register each cell with a certain sensor measurement such as [55] where each cell is labelled according to a reflectance measurement of the surface, issued from a LIDAR device, creating a reflectance image that models the environment.

Geometric models describe the environment reproducing its geometry either partially or fully. Examples describing the world by 2D geometric instances such as segments, corners or door openings are [43, 3, 117]. Also works in 2D, but following some standard mapping format such as Geographical Information Systems (GIS) are [70, 25]. Geometric representations in 3D can be found in [38, 78, 24, 111].

Topological models represent the world as a graph where the nodes are ususally places such as trajectory corners, waypoints, corridor crossings or halls, and links are stablished when nodes have some relation in terms of metric proximity or action [104, 52]. Some authors have also proposed hybrid approaches such as [4, 48, 119], where both metric (geometric or grids) and topological information is used to represent the environment.

Appearance maps model the environment with a set of sensor views logged by the robot during a setup phase of development. Afterwards, during navigation, the appearance map is used as a reference set to decide which is the best matching with the current sensor readings. Such maps have been used, basically, for visual localization as, for instance, in [66, 91, 115, 1].

Beyond the model used to represent the environment, some authors make a very realistic consideration about the incompleteness of the model, in the sense that the map does not model exhaustively the environment, thus some sensed objects or parts of the environment can be missing in the map [117, 82, 25, 54, 111]. This is of special interest in dynamic environments such as offices, homes

or cities where a coarse map is usually available (up to some fidelity), but a precise and complete map is often unavailable and hard to build.

2.1.3 Platform Type

Mechanics, kinematics and dynamics of the mobile platform are also key points to take into account when designing localization methods for mobile robots. Two big families can be divided following the mechanical principles of platform motion: legged platforms and wheeled ones. Legged platforms have important advantages in rough terrains [93]. Moreover, in other cases they can keep the human appearance that can be interesting for some applications [101]. In the other hand, wheeled platforms offer advantages, specially in terms of energy and simplicity in both mechanical design and kinematic and dynamic modelling. Wheeled platforms can be also classified into two groups: steered and differential. Steered wheeled platforms refer to all car-like vehicles such as those used in [55, 95, 78, 58], as well as automatic wheelchairs such as the used in [52]. Differential wheeled platforms are those vehicles that can turn on the spot by commanding oposite velocities to right-side and left-side wheels. These platforms are among the most used in mobile robotic research due to practical benefits [13, 14, 108, 4, 38, 56, 71, 63]. However, turning with differential platforms can cause high wheel slip, even in slow velocities, and specially in four-wheeled platforms. Another kind of differential wheeled platform that reduces wheel slips is the two-wheeled self-balancing one, used in [64, 75, 25, 106, 111]. It offers high mobility and payload but causes some perception and control issues due to its pitch-free behavior (see appendix A). Beyond classical approaches, mechanical researchers and engineers have produced other wheeled platforms such as, for instance, an amazing differential wheeled platform with only one *ball* wheel, the Ballbot [74].

Independently to mechanical construction, speed and dynamics at which the platform moves are also key issues to take into account. Localization process should iterate at a rate enough to estimate all platform motion relevant for its navigation and control. For instance, localization processes iterate at medium rates of $1 - 10Hz$ in [38, 25, 111]. Other faster platforms [56, 55, 95] require high localization rates ($> 10Hz$) to allow navigation processes to compute next driving actions.

2.1.4 Sensory System

A mobile robot is usually equipped with a set of sensors to solve the localization task. The following list presents and briefly discusses the most commonly used sensors for localization in mobile robots:

Encoders: robot motors are usually embedded with an encoder device that counts the number of turns of the motor axis, which can be converted to wheel turns by means of a constant given by the gear setup. With this data, and with a model of the robot kinematics, we can make an estimate of the robot motion, a very useful information for short-term prediction of the position. However, model innacuracies and incremental computations do not assure bounds on the error of this calculated motion. Works characterizing the odometry error are [45,

69, 8], and an example of a method to global localize a robot in an idealized geometric environment using only odometry data can be found in [80].

Accelerometer: Measuring linear accelerations can be useful for weighted platforms or for that driving at large velocities. Accelerometer data can improve motion estimation when is used with a kinematic and dynamic model of the robot platform.

Gyroscope: Gyroscope device provides rotational speeds around an axis. Platforms equipped with three gyroscopes, mounted following the three axis of the local frame, can measure the rotation rates of heading, pitch and roll.

IMU: Inertial measurement unit is usually referred as an embedded device where three accelerometers and three gyroscopes are mounted following the three orthogonal dimensions of the space, so that the device outputs linear accelerations within each of the three space components, and rotation rates in each of the three axis. These devices have become popular in navigation systems, specially for high-speed or high-precision applications such as [55, 95, 71, 58, 46, 63]. The main advantage of such devices is the fact that they provide a set of sensor data fully synchronized by hardware means.

Inclinometer: This device measures inclination of the surface where it is mounted on with respect to the gravity vector. It can be very useful in some situations where three dimensional perception data is mandatory for robust localization [111].

Compass: Electronic compass is a device that can be very useful, since its raw data gives directly the orientation of the platform with respect to the magnetic north vector. However, special care has to be taken when attaching a compass near computers or robot motors that can modify the earth magnetic field sensed by the device. Some devices offer calibration procedures to take into account the local magnetic fields in the mounting place on board the vehicle. These devices are capable to detect unexpected distortions of the sensed magnetic field and to trigger an alarm when they occur. In a mobile robot, these distortions can appear when the robot passes close to ferrous obstacle like doors, trashes and so on. A mobile robot equipped with such device for localization is [38].

Sonar: Sonar rings are widely used in mobile robotics as a range device that detect near obstacles around the robot. From sonar data, the robot can build a model of its surroundings. Sonars are adequate to measure distances up to $10m$, so they are used mainly in indoor environments. However the availability of laser scanners has placed sonar rings in a second term for accurate ranging measurements. Three cases of mobile robots using sonars for localization are [13, 35, 52].

Laser scanner: Laser scanner has become a very popular sensor in mobile robot platforms thanks to its accuracy on range and bearing measures and their usefulness in localization, mapping and obstacle avoidance. The most popular

scanners provide two-dimensional data [6, 110, 3, 25, 111]. Some researchers have equipped mobile platforms with three dimensional scanners [38, 58] or have mounted a two dimensional scanner on a tilt unit [56, 40, 113].

Cameras: Computer vision field has a vaste work concerning mobile robot navigation [27]. Robots can carry a perspective camera, a stereo one or an omnidirectional one. Cameras are passive devices since they sense the energy of light without delivering energy to the environment as sonars or lasers do. Cameras provide images that potentially cover all the mobile robotics needs for navigation. However, two important drawbacks limit their use: first, concerning its computationally expensive processing and second, the not yet solved problem of robust segmentation and feature extraction in a general case, specially for outdoor environments where illumination changes are severe. Experiments with mobile robots using cameras for localization are: with a perspective camera [115, 78], with a stereo camera [91, 117, 41] and with an omnidirectional one[66, 65, 72, 1].

GPS: Global Positioning System devices can solve directly robot location on a georeferenced map or coordinate frame for outdoor applications. However, robot researchers have pointed out two important limitations regarding its accuracy and availability [55, 117, 58] due to radiolectric shadowing in areas where obstacles avoid direct vision to satellites, or caused by the multipath reception of satellite signals, as it happens in *urban canyon* environments. DGPS has appeared as an enhanced system where well located local radio stations broadcast corrections to DGPS receivers in order to reduce, mainly, atmospheric errors [44]. However, GPS is very useful for rough localization in large outdoor environments in order to reduce search space and to bound the localization error in tracking applications. Works reported in [107, 38, 71] have equipped their robots with GPS devices.

GSM: Global System Mobile is an ubiquitous system deployed nowadays in all urban areas forming a dense network of well localized base stations that can be used as beacons for GSM devices. Research efforts are done on methods to localize mobile phones through a GSM network. In [18] an experiment demonstrates that global localization with an error of about $100m$ can be achieved by the GSM system in urban areas. GSM devices can be very useful in large urban environments where a first coarse global localization can reduce drastically the search space.

WiFi: Wireless local access networks (WLAN) are common infrastructures usually deployed in office environments and also in some outdoor areas such as university campus. A robot equipped with a wireless card can monitor the signal strength or footprint associated with the different radio access points forming the infrastructure, thus the access points act as radio beacons. Due to the complexity on modelling the signal propagation in such environments, an offline mapping step is required in order to create a *radio signal map*. This map will be used to compare online signal readings with the stored ones in order to estimate the robot position [98].

2.1.5 Representation of Position Uncertainty

Whatever the sensory system used for the mobile robot, robust methods for localization have to consider uncertainty coming from three main stochastic processes: (1) the sensor readings, (2) the innacuracies of the robot kinematics model, and (3) the failures in the data association or poor likelihoods of expected observations due to map innacuracies or environment dynamics. While the two firsts sources of uncertainty are well known, acotated and commonly modelled by means of normal stochastic processes, the later has a much more complex variability and is the source of failures that usually reduce the localization robustness. Given this intrinsic position uncertainty of the localization problem, proposed methods have to represent it by means of some mathematical tool in order to quantify localization error bounds. Having a realistic estimation of the position uncertainty is an important point in order to limit the search space of an iterative method. Moreover, for system integration purposes (see figure 1.1), estimation of the uncertainty can be also required by other navigation processes.

Figure 2.1 shows three common representations of the uncertainty in a single dimension space: the multi-gaussian representation, the complete discretization and the importance sampling of the state space.

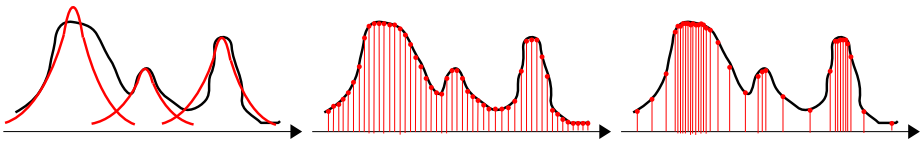


Figure 2.1: Representation of the uncertainty in a single dimension space. a) multi-gaussian, b)complete discretization (in 2D it would be a regular grid) and c) importance sampling.

Closely related to the uncertainty representation, researchers have to address the integration of the continuous data flow coming from sensor readings. The goal is to recursively estimate the robot position and its uncertainty and, therefore, a data fusion technique has to be implemented to integrate all these data. Common fusion algorithms are based on the optimal state estimation problem [109, 105], where three main approaches can be highlighted: (1) Multi-Hypothesis Extended Kalman Filter (EKF) [43, 3, 117], parameterizing the uncertainty as a multi-Gaussian variable, (2) Markov Localization (ML) [35, 81], representing the uncertainty with a regular sampling of the whole state space, and (3) Monte Carlo Localization (MCL) or particle filter, introduced in mobile robot localization by [26, 110], where a set of weighted samples in the state space (particles) is used to represent the uncertainty of the robot position.

Particle Filter Map-based Localization. Mobile robot localization using particle filters are nowadays a well stablished framework [109] and has been widely and successfully used by the mobile robotics community [66, 115, 55, 119, 78, 50, 25, 63, 54, 111]. In the basic version, called *sampling, importance, resampling* (SIR), the particle filter iterates through three steps: the first one propagates the particle set by computing a probabilistic kinematic model and outputs a *prior* density estimate. The second step weights each particle according to how likely are the current sensor observations with the expected

observations, the later computed with an observation model considered from the particle state point. This second step produces the *posterior* estimate. Finally, the third step draws a new particle set according the importance weights of each particle.

The most research efforts are to optimize the computational charge of the filter to keep real-time requirements of autonomous navigation, specially when computing the expected observations. In this direction, some authors precompute observation models over a grid of the state space [110, 91, 119] but others develop fast algorithms to compute observation models on-line, thus avoiding state-space discretization [55, 78, 25, 111]. Other authors have worked with the *auxiliary* particle filter that investigates alternative sampling strategies that better use the computational resources, avoiding to sample regions of the state space unlikely with the current observations, [110, 114, 11]. Moreover, a set of *adaptative* techniques have been developed to minimize the number of samples following a criteria based on the probabilistic distance between the prior density and that of the observation model [32, 49, 11], so that large distances lead to larger particle sets. More refined methods to generate a prior density have been proposed by [8], while other authors work on novel strategies in the resampling step to deal with model incompleteness [54].

Therefore, it seems that uncertainty representation with a set of samples over the state space, the so called particle representation, has advantages over other uncertainty representation methods, specially in terms of flexibility and robustness to cope with the stochasticity of the localization process. Therefore the mobile robot community seems to converge to that uncertainty representation and the associated particle filter for data fusion, specially for map-based localization approaches.

2.1.6 Cooperative Issues

Cooperation in robotics has atracted great attention from the last ten years[2, 31, 88]. In terms of localizaton, cooperation has been addressed specially in the context of the estimation algorithms that fuse information coming from remote observers (i.e. other robots or sensors outside the robot) [96, 33, 100, 61, 15, 22]. These works focus in cooperative issues such as uncertainty propagation or delayed information. However, planning joint coordinated actions to solve the map-based global localization problem is much less investigated in the literature [9, 23, 10].

In either case, robots need to be identified and positioned between them, so that the so called *relative localization* methods are key modules when some multi-robot approach is investigated [97, 103, 68]. These methods commonly provide range and/or bearing observations of the other detected robots that run within the closeness of the robot.

2.2 A Classification of Localization Implementations

This section proposes a classification of different implementations of mobile robot localization. This classification is based on considering the localization approach from the point of view of the overall navigation system, but it is

transversal with respect to the six key points listed in the previous section, so that each implementation deals with these key aspects in a particular way with independence of this classification. Figure 2.2 summarizes the proposed classification.

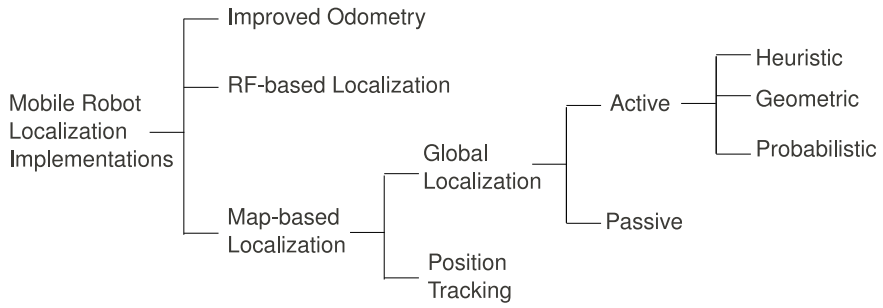


Figure 2.2: Classification of mobile robot localization implementations

Improved odometry. These methods refer to those techniques that do not use a prior map and that try to estimate as accurately as possible the metrics of the executed path of the robot from the start of the motion. These techniques usually extract laser scanner [57, 56] or visual features [76, 41, 62, 89, 67] to be used as external references to correct wheel odometry drifts, so that an improved odometry data is produced as the output of the process. In extended versions, the goal can be also to create a map of the environment following some world representation, so that a mapping process is in charge of registering the extracted features with the estimated position, so that the so called SLAM (simultaneous localization and mapping) problem arises [29].

Radio frequency based methods. These localization techniques define an origin of the coordinate frame at which the localization output will be referenced. Thereafter, thanks to the range measurement to a set of reference stations (i.e. RF-beacons, satellites), the receiver can be localized through triangulation. Some examples of such systems used to localize mobile robots are, by means of GPS [107], using a wireless LAN infrastructure [98] or using RF-beacons [39, 12].

Map-based Localization. The third group of the proposed classification places those implementations that refer the output position estimate to an external coordinate frame associated with a provided a priori map, thus the robot has to be localized within that map. This group is usually divided by those methods that solve the position tracking situation, so that a correct starting position is provided to the system, and those approaches that deal with the lost robot problem, where the robot has no knowledge about its initial position on the map and tries to converge to a single position estimate by fusing sensory information. Particle filter and Markov localization approaches can solve both problems in an unified framework [35, 26, 110, 109], while Kalman filter based approaches need an extra step to generate initial position hypotheses to be tracked by the filter [43, 3, 117].

From a practical point of view, map-based global localization requires much more computation efforts than map-based position tracking, but position tracking has hard real-time constraints when it is used within an autonomous navigation system as that diagrammed in figure 1.1. Implementations of the map-based position tracking process within a complete autonomous navigation system can be found in [13, 108, 110, 4, 38, 55, 78, 25, 111]. On the other hand, global localization approaches can be also subdivided in passive or active methods. While passive methods do not address the action planning of the robot, active methods select the best action to move the robot in order to solve the localization ambiguities, while fusing all sensory information up to a single-hypothesis convergence of the localization algorithm. Three approaches can be identified among these active strategies according to the criteria to choose the best action: heuristic, geometric and probabilistic. Heuristic approaches [47, 43, 37] select the action following an heuristic criteria given by the environment characteristics, such as following a wall or enter to doors. Geometric strategies such as [94, 80] propose a pure geometric optimization to decide which is the action that better disambiguates the multi-hypotheses situation. Finally, probabilstic approaches are introduced by [17] and reformulated by [34] using a entropy-based criteria. These approaches offer more flexibility and generality and are also investigated by [91, 118, 119, 21, 9, 23].

2.3 Main Contributions of the Thesis within the State of the Art

According section 2.1 of this chapter, the work reported in this thesis focuses urban pedestrian outdoor environments, modelling them with 2D and 3D geometric maps. The platform used for field experiments is a two-weel, self-balancing platform. Sensors fused in these experimental sessions are wheel encoders, inclinometers and laser devices, but compass, GPS and camera network are also considered in simulations. As a method to represent the position uncertainty the thesis chooses importance sampling, so it deals with the associated particle filter for data fusion. Finally, cooperative issues are investigated in either cases: sharing information and planning joint actions to solve global localization.

Beyond the above mentioned items, the main contributions of the thesis within the classification of implementations proposed in section 2.2 are placed in both map-based position tracking and global localization techniques.

In position tracking, the thesis reports long-term experiments in real urban pedestrian environments by using both 2D and 3D environment models [25, 111]. For the 3D case, it has been developed a fast and accurate technique for on-line expected range data computation that allows position tracking to run under real-time constraints [24]. In position tracking domain the thesis also investigates the fusion of asynchronous observations in the particle filter framework [22].

Global localization is addressed from a more theoretically point of view, and the thesis proposes a novel active, cooperative and probabilistc approach to solve the localization ambiguities by selecting best actions according expected remaining localization hypothesis [21, 23]. The thesis compares this approach with current existing active and probabilistc techniques in terms of computational complexity , showing the advantages that the proposed method offers.

Chapter 3

Sensor Readings and Real Observations

All robot systems that have to interact with the environment have a set of sensor devices that provide measurements of some physical variables, such as range or light, during robot's operation. Raw outputs of such sensor devices are called *sensor readings*. However, from the point of view of a robot system, some signal processing is required, ranging from trivial manipulation to complex feature extraction or tracking, in order to deliver to the estimation process only the relevant information of sensor readings. This relevant and processed information is called *observation*. Robots use these observations as the input for the localization process.

3.1 Real Observations

The term *real observation* is used in this thesis for clarifying purposes, to define the input data of the localization process. In a general approach, real observations are provided by perception processes, that can be from simple acquisition to complex estimation processes. From this definition, an *observation process* is a computer process delivering a real observation. Figure 3.1 shows the chain from the environment up to real observations. Following this definition, the localization process itself could be also an observation process from the point of view of an upper-level process requiring localization data.

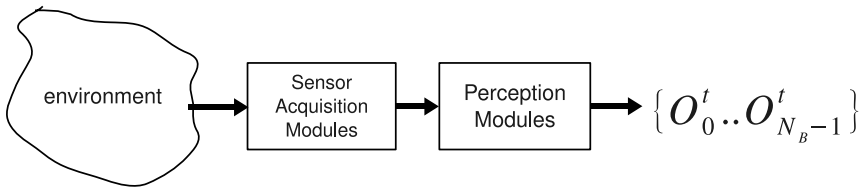


Figure 3.1: From sensor acquisition to real observations.

The localization process studied in this thesis uses input data from N_B observation processes. The observation O_k^t is the real observation issued from the k^{th} observation process, available at the estimation processing unit at iteration t. Such real observation can be defined as:

$$O_k^t = (o_k^t, C_k^t, \tau_k^t, s_k^t) \quad \forall k = 0..N_B - 1 \tag{3.1}$$

where o_k^t is the observation vector and belongs to the k^{th} observation space $\mathcal{O}_k$, C_k^t is the parameterization of the observation uncertainty, that can be expressed as a $\mathcal{O}_k \times \mathcal{O}_k$ covariance matrix, τ_k^t is the time stamp related to that observation, and s_k^t is the observation status, set to $s_k^t = READY$ for 'ready to use', $s_k^t = USED$ when the observation has been already used, and set to $s_k^t = ERROR$ for possible errors in sensor acquisition or perception processes.

3.2 Sensors and Observations for Mobile Robot Localization

This section details the sensors studied in this thesis and the real observations issued from their readings. From the point of view of an estimation process, such as the localization one, for each observation, special attention has to be taken for the following features:

- Output **rate**: period between two consecutive observations.
- Observation **latency**: Delay between the sensor reading and the moment when the observation is available at the estimation processing unit.
- **Uncertainty**: bounded, unbounded, fixed or measured on-line.
- **Availability**: some sensors only provide readings under some environmental or radioelectric conditions, partially encountered during robot operations.

Please note that, for clarity purposes, sometimes in the following sections and chapters the index k will identify each observation with a key letter instead of with numerical indexation, so that a GPS observation, for instance, can be labelled as O_G^t.

3.2.1 Encoders and Odometry

Encoders are electro-mechanical devices that count the wheel turns when a robot is runnning. In most of the wheeled robotic platforms they are embedded with motors in the gearbox of the wheel axes. Their use is vaste since they can provide easily the *odometry* (the metrics of the travel) of the platform, with a fine accuracy in short trajectories without wheel slip, a very useful information for mobile robot position tracking applications. Knowing the integration time of the wheel counter, a velocity observation can also be computed but it will be probabilistically dependent on the odometry. The main characteristics of the odometry observation are:

- High acquisition rate (typically $> 20Hz$).
- Very Low latency (typically $< 10ms$).
- Accurate precision in short displacements without wheel slipping, but unbounded uncertainty for long travels.
- Full availability in most platforms.

The observation vector for the odometry, o_U^t is defined as:

$$o_U^t = (\Delta\rho^t, \Delta\theta^t);\ \Delta\rho^t \in \mathbb{R}, \Delta\theta^t \in (-\pi, \pi] \tag{3.2}$$

where $\Delta\rho^t$ indicates the linear translation of the platform pivot point and $\Delta\theta^t$ is the angular travel of the platform heading. $\Delta\rho^t$ and $\Delta\theta^t$ are obtained by accumulating the increments from the last odometry reset up to τ_U^t. Usually odometry data is reset after integrating each odometry observation, thus the time to accumulate encoder data is between τ_U^{t-1} and τ_U^t. Figure 3.2 shows these two components between two robot positions on the 2D plane.

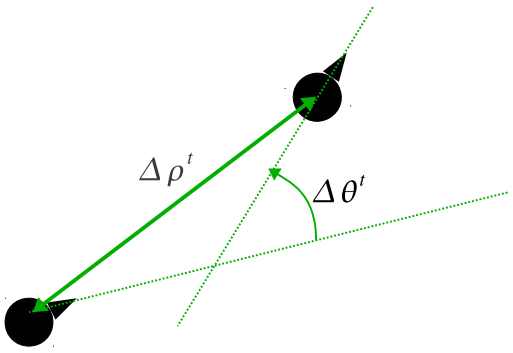


Figure 3.2: The two components of the odomety observation vector.

The uncertainty of these observations is usually parameterized with a standard deviation of the error in translation,σ_R^t, and a standard deviation of the

error in rotation, σ_θ^t. Both standard deviations are usually modelled as a fraction of the observed translation and rotation.

$$C_U^t = (\sigma_\rho^t, \sigma_\theta^t) \tag{3.3}$$

where $\sigma_\rho^t = \epsilon_\rho \Delta_\rho^t$ and $\sigma_\theta^t = \epsilon_\theta \Delta\theta^t$. This uncertainty model represents the odometry error as a normal density with linear increase of standard deviation as odometry measure grows up.

3.2.2 Inclinometers

Inclinometers measure the angles between the gravity vector and the normal vector of the platform, allowing to get directly the pitch and roll observations of the platform where the device is mounted on (see figure 3.3). The observation vector provided by an inclinometer is:

$$o_I^t = (\phi_I^t, \psi_I^t);\ \phi_I^t, \psi_I^t \in (-\pi, \pi] \tag{3.4}$$

The main characteristics of the observation provided by inclinometers are:

- High acquisition rate (typically $> 20Hz$)
- Very Low latency (typically $< 10ms$)
- Full availability while the sensor is within the operational tilt range.

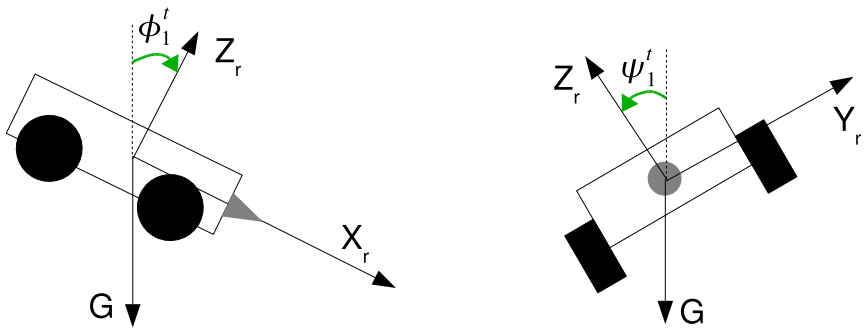


Figure 3.3: Inclinometer data as pitch (left) and roll (right) of a vehicle with respect to the gravity vector G. Z_r is the platform's normal vector.

The uncertainty of pitch and roll data issued from an inclinometer is represented with a time-invariant gaussian model with standard deviations, σ_{ϕ_I} and σ_{ψ_I}, usually provided directly by the sensor manufacturer in the device datasheet.

3.2.3 Global Positioning System

Global Positioning System is a large infrastructure widely used around the world. The system is formed by more than twenty satellites and the ground stations to support their operations. GPS receivers get the satellite signals and track their orbital positions. When a receiver gets signal from more than three

satellites it can compute its location in terms of geographic or some other Earth coordinates. The accuracy on computing the receiver location will be strongly related to some measurable and unmeasurable factors. With the measurable factors most of receivers can compute a model of the position uncertainty and output it online, as a covariance matrix of a gaussian error model. To overcome some of the error sources, the Differential GPS, DGPS, is also widely used for applications requiring more accuracy. DGPS is based on ground stations that know their geographic position thanks to precise topographic measurements. These reference stations broadcast through some radio network the real-time correction encountered between the measured GPS position and the known position. Since DGPS stations are in good places in terms of satellite visibility, these GPS deviations are assumed to be provoked mainly by atmospheric factors. Since it is also assumed that atmosphere doesn't change around small areas, GPS receivers in proximity to a differential reference station can also receive correction data and use it to improve the position estimate. Table 3.1 summarizes the error sources of the GPS localization, indicates if they are measurable for a single receiver and specifies if DGPS can overcome their effects.

Table 3.1: GPS error sources

Source	Measured at receiver	DGPS Correction
Number of visible satellites	Yes	No
Geometry of the triangulation	Yes	No
Atmospheric conditions	Modelled	Yes
Multi-path reception	Some receivers	No
Electronic and antenna noise	Modelled	No

Table 3.1 highlights the mulipath reception as one of the critical error sources since it can only be partially mitigated by dedicated signal processing implemented only on some receivers. This issue is critical in urban pedestrian environments, since narrow streets act as urban canyons where reflected signals usually arrive with greater strength to the receiver than those following the direct path. This leads to errors in the position computations due to the fact that the reflected signal has covered a longer distance than the actual range from the satellite to the receiver. A second important issue to take into account in urban pedestrian areas is the GPS coverage. Urban streets often have partial visibility to satellites, leading GPS receivers to get signal from an insufficent number of satellites to compute the triangulation.

Most GPS receivers output also the velocity from Doppler measurements of the received signals. The velocity measurement is independent in terms of magnitude from the position output but not in terms of the velocity direction [44]. With these velocity measurements, some receivers incorporate a filter that improves the estimate of the position and overcomes the lose of coverage for short periods. However if a GPS receiver is within an uncovered area for a long time the estimated uncertainty of that filter will increase up to large unbounded values.

The observation provided by a GPS device is

$$o_G^t = (x_G^t, y_G^t, z_G^t, vx_G^t, vy_G^t, vz_G^t) \tag{3.5}$$

This observation is characterized by:

- Medium acquisition rate (typically $1 - 10Hz$)
- Very Low latency (typically $< 10ms$)
- Accuracy of absolute localization depending on triangulation quality and multi-path propagation, both related to sky visibility.
- Partial availability, strongly related on environmental conditions provoking occlusions of satellite line of sight.

Appendix C details how to transform the GPS data, in terms of Earth coordinates, to positions with respect to the reference map coordinates used in this thesis.

3.2.4 Electronic Compass

State of the art 3D electronic compasses are accurated sensors that measure the three dimensional Earth magnetic field. They also measure the pitch and roll position of the sensor with respect to the gravity by means of two internal inclinometers. By using the magnetic and inclinometer data, they compute the heading of the device with respect to the magnetic north. This procedure is also known as *electronic gimbaling* opposite to the mechanical gimbaling of traditional compasses.

$$o_C^t = \theta_C^t;\ \theta_C^t \in (-\pi, \pi] \tag{3.6}$$

The main characteristics of an observation provided by an electronic compass are:

- Medium acquisition rate (typically $10 - 20Hz$)
- Very Low latency (typically $< 10ms$)
- Accurate observation on the heading component of the state. They usually output the inclinometer data too.
- Partial availability, depending on environmental conditions.

The data availability of such sensors is clearly the weakness since they are very sensible to local variations of the Earth magnetic field. Such variations can be provoked by ferrous objects or motors being close to the sensor. Modern electronic compasses provide an automatic calibration procedure based on taking a set of calibration samples in different vehicle positions. With this calibration sample set, the device computes a model of the local magnetic distortions, provoked by vehicle structure or magnetic sources. Afterwards, in acquisition mode, the compass compensates for the local contribution in order to obtain a more accurated reading. Since in most robot applications the compass devices are mounted on board of the robot platform, which usually has a high presence of metals and electrical motors, a good calibration of such devices is important before using them. Finally, a map calibration is also required to find the angular offset of the geographic North with respect to the map axes. Figure C.2 of appendix C shows a way to compute this offset.

3.2.5 Laser Scanner

Laser scanners are widely used devices in mobile robotics since they provide directly a set of N_L ranges from the central point of the device to the closest obstacles of the environment, sweeping in a plane an angular aperture of $\Delta\alpha$. In most of the cases the angular step between two consecutive ranges remains constant, thus it is $\delta\alpha = \Delta\alpha / N_L$. Figure 3.4 shows the basic geometry of such devices.

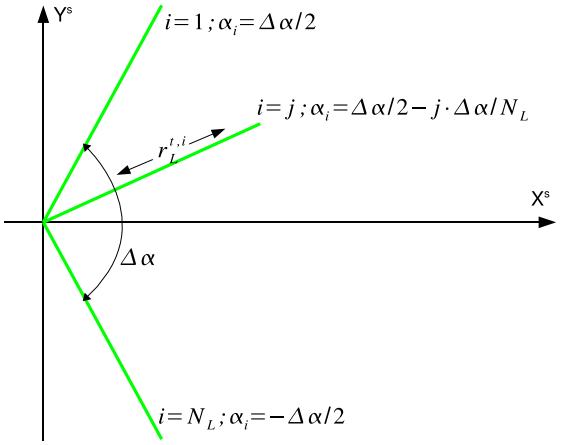


Figure 3.4: Laser scanner rays in the sensor coordinate frame.

By processing laser scanner data a wide variety of observations, such as raw range points, corners, segments, arcs or some other predefined pattern, can be dervied. In this thesis, range data has been directly used after applying range limit checking operations. Therefore, the range observation data vector provided by a laser scanner is:

$$o_L^t = \{r_{L,j}^t\} \mid r_{L,j}^t \in [R_L^{min}, R_L^{max}], \forall j = 1 \ldots N_L \tag{3.7}$$

Main characteristics of such laser scanner observations are:

- Medium acquisition rate (typically $5 - 20Hz$)
- Medium latency (typically $\sim 40ms$)
- Uncertainty expressed as an standard deviation of the range measurement, fixed for simple models, or as a function of the range for calibrated devices.
- Full availability, exceptuating for some devices, extreme situations as frontal incidence of sun light, dense fog, rain or dust clouds.

Chapter 4

Environment Models and Expected Observations

In the previous chapter, real observations have been introduced as the inputs to the localization module, coming from processing actual sensor readings or from other estimating processes. Moreover, a robot can be provided with some initial knowledge of the environment, described with some kind of spatial representation, also called environment model or map. This model is the source to compute two kind of data playing a key role for successful map-based localization: *model constraints* and *expected observations*. Model constraints are used to reduce the search space of all possible positions to those accomplishing some physical conditions. Expected observations are *synthetic* observations computed given a system state, an observation model and an environment model. Environment and observation models are important software modules in robotics and an accurate and fast processing of them is a key aspect for succesful filtering.

4.1 Basic Concepts and Notation

This chapter is divided into two main sections, 2D and 3D approaches, showing the improvement of the work through the research. The work reported on this thesis started using 2D representations [19] but, after some experience gained analyzing the research results, the focus of the work moves to better and more accurate 3D models [24]. The change has been done without compromising the computational efficiency, so while the 2D approach is based on a set of 'hand-made' geometric primitives, 3D approach uses a standard graphics programming library to compute the 3D model. Despite this big implementation difference, some common concepts and notation for both sections are introduced in the following paragraphes.

Environment models. In map-based localization approaches, the robot has some previous knowledge of the environment, represented by an environment model, also called spatial representation or map, $\mathcal{M}$. With such models, the localization process computes two kinds of useful data: *map constraints* and *expected observations*. The former improves the localization performance since it reduces the solution space to those positions that accomplish some physical constraints such as collision-free condition or gravity effects, while the later provides expected observations from a given position to be compared with real ones, in order to score that given position according to some likelihood function. Moreover, when the spatial representation incorporates metric data, as geometric models do, the map also defines a coordinate frame where this data is referenced. This map frame acts as the so called *world* frame and localization is expressed with respect to that model coordinate frame.

Points and positions. A given point p expressed with respect to map coordinate frame is denoted as Q_p^M, while a position (i.e. a point plus attitude angles) is written as X_p^M. In both cases the M superindex indicates that point or position are referenced to the map frame. Next sections detail point and position representations for 2D and 3D cases.

Physical constraints. Robot wheeled platforms running on a real environment have some constraints regarding their position caused by the presence of obstacles in the environment or by the effect of the gravity force. If the environment model allows the computation of such constraints, the search space of the localization problem can be dramatically reduced and, therefore, resulting in a better localization performance.

Observation models. An observation model is a software module that outputs a *synthetic* or *expected observation* given a robot state as input, as, for instance, the robot position. They are also called *sensor models* through the robotics literature. An expected observation computed from the position X_p^M is denoted through this thesis as $o_k^s(X_p^M)$. Subindex k, as used in chapter 3, identifies to which observation process the expected observation is associated, while s superindex indicates that these observations have a synthetic nature and that they are not time dependent as real observations were. For sensors interacting with the environment, as laser scanners do, the observation model has

to compute in an efficient way the environment model to obtain these expected observations. Next sections explain the approach used in this thesis to compute the expected laser scanner observation in 2D and 3D environment models.

4.2 2D Approach

The 2D approach represents the environment by means of 2D geometry, thus a point p referenced to the map frame, Q_p^M, is described with a pair of coordinates, $Q_p^M = (x_p^M, y_p^M)$. A position p incorporates an orientation, described as a heading angle from the x axis of the map frame, $\mathbf{x}^M$. Therefore, a position p is expressed as $X_p^M = (x_p^M, y_p^M, \theta_p^M)$.

4.2.1 2D environment model description

This subsection presents how spatial information is arranged to form a 2D environment model [19]. The representation is on the 2D plane, based on geometric entities and inspired from the Geographical Information Systems (GIS) vector format [59, 70]. However, height information of geometric entities is added to provide the map with some 3D information, thus allowing to model stairs and borders since they are common obstacles in urban pedestrian areas.

The map $\mathcal{M}$ is represented as a list of NB obstacles and two points limiting its borders, representing the left-up corner and the right-down one.

$$\mathcal{M} = \{b_1, \ldots, b_{NB}, Q_{lu}^M, Q_{rd}^M\} \tag{4.1}$$

The k^{th} obstacle of the map, b_k, is parameterized as a list of NS_k segments, an integer id_k assigned to identify the obstacle and an integer st_k describing the type of the shape representing the obstacle.

$$b_k = \{s_{k,1}, \ldots, s_{k,NS_k}, id_k, st_k\} \; \forall k = 1 \ldots NB \tag{4.2}$$

where $st_k = 1$ when obstacle is represented with a closed polygon, $st_k = 2$ for an opened polygon, and $st_k = 3$ for stairs. The l^{th} segment of the k^{th} obstacle, $s_{k,l}$, is defined from the point $Q_{a,k,l}^M$ to the point $Q_{b,k,l}^M$. A value $h_{k,l}$ is also associated to the segment respresenting its height:

$$s_{k,l} = \{Q_{a,k,l}^M, Q_{b,k,l}^M, h_{k,l}\} \forall k = 1 \ldots NB, \; \forall l = 1 \ldots NS_k \tag{4.3}$$

All segments are oriented, so they are defined from left to right when they are seen from a free space point of view. GIS format allows adding semantic information to each element, therefore a rich map for extended purposes can be built. Figure 4.1 shows the map representing the UPC campus area described following this model.

With the aim of speeding up the algorithms that process this model a *bounding ball* for each obstacle is computed, and its center and radius are stored with the obstacle information. Bounding ball is computed as the minimum circle bounding all the point coordinates of the segments of a given obstacle [36]. This allows fast discarding when computing geometric primitives with this model. Obstacle b_k has its associated bounding ball β_k, parameterized with a center, β_k^C, and radius, β_k^R.

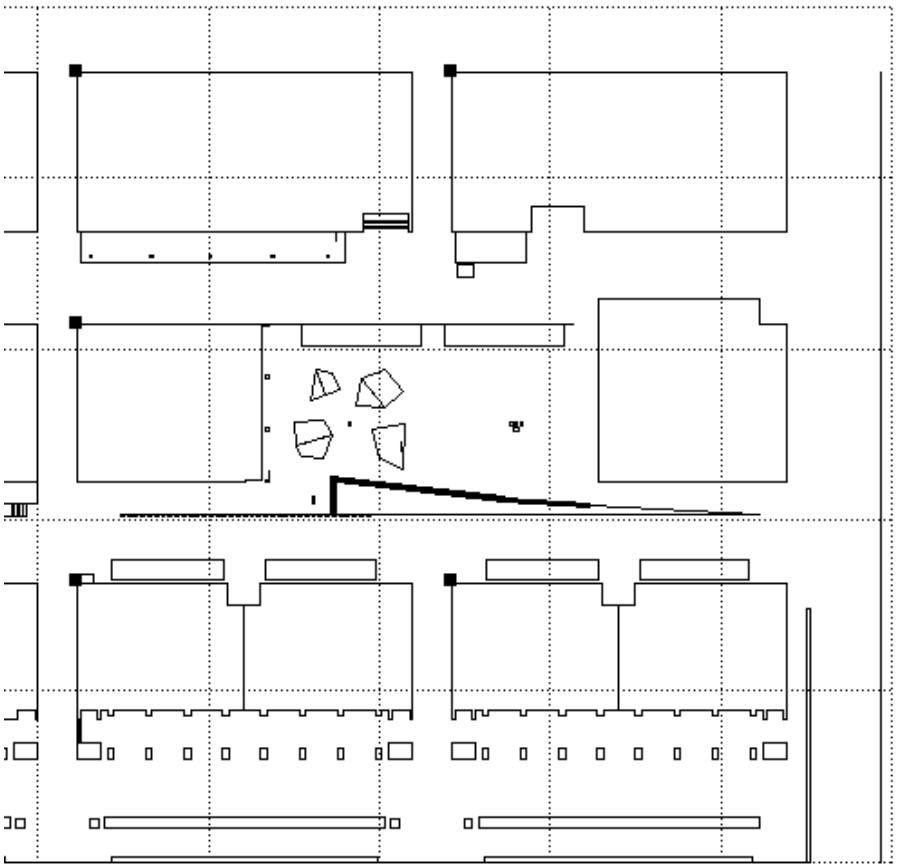


Figure 4.1: 2D environment model of the UPC campus area.

4.2.2 Collisions on 2D

To compute if a point collides with any obstacle of the model, two geometric primitives have been implemented. They are outlined in the following paragraphes.

Segment-segment interference. This primitive evaluates if two segments, s_i and s_j, intersect. The primitive returns a boolean value indicating intersection or not and, if it is the case, the interception point coordinates.

Segment-obstacle interference. Computes if a segment s_i intersect with the borders of a given obstacle b_k, by using the segment-segment interference primitive over the entire list of segments for the k^{th} obstacle. Returns a boolean value indicating collision or not and, if it is the case, the collision point. If the segment collides more than once with the obstacle, the collision point is set to that closer to the $Q^M_{a,i}$ point (i.e. the starting point of the segment s_i).

Point collisions on 2D model are computed using the segment-obstacle interference primitive. For a given point p, Q_p^M, the procedure starts drawing four segments starting at Q_p^M up to each of the four limiting corner points of the model. Then the segment-obstacle interference is evaluated for each of these four segments. If all four segments collide with the same obstacle, then it is assumed that the point lies inside an obstacle so the algorithm returns COLLISION. Algorithm 1 summarizes the procedure.

Algorithm 1 Collision constraint algorithm for 2D models

INPUT: $\mathcal{M}, Q_p^M$
OUTPUT: g_c

```
δ_min = MAXDISTANCE
s_1 = drawSegment(Q_p^M, Q_lu^M);
s_2 = drawSegment(Q_p^M, (x_rd, y_lu));
s_3 = drawSegment(Q_p^M, (x_lu, y_rd));
s_4 = drawSegment(Q_p^M, Q_rd^M);
for i = 1..4 do
  for j = 1..NB do
    Q_I^M=segmentObstacleInterference(s_i, b_j) //set the collision point
    if Q_I^M ≠ ∅ then
      δ = distance(Q_p^M, Q_I^M)
      if δ < δ_min then
        δ_min = δ
        c_i = j //holds the obstacle index
      end if
    end if
  end for
end for
if c_1 = c_2 = c_3 = c_4 then
  return COLLISION;
else
  return FREE;
end if
```

This procedure is an approximation. It assures that all positions inside an obstacle are identified as a non-free positions. However, in some cases, free positions very close to non-convex obstacles can be also identified as non-free. This issue has minor practical relevance on the context of this thesis.

4.2.3 Laser scanner observation model on 2D

The laser scanner observation model on 2D, is also based on computing the two other geometric primitives described in the paragraphs below.

Ray-obstacle interference. This primitive is similar to the segment-obstacle interference, described in subsection 4.2.2, but takes into account a height parameter for the ray and the heights of the segments forming the obstacle. The ray collision will ignore those collisions against segments with height less than

the ray height. If there is interference it returns directly the range from the start point of the ray to the collision point. Otherwise it returns the ray length.

Ray-map interference. Evaluates the intersection of a given ray over all the obstacle list building the model. In order to reduce the computational cost on processing the model, prior to call ray-obstacle interference for the obstacle b_k, this procedure checks two geometric features. First, it checks if the minimum bounding ball of the k^{th} obstacle,β_k, is close enough to the ray. If not, the entire obstacle is discarded. Else a second test is done, consisting on checking if the ray is oriented towards the k^{th} obstacle, by computing the angle between the ray itself and the segment built from the ray origin point to the β_k^C point. If the angle is out of $[-\frac{\pi}{2}, \frac{\pi}{2}]$, the k^{th} obstacle is discarded. Else the ray-obstacle interference is called. These two geometric checks improve the computational cost, specially for large environments where a significant amount of obstacles are far away from the ray or at the bottom of it.

Algorithm 2 summarizes the procedure to compute the ray-map interference on the 2D model, given a ray position X_s^M, a ray height, h_s, and a maximum range, r_{max}.

Algorithm 2 Algorithm for the ray-map interference on 2D

INPUT: $\mathcal{M},\ X_s^M = (x_s^M, y_s^M, \theta_s^M),\ h_s,\ r_{max}$
OUTPUT: r

```
for j = 1..N_B do
  δ² = squareDistance(X_s^M, β_k^C)
  if δ² > (r_max + β_k^R)² then
    return r_max
  else
    ray = drawSegment(X_s^M, r_max) //ray segment
    s1 = drawSegment(X_s^M, β_k^C) //from X_s^M to bounding ball center
    α = angle(ray, s1)
    if α ∉ [-π/2, π/2] then
      return r_max
    else
      r = rayObstacleInterference(b^j, ray, h_s)
      return r
    end if
  end if
end for
```

To compute an entire expected laser scanner, the ray-map interference primitive is called N_L times while incrementing the ray heading according to the sensor angular step. N_L is the number of rays of an entire scan.

4.3 3D Approach

The 3D approach models the environment using 3D geometry. A point p referenced to the map frame is a triplet of coordinates, $Q_p^M = (x_p^M, y_p^M, z_p^M)$. A position p in 3D is represented as a point accompanied with three orientation angles, $X_p^M = (x_p^M, y_p^M, z_p^M, \theta_p^M, \phi_p^M, \psi_p^M)$. The definition of the three angles with respect to the map frame follows the $Z - Y - X$ Euler convention and is detailed in the appendix B.

4.3.1 3D environment model description

Real urban pedestrian environments are three dimensional, so an accurate three dimensional model gives us a richer representation of the real world than the 2D models do and, therefore, 3D models provide, potentially, better synthetic observations and map constraints than 2D representations.

The three dimensional environment model used in this thesis is formed by two geometric maps, built manually, edited with a 3D model editor and stored as standard .obj files [79]. One model describes the entire environment, $\mathcal{M}$, while the other only models the floor surface, $\mathcal{M}_{floor}$, leaving holes where obstacles are encountered. The use of the *obj* standard file format allows the map manipulation through openGL [83], a widely used programming library to manipulate 3D models and create 3D scenes using the computer graphics hardware. Using openGL for model computations in vehicle localization has been suggested, but not detailed, by some authors [55, 78]. Thus, this section details how to efficiently use the openGL library to compute collisions, ground constraints and expected observations [24]. Figure 4.2 shows a view of the full model for the UPC campus area.

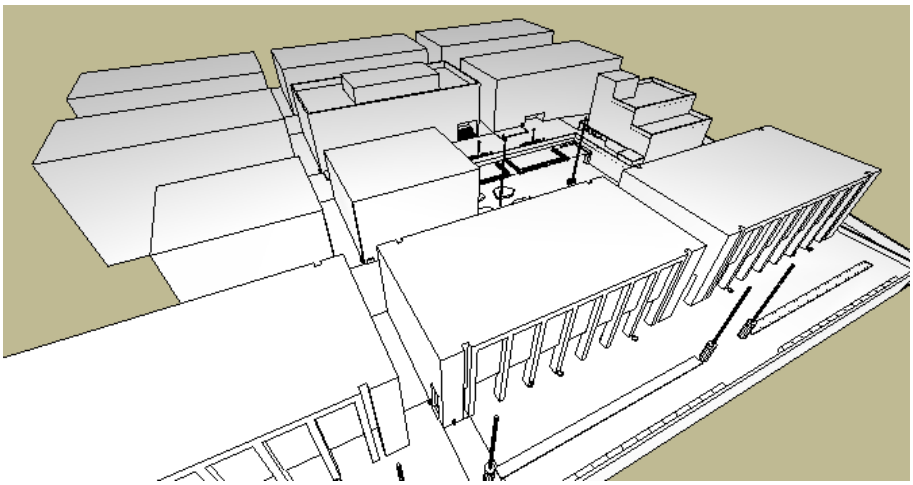


Figure 4.2: 3D environment model of the UPC campus area.

Figure 4.3 shows real pictures of this environment and their approximate corresponding view in the 3D model. The reader can observe in this figure how the 3D map only models the most static part of the environment as buildings and urban solid furniture, while vegetation, pedestrians and other light objects re-

main an unmodelled part of the environment, thus the used model is considered *incomplete*, a realistic and challenging assumption for map-based localization in urban pedestrian environments.

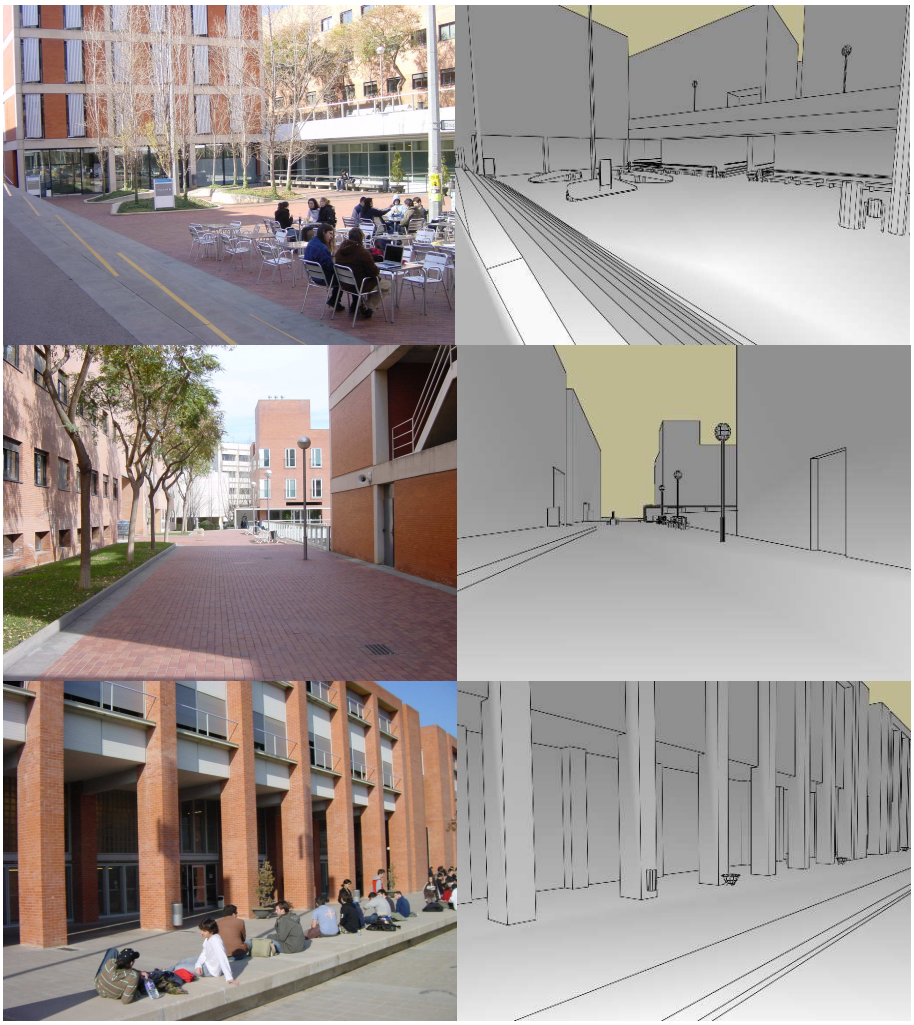


Figure 4.3: Pictures of the UPC campus environment and their approximate corresponding snapshots of the 3D map. The 3D map is static and only models the most significative parts of the environment.

4.3.2 Collisions in 3D

To compute collisions, only the floor part of the model is used, $\mathcal{M}_{floor}$, to detect if a (x^M, y^M) pair is expected to collide according to the model. To efficiently compute expected collisions, the openGL library is used for a fast manipulation of the 3D model. Algorithm 3 summarizes ths procedure to find out if a given map location (x^M, y^M) is expected to collide with the model. The key idea of algorithm 3 is to bound the rendering volume to a maximum depth of z_{max}

Algorithm 3 Collision constraint algorithm for 3D models

INPUT: $\mathcal{M}_{floor}, (x^M, y^M)$
OUTPUT: g_c

```
setWindowSize(5,5); //sets a rendering window of 5x5 pixels
setProjection(1°, 1, z_min, z_max) //bounds rendering volume
X_oh = (x^M, y^M, z^M_bird, 0, π/2, 0); //sets an overhead view point, pitch= π/2
renderUpdate(M_floor, X_c); //renders the model from X_oh
r = readZbuffer(3,3); //reads depth of central pixel
if r == z_max then
  return COLLISION; //a hole is present
else
  return FREE; //floor is present
end if
```

and an aperture view of Δ_c. Since the floor map has holes instead of obstacles, a renderization of the floor map from an overhead view point detects holes as pixels with depth equal to z_{max}.

4.3.3 Ground constraints in 3D

Due to gravity force effects, wheeled platforms run always on the floor. Moreover, for low-speed platforms, it can be assumed that the whole platform is a rigid body with all wheels in contact with the floor, and shock absorbers, if present, do not modify the pitch and roll of the vehicle. With this reasonable assumptions for low-speed platforms, three ground constraints are derived from the gravity action and the environment: height, pitch and roll. These constraints can be computed only when a 3D environment model is used. Table 4.1 summarizes these ground constraints, indicating to which kind of wheeled platform apply and the equation relating the constraint with other unconstrained robot position coordinates.

Table 4.1: Ground constraints of low-speed wheeled platforms

constraint	platform type	equation
height	all platforms	$z_r^M = g_z(x_r^M, y_r^M)$
pitch	Not on self-balancing platforms	$\phi_r^M = g_\phi(x_r^M, y_r^M, \theta_r^M)$
roll	all platforms	$\psi_r^M = g_\psi(x_r^M, y_r^M, \theta_r^M)$

The height constraint sets a height, z^M, given a coordinate pair (x^M, y^M). To compute it, only the floor part of the map is used. Algorithm 4 outlines the procedure. As for the collision constraint computation, the basic idea is to limit the rendering volume of the model in depth and aperture, and renderize the scene from an overhead view point and, then, read the depth data of the central pixel computed by the graphics card.

The pitch constraint fixes the pitch variable of the platform to a given coordinate triplet $(x_p^M, y_p^M, \theta_p^M)$. The algorithm to compute the pitch constraint is outlined in algorithm 5. It employs the previous height constraint algorithm to compute the floor model's difference in height between the platform's frontmost

Algorithm 4 Height constraint algorithm for 3D models

INPUT: $\mathcal{M}_{floor}, (x^M, y^M)$
OUTPUT: g_z

$setWindowSize(5,5)$; //sets rendering window size to 5x5 pixels
$setProjection(1^\circ, 1, z_{min}, z_{max})$ //1° of aperture, aspect ratio, depth limits
$X_{oh} = (x^M, y^M, z^M_{oh}, 0, \pi/2, 0)$; //sets an overhead view point, pitch= $\pi/2$
$renderUpdate(\mathcal{M}_{floor}, X_{oh})$; //renders the model from X_{oh}
$r = readZbuffer(3,3)$; //reads depth of central pixel
$g_z = z^M_{oh} - r$;
return g_z;

and backmost points, g_{zf} and g_{zb}. The pitch angle is then computed using trivial trigonometry, while A is the distance between the above mentioned platform points. The roll constraint can be found in a similar way but computing the

Algorithm 5 Pitch constraint algorithm for 3D models

INPUT: $\mathcal{M}_{floor}, A, (x^M_p, y^M_p, \theta^M_p)$
OUTPUT: g_ϕ

$x^M_f = x^M_p + \frac{A}{2}\cos\theta^M_p$; //compute the platform's frontmost point
$y^M_f = y^M_p + \frac{A}{2}\sin\theta^M_p$; //likewise
$g_{zf} = heightConstraint(x^M_f, y^M_f)$; //compute height at frontmost point
$x^M_b = x^M_p - \frac{A}{2}\cos\theta^M_p$; //compute the platform's backmost point
$y^M_b = y^M_p - \frac{A}{2}\sin\theta^M_p$; //likewise
$g_{zb} = heightConstraint(x^M_b, y^M_b)$; //compute height at backmost point
$g_\phi = atan2(g_{zf} - g_{zb}, A)$;
return g_ϕ;

height constraint for the leftmost and rightmost platform points.

4.3.4 Floor map discretization

The constraints presented in the previous subsections are computed massively in the filtering loop of map-based localization. To speed up online computations during real-time executions, a height grid is computed offline for the floor map. Thus, G_{height} is a grid containing the height value z^M for pairs (x^M, y^M), thus being a discrete representation of the height constraint with a discretization step γ:

$$G_{height}(i,j) = g_z(x^M_p, y^M_p) \quad | \quad i = (int)\frac{x^M_p - x^M_0}{\gamma},\ j = (int)\frac{y^M_p - y^M_0}{\gamma}, \quad (4.4)$$

where x^M_0 and y^M_0 are the map origin xy coordinates. The grid is computed offline by means of the height constraint (algorithm 4) along the grid points, and it is stored in a file, to be load for online executions. Figure 4.4 shows the height grid for the UPC campus environment.

Computing pitch and roll constraints requires several z^M computations, so the height grid speeds up these procedures as well. However, to avoid discretization problems, specially when computing pitch and roll constraints using

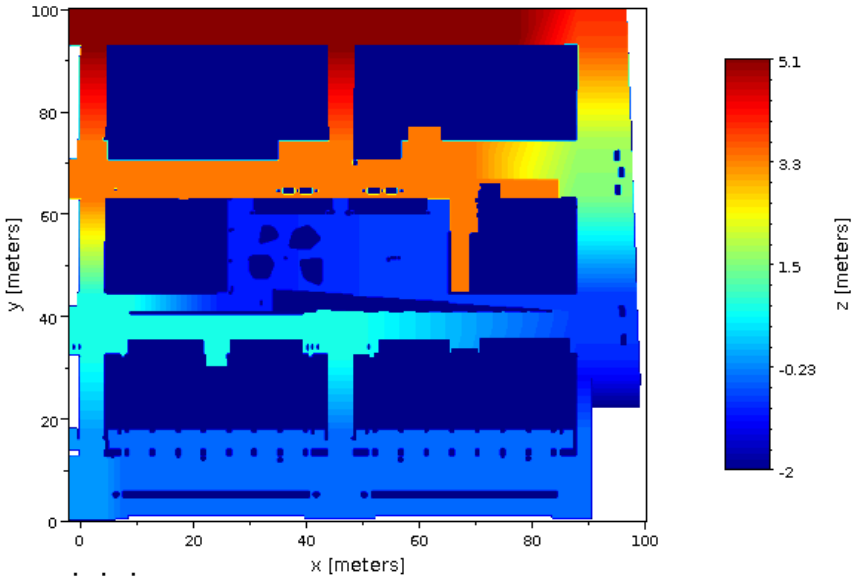


Figure 4.4: Floor map of the UPC campus environment.

G_{height}, we use lineal interpolation on the grid. Algorithm 6 summarizes how the height constraint is computed using the grid.

Algorithm 6 Height constraint using the grid

INPUT: $G_{height}, (x_p^M, y_p^M)$
OUTPUT: g_z
$i = (int)\frac{x_p^M - x_0^M}{\gamma}$; $j = (int)\frac{y_p^M - y_0^M}{\gamma}$
$z_1 = G_{height}(i, j)$;
$(i_2, j_2) = nearestDiagonalGridIndex()$; $//i_2 = i \pm 1$; $j_2 = j \pm 1$;
$z_2 = G_{height}(i_2, j)$; //height of a neighbour cell
$z_3 = G_{height}(i, j_2)$; //height of a neighbour cell
$z_4 = G_{height}(i_2, j_2)$; //height of a neighbour cell
$g_z = interpolation(z_1, z_2, z_3, z_4, x_p^M, y_p^M)$;
return g_z;

4.3.5 Laser scanner observation model in 3D

This subsection outlines how, from a given 3D position in the environment, expected 2D range scans or expected 3D point clouds are computed. A common problem in either cases is the computation of range data given the sensor position and a set of sensor parameters like angular aperture, number of points and angular accuracy. To compute these observation models, openGL has been used to take advantage of graphics card acceleration. The approach specially focuses

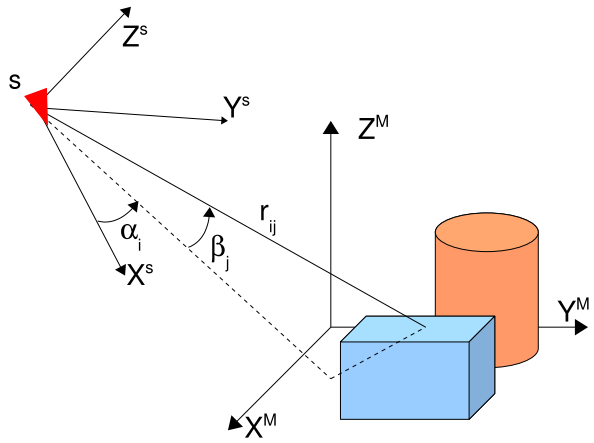


Figure 4.5: Model frame (X^M, Y^M, Z^M) and sensor frame (X^S, Y^S, Z^S), ray angles α_i and β_j, and the computed output ranges r_{ij}.

on minimizing the computation time without violating sensor's accuracy and resolution. This minimization is achieved by reducing the rendering window size and the renderig volume as much as possible, while keeping the sensor's accuracy.

The goal of a range observation model is to find a matrix $\mathbf{R}$ of ranges for a given sensor position X_s^M. Each element r_{ij} of the matrix $\mathbf{R}$ is the range from the X_s^M position up to the first obstacle of the model, following the ray given by angles α_i and β_j. Figure 4.5 shows these variables as well as coordinate frames for the map, $(\mathbf{XYZ})^M$, and for the sensor, $(\mathbf{XYZ})^s$.

The range observation model has the following inputs:

- A 3D geometric model, $\mathcal{M}$.
- A set of sensor parameters: horizontal and vertical angular apertures, Δ_α and Δ_β, horizontal and vertical angular accuracies, δ_α and δ_β, the size of the range matrix, $n_\alpha \times n_\beta$, and range limits, r_{min}, r_{max}.
- A sensor position, $X_s^M = (x_s^M, y_s^M, z_s^M, \theta_s^M, \phi_s^M, \psi_s^M)$.

The operations to execute in order to compute ranges r_{ij} are:

1. Set the projection to view the scene.
2. Set the rendering window size.
3. Render the scene from X_s^M.
4. Read the depth buffer of the graphics card and compute ranges r_{ij}.

Set the Projection. Before using the openGL rendering, the projection parameters have to be set. These parameters are the vertical aperture of the scene view, which is directly the vertical aperture of the modelled sensor, Δ_β, an image aspect ratio ρ, and two parameters limiting the viewing volume with two

planes placed at z_N (near plane) and z_F (far plane)[1]. These two last parameters coincide respectively with r_{min} and r_{max} of the modelled sensor. The only parameter to be computed at this step is the aspect ratio ρ. To do so, first the metric dimensions of the rendering area, width, w, and height, h, have to be found. The aspect ratio will be derived from them:

$$w = 2r_{min}tg(\frac{\Delta_\alpha}{2}); h = 2r_{min}tg(\frac{\Delta_\beta}{2});\ \rho = \frac{w}{h}; \tag{4.5}$$

Figure 4.6 depicts the horizontal cross section of the projection with the associated parameters. The vertical cross section is analogous to the horizontal one.

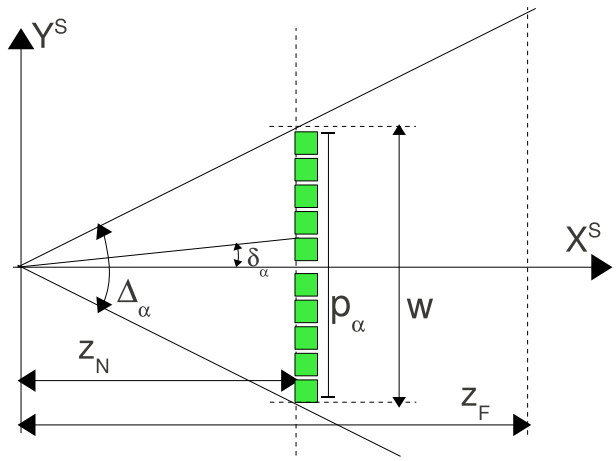


Figure 4.6: Horizontal cross section of the projection with the involved parameters. Green squares represent the pixels at the image plane.

Set the rendering window size. Before rendering a 3D scene, the size of the image has to be set. Choosing this size as small as possible is the key issue to speed up the proposed algorithm. Since range sensors have a limited angular accuracy, we use that to limit the size of the rendering window, in order to avoid rendering more pixels than those required. Given a sensor with angular accuracies δ_α and δ_β, pixel dimensions of the rendering window are set to:

$$p_\alpha = (int)2\frac{tg(\Delta_\alpha/2)}{tg(\delta_\alpha)};\ p_\beta = (int)2\frac{tg(\Delta_\beta/2)}{tg(\delta_\beta)}; \tag{4.6}$$

Figure 4.6 shows an horizontal cross section of the projection and the related variables to compute the horizontal pixel size of the rendering window (the vertical pixel size is found analogously). Equation 4.6 presents a singulariy when $\Delta_{\alpha,\beta} = \pi$. In practical situations, this singularity forces to sectorize the computation of wide aperture sensors.

[1] z_N and z_F are openGL depth values defined in the screen space.

Render the scene. The scene is renderized from the view point situated at sensor position X_s^M. Beyond computing the color for each pixel of the rendering window, the graphics card also associates to each one a depth value. Moreover, graphics cards are optimized to discard parts of the model escaping from the scene, thus having limited the rendering window size and volume speeds up this step. Renderization can be executed in a hidden window.

Read the depth buffer. Depth values of each pixel are stored in the *depth buffer* of the graphics card. They can be read by means of an openGL function that returns data in a matrix $\mathbf{B}$ of size $p_\alpha \times p_\beta$ which is greater in size than the desired matrix $\mathbf{R}$. Moreover, depth values of matrix $\mathbf{B}$, b_{kl}, are normalized values of the renderized depth for each pixel. To obtain the desired ranges the procedure computes first the $\mathbf{D}$ matrix which holds the non-normalized depth values, d_{ij}, following the $\mathbf{X}^s$ direction:

$$\begin{aligned} k &= (int)(\frac{1}{2} - \frac{tg(\alpha_i)}{2tg(\frac{\Delta_\alpha}{2})})p_\alpha; \quad l = (int)(\frac{1}{2} - \frac{tg(\beta_j)}{2tg(\frac{\Delta_\beta}{2})})p_\beta \\ d_{ij} &= \frac{r_{min}r_{max}}{(r_{max} - b_{kl})(r_{max} - r_{min})}; \end{aligned} \tag{4.7}$$

The last equation undoes the normalization computed by the graphics card to store the depth values. The $\mathbf{D}$ matrix has $n_\alpha \times n_\beta$ size, since we compute d_{ij} only for the pixels of interest. Finally, with basic trigonometry we can calculate the desired r_{ij} as:

$$r_{ij} = \frac{d_{ij}}{cos(\alpha_i)cos(\beta_j)} \tag{4.8}$$

Figure 4.7 shows the variables involved on this last step, showing the meaning of the d_{ij} and r_{ij} distances in an horizontal cross section of the scene. Equation 4.8 presents numerical problems when α_i or β_j get close to $\pi/2$. This will limit the aperture of our sensor model. This limitation coincides with that already commented for equation 4.6 and has been overcome dividing the aperture in two sectors.

The overall procedure to compute expected range observations in 3D environments is outlined in algorithm 7. Inside the `for` loops, variables k and l are directly functions of i and j respectively, so expressions in equation 4.7 for k and l can be precomputed and stored in a vector.

Appendix D details the quantitative values of the range observation models for two real laser scanner devices widely used through the work. Video 3 outlined in appendix E shows the output of this range observation model for a laser scanner and for a time of flight camera.

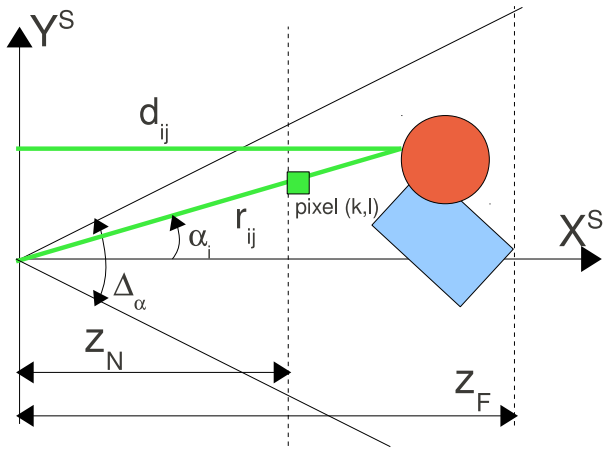


Figure 4.7: Horizontal cross section of the projection, with distance d_{ij} and range r_{ij} of the corresponding kl^{th} pixel.

Algorithm 7 Range Observation Model

INPUT: $\mathcal{M}, \Delta_\alpha, \Delta_\beta, \delta_\alpha, \delta_\beta, n_\alpha, n_\beta, r_{max}, r_{min}, X_s^M$
OUTPUT: $\mathbf{R}$

$w = 2r_{min}tg(\frac{\Delta_\alpha}{2})$; $h = 2r_{min}tg(\frac{\Delta_\beta}{2})$; $\rho = \frac{w}{h}$;
$glSetProjection(\Delta_\beta, \rho, r_{min}, r_{max})$;//sets the rendering volume
$p_\alpha = (int)2\frac{tg(\Delta_\alpha/2)}{tg(\delta_\alpha)}$; $p_\beta = (int)2\frac{tg(\Delta_\beta/2)}{tg(\delta_\beta)}$;
$glSetWindowSize(0, 0, p_\alpha, p_\beta)$;
$glRenderUpdate(\mathcal{M}, X_s^M)$; //renders the model from the sensor position
$\mathbf{B} = glReadZbuffer(ALL\_IMAGE)$; //reads normalized depth values
for $i = 1..n_\alpha$ **do**
  $\alpha_i = \Delta_\alpha(0.5 - \frac{i}{n_\alpha})$;
  $k = (int)(0.5 - \frac{tg(\alpha_i)}{2tg(\Delta_\alpha/2)})p_\alpha$;
  for $j = 1..n_\beta$ **do**
    $\beta_j = \Delta_\beta(0.5 - \frac{j}{n_\beta})$;
    $l = (int)(0.5 - \frac{tg(\beta_j)}{2tg(\Delta_\beta/2)})p_\beta$;
    $d_{ij} = \frac{r_{min}r_{max}}{(r_{max}-b_{kl})(r_{max}-r_{min})}$;
    $r_{ij} = \frac{d_{ij}}{cos(\alpha_i)cos(\phi_j)}$;
  end for
end for
return $\mathbf{R}$;

Chapter 5

Particle Filter Overview

A particle filter is a recursive algorithm that estimates a probability distribution of the state of a system given a set of observations and, optionally, a predictive model of the state of that system, as, for instance, the kinematic and/or dynamic models. The representation of this probability distribution is made by estimating the associated probability density function with a set of samples as points in the state space, each one having an associated weight quantifying the likelihood between real observations and the expected ones computed at the sample state point. The pair formed by a sample point and its weight is called a *particle*.

5.1 Bayesian Filtering

The bayesian filter for mobile robot localization provides the theoretical framework for particle filter localization [26, 110]. Let X^t be the state of the system, expressing the position of the robot at iteration t with respect to some reference coordinate frame, and let $\Omega_\forall^t$ be the set of all observations available at iteration t in the computer unit that executes the estimation process:

$$\Omega_\forall^t = \{O_k^t \ \forall k | s_k^t = READY\} \tag{5.1}$$

Then, the belief of the system state at iteration t, defined as,

$$Bel(X^t) = p(X^t | \Omega_\forall^t \dots \Omega_\forall^0), \tag{5.2}$$

expresses the probability density function of the state of the system, given the observations from the beginning of the filter execution up to the current iteration. The $\Omega_\forall^t$ set can be expressed separating the odometry observation and the rest:

$$\Omega_\forall^t = \{\Omega^t, O_U^t\} \tag{5.3}$$

leading to the following expression of the belief:

$$Bel(X^t) = p(X^t | \Omega^t, O_U^t, \Omega_\forall^{t-1} \dots \Omega_\forall^0) \tag{5.4}$$

Applying the Bayes rule to the last equation we obtain:

$$Bel(X^t) = \frac{p(\Omega^t | X^t, O_U^t, \Omega_\forall^{t-1} \dots \Omega_\forall^0) \ p(X^t | O_U^t, \Omega_\forall^{t-1} \dots \Omega_\forall^0)}{p(\Omega^t | O_U^t, \Omega_\forall^{t-1} \dots \Omega_\forall^0)} \tag{5.5}$$

The denominator term can be interpreted as a normalization constant $1/\eta$ since it does not depend on X^t. For the numerator term, we will apply the Markov assumption of the system evolution, implying that the current observations depend only on the current state:

$$Bel(X^t) = \eta \ p(\Omega^t | X^t) \ p(X^t | O_U^t, \Omega_\forall^{t-1} \dots \Omega_\forall^0) \tag{5.6}$$

The last term can be computed as an integral, leading to the following expression of the belief:

$$\begin{aligned} Bel(X^t) &= \eta \ p(\Omega^t | X^t) \ \int p(X^t | O_U^t, X^{t-1}) p(X^{t-1} | \Omega_\forall^{t-1} \dots \Omega_\forall^0) \ dX^{t-1} = \\ &= \eta \ p(\Omega^t | X^t) \ \int p(X^t | O_U^t, X^{t-1}) Bel(X^{t-1}) \ dX^{t-1} \end{aligned} \tag{5.7}$$

The interpretation of the equation 5.7 is that the belief of the system at a given iteration can be computed iteratively as:

$$Bel(X^t) = \eta \ \cdot p(\Omega^t | X^t) \cdot \ Bel(X^{t-}), \tag{5.8}$$

where $Bel(X^{t-})$ is the estimated probability density function representing the *prior* distribution, $p(\Omega^t | X^t)$ represents a likelihood of the observation set given the system state, and $Bel(X^t)$ is the target probability density function to be estimated, associated to the *posterior* distribution.

The prior distribution, also called *proposal*, is expressed as:

$$Bel(X^{t^-}) = \int p(X^t|O_U^t, X^{t-1}) Bel(X^{t-1})\ dX^{t-1} \tag{5.9}$$

where the expression $p(X^t|O_U^t, X^{t-1})$ refers to a probabilistc propagation model of the system state.

The likelihood $p(\Omega^t|X^t)$ in equation 5.8, holding the conditional probability of the observation set Ω^t given the robot state X^t, is approximated by means of a similarity function between real observations (chapter 3) and the expected ones computed by observation models at the given state point (chapter 4):

$$p(\Omega^t|X^t) \sim \mathcal{L}(\Omega^t, \Omega^s(X^t)) \tag{5.10}$$

Using equation 5.10 the posterior distribution at iteration t can be represented by:

$$Bel(X^t) \sim \eta\ \cdot \mathcal{L}(\Omega^t, \Omega^s(X^t)) \cdot\ Bel(X^{t-}), \tag{5.11}$$

This last equation summarizes the iterative principle of the particle filter introduced in the next section.

5.2 Particle Filter Algorithm

The implemented solution for the particle filter is the *Sampling Importance Resampling* (SIR) method, also called *Bootstrap* filter [28]. This implementation aproximates the $Bel(X^t)$ density function with a set of particles, while each particle is a pair formed by a sample in the state space, X_i^t, and an associated weight, w_i^t:

$$Bel(X^t) \sim P^t = \{(X_i^t, w_i^t)\}, \quad \forall i = 1 \dots N_P \tag{5.12}$$

The algorithm starts with an **initialization**, drawing a particle set labelled as P^0. If some prior knowledge about the system state is available, denoted as $\mathcal{K}^0$, the particles are generated satisfying that knowledge. Otherwise, the particles are generated sampling uniformly the whole state space.

Given the initial particle set, the recursive part of the algorithm starts with a **propagation** step, also called prediction or sampling, where a new particle set, P^{1-} is generated from the initial one, using a sampling model of the robot's kinematics, $f()$, and the current odometry reading, O_U^1 [109]. For the general case of the t^{th} iteration, at propagation step the state of the i^{th} particle is updated but particle weights are not modified:

$$\begin{aligned} X_i^{t-} &= f(X_i^{t-1}, O_U^t), \quad \forall i = 1 \dots N_P \\ w_i^{t-} &= \frac{1}{N_P}, \quad \forall i = 1 \dots N_P \end{aligned} \tag{5.13}$$

This first step outputs an approximation of the prior belief: $P^{t-} \sim Bel(X^{t-})$.

A second step, called importance updating or **correction**, reshapes the prior density function by updating the weights of each particle but keeps the particle states. Each weight w_i^t is updated according a likelihood function computed with the available observations, Ω^t, given that the system is in the state X_i^t:

$$\begin{aligned} X_i^t &= X_i^{t-}, \quad \forall i = 1 \dots N_P \\ w_i^t &= \mathcal{L}(\Omega^t, \Omega(X_i^t)), \quad \forall i = 1 \dots N_P \end{aligned} \tag{5.14}$$

A third step is a **normalization** of the particle weights, forcing the sum of all of them to be 1:

$$w_i^t \leftarrow \frac{w_i^t}{\sum_{j=1}^{N_P} w_j^t} \tag{5.15}$$

At this point, the particle set approximates the target posterior distribution:

$$P^t \sim Bel(X^t) \tag{5.16}$$

After correction and normalization, it can be useful, for system integration purposes or evaluation, to compute a parametrization of P^t set, extracting, for instance, a mean and a covariance matrix from the particle set, $\hat{X}^t$ and $\hat{C}^t$ respectively.

Finally, the **resampling** is the last step in which a new particle set is drawn keeping the mean of the old one, but reseting all particle weights to $1/N_P$:

$$P^t \leftarrow resampling(P^t) \tag{5.17}$$

Figure 5.1 shows the three iterative steps of a single iteration of the particle filter and how the particle set is labelled after each step.

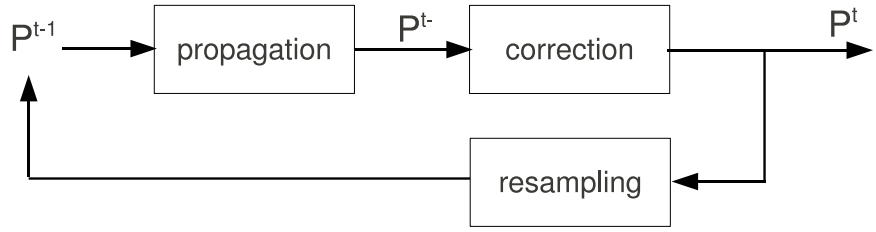


Figure 5.1: Iteration of the particle filter. P^t is the output particle set that estimates the posterior distribution of the system state.

Algorithm 8 summarizes the presented basic particle filter, also called *Sampling Importance Resampling* or *Bootstrap* filter [28].

Algorithm 8 General particle filter algorithm

INPUT: $\mathcal{K}^0, \Omega_\forall^t$
OUTPUT: P^t

```
t = 0 //iteration counter
P^0 = initialization(K^0)   //initialization
while running do
  t ← t + 1
  for i = 1 ... N_P do
    X_i^t = f(X_i^{t-1}, O_U^t)   //propagation
  end for
  for i = 1 ... N_P do
    w_i^t = p(Ω^t, X_i^t)   //correction
  end for
  for i = 1 ... N_P do
    w_i^t ← w_i^t / Σ_{j=1}^{N_P} w_j^t   //normalization
  end for
  P^t ← resampling(P^t)   //resampling
end while
```

5.2.1 Resampling Algorithm

The goal of the resampling algorithm is to draw a new particle set keeping the mean of the old one, but formed by new particles with their weights reset to $1/N_P$. The resampling step is necessary to avoid *particle depletion* [28, 5], an undesired phenomenon of particle filters where the particle set collapses to a single state point provoking the filter being no longer capable to explore new solutions for the estimation, therefore compromising the robustness of the solution. Several methods are proposed for resampling step in particle filters [28, 5, 109]. This section overviews the used method, the regularized resampling, and also outlines an alternative method as a suggestion that could improve results in future works.

Regularized Resampling

The implemented resampling solution through the different filters presented at this thesis is the regularized resampling method [5]. Being P^t the current particle set after correction and normalization, the resampling begins creating a sorted version of that set, labelled as, $\tilde{P}^t$, where the order is stablished by the particle weights, starting from the most important particle:

$$\tilde{P}^t = \{\ \{(X_1^t, w_1^t), \ldots, (X_{N_P}^t, w_{N_P}^t)\} | w_i^t \geq w_j^t \Leftrightarrow i < j, \forall (X_i^t, w_i^t) \in P^t\} \quad (5.18)$$

Once the $\tilde{P}^t$ set is built, a cumulative weight for each particle in $\tilde{P}^t$ set is assigned as:

$$cw_i^t = \sum_{j=1}^{i} w_j^t \quad (5.19)$$

The computation of these cumulative weights provides a cumulative distribution function of the particle weights over the index i, as it is shown in the figure 5.2.

With such a function, the resampling process draws a random number uniformly distributed, $r \in [0, 1]$, and the selected particle to be resampled is:

$$X_R^t = X_j^t \in \tilde{P}^t | cw_{j-1}^t < r \leq cw_j^t, r \sim rand(0, 1) \tag{5.20}$$

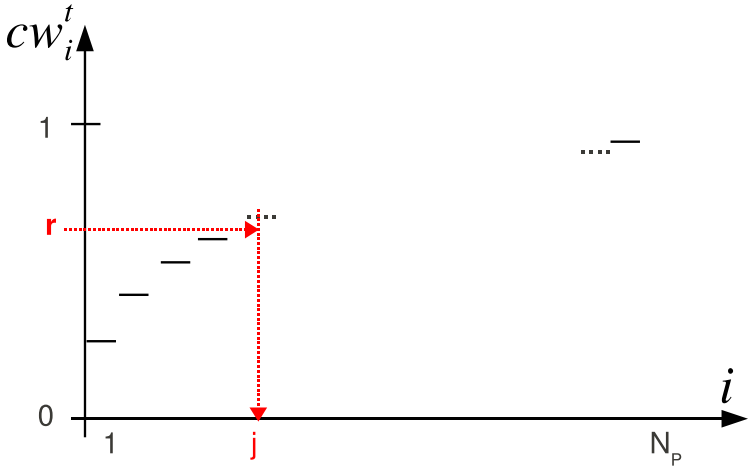


Figure 5.2: Cumulative weight as a function of the particle index of the sorted set (coincides with cumulative distribution of particle weights). r is a random number $\in [0, 1]$ used to select the j^{th} particle to resample.

To avoid the particle depletion phenomenon, a *resampling noise*, zero-mean and gaussian distributed, is added when drawing the new particle X_i^t from the selected one X_R^t. This gaussian noise should be small with respect the uncertainty introduced by the probabilistic kinematic model used at the propagation step. Resampling noise is parameterized with a covariance matrix C_γ. Algorithm 9 summarizes the regularized resampling algorithm.

Algorithm 9 Regularized Resampling

INPUT: P^t, C_γ
OUTPUT: P^t //resampled version of the posterior
  $\tilde{P}^t \leftarrow sortByWeight(P^t)$ //Assures $w_i^t \geq w_j^t \rightarrow i < j$
  for $i = 1 \ldots N_P$ **do**
    $cw_i^t = \sum_{j=1}^{i} w_j^t$
  end for
  for $i = 1 \ldots N_P$ **do**
    $r = rand(0, 1)$ //random number uniformly distributed
    $X_R^t = X_j^t \in \tilde{P}^t$, $j = min\ k\ | cw_k^t > r$ //select particle for resampling
    $X_i^t \sim \mathcal{N}(X_R^t, C_\gamma)$ //draw new particle with mean X_R^t and covariance C_γ
  end for
  $P^t = \{X_i^t\},\ i = 1 \ldots N_P$

Low Variance Resampling

Low variance resampling is an alternative method for resampling, where only a random number, $r \in [0, \frac{1}{N_P}]$, is drawn to choose the first particle to be resampled [109]. From that random value, the rest of the particles will be choosen adding $1/N_P$ to that initial value in the way indicated by algorithm 10. This method is quite more efficient in terms of complexity because of it does not need to sort the particle set and it only uses a single random number.

Algorithm 10 Low Variance Resampling

INPUT: P^t, C_γ
OUTPUT: P^t //resampled version of the posterior
for $i = 1 \dots N_P$ **do**
$cw_i^t = \sum_{j=1}^{i} w_j^t$
end for
$r = rand(0, \frac{1}{N_P})$ //random number uniformly distributed
for $i = 1 \dots N_P$ **do**
$X_R^t = X_j^t \in P^t \ , \ j = min \ k \ | cw_k^t > r$ //select particle for resampling
$X_i^t \sim \mathcal{N}(X_R^t, C_\gamma)$ //draw new particle with mean X_R^t and covariance C_γ
$r = r + \frac{1}{N_P}$
end for
$P^t = \{X_i^t\}, \ i = 1 \dots N_P$

Chapter 6

Position Tracking

A mobile robot needs to track its position within the navigation coordinate frame in order to close the navigation loop and to decide next driving actions. In map-based navigation the navigation coordinate frame is the map coordinate frame and position tracking is the process in charge of estimate the position of the robot, both location and orientation, by means of the on board sensors and other remote observations available at the computer executing the process.

This chapter describes two approaches, both based on the particle filter algorithm summarized in the previous chapter. The first approach considers that the robot navigates on a 2D world, so that the environment model lies on the 2D plane and the state space is that of positions on that plane. The later approach considers a more realistic situation where the robot moves in 3D environments, so the environment model is 3D and the state space is that of positions in 3D. In both cases extensive field experiments are reported and some improvements on the basic version of the filter are discussed.

6.1 Goal and Requirements

The goal of the position tracking process is to output, periodically, an estimate of the robot position in terms of the map coordinate frame. This estimate comprises a time stamp, a point in the state space and a parameterization of the associated uncertainty. The three main items that can measure the quality of the position tracking are accuracy, robustness and output rate.

Accuracy stands for the precision in which the process gives the localization estimate. To measure it, real or simulated experiments require localization ground truth data in order to compute the error as the euclidean distance between the estimate and the ground truth. With this localization error, the tracking process can be evaluated in absolute terms.

Robustness refers to the capacity of the filter to not loose the target, recovering situations where accuracy error has grown up for any reason. Given an estimation of the position and its associated uncertainty, a measure that indicates robustness of position tracking can be the time ratio at which position error is inside a given bound of that estimated uncertainty. This measure also requires ground truth data to obtain position error.

Output Rate is the frequency at which the filter computes the position estimate. The localization output rate depends on the computational cost of the filter iteration. Localization rate constraints the vehicle speed in autonomous navigation since localization output is required to update the current local goal to send to the reactive loop (see section 1.2).

6.2 2D Position Tracking

This section overviews a basic implementation of the particle filter for mobile robot position tracking on a two dimensional world. The estimated state of the robot is a vector in the continuous space of positions referenced to the map coordinate frame. Let be $X_r^\tau = (x_r^\tau, y_r^\tau, \theta_r^\tau)$ the robot *true* state at time τ, where (x_r^τ, y_r^τ) refers to the location coordinates of the robot and θ_r^τ to its heading. This true state remains always unknown and is the target state vector to estimate. The output of the t^{th} iteration of the position tracking process is the position estimate $\hat{X}_r^t = (\hat{x}_r^t, \hat{y}_r^t, \hat{\theta}_r^\tau)$, the estimated covariance matrix $\hat{C}_r^t$ and the time stamp of these estimates τ^t. Please note that τ refers to continuous time and t indicates an iteration index. Estimated and true robot positions are always referenced to the map coordinate frame, so the superindex ^{M} has been ommited to simplify notation.

6.2.1 Basic Particle Filter

Being $\mathcal{X}$ the state space of the continuous positions on the plane but bounded to a working area limited by $(x_{min}, x_{max}), (y_{min}, y_{max})$,

$$X_r^\tau, \hat{X}_r^t \in \mathcal{X} = \{(x_{min}, x_{max}), (y_{min}, y_{max}), (-\pi, \pi]\} \tag{6.1}$$

As suggested by equations 5.2 and 5.12, the approximation made by the sample representation of the probability density function at iteration t can be written as:

$$Bel(X^t) \sim p(X_r^t | \Omega_\forall^t \dots \Omega_\forall^0) \;\sim\; P^t = \{(X_i^t, w_i^t)\},\ \forall i = 1..N_p \tag{6.2}$$

The above expression indicates that the probability density function is approximated with the set P^t formed by N_P particles. Each particle is a pair formed by a sample in the state space, $X_i^t = (x_, ^t y_i^t, \theta_i^t)$, and a weight, $0 \leq w_i^t \leq 1$.

The output estimate takes into account all observations from the start of the filter execution, $\{\Omega_\forall^0, \ldots, \Omega_\forall^t\}$. Conditional probabilities of a real observation O_k^t given the state X_i^t are approximated with a likelihood function as:

$$p(O_k^t|X_i^t) \sim \mathcal{L}_k(O_k^t, O_k^s(X_i^t)) \tag{6.3}$$

where $\mathcal{L}_k$ function is a likelihood between two observations: the one made by the k^{th} observation process, O_k^t, and the *expected* one, $O_k^s(X_i^t)$, resulting from computing the k^{th} observation model at particle position X_i^t (see chapter 4). Such likelihood function should return a value in $[0, 1]$, being close to one for similar observations and near to zero when observations mismatch.

Algorithm 11 summarizes an iteration of the implemented basic particle filter with the following main steps: (1) propagation, (2) correction, (3) normalization, (4) setting and publishing the position estimate and (5) resampling the particle set. Next paragraphs detail each of these steps.

Algorithm 11 Basic particle filter iteration for 2D position tracking

INPUT: P^{t-1}, Ω^t
OUTPUT: $P^t, (\hat{X}^t, \hat{C}^t, \tau^t)$

```
t ← t + 1
for i = 1..N_P do
    X_i^t = propagation(X_i^{t-1}, O_U^t, ε_ρ, ε_θ)
end for
for i = 1..N_P do
    w_i^t = ∏_{k=1}^{N_B} L_k(O_k^t, O_k^s(X_i^t)),  ∀k|s_k^t = READY  //correction
end for
W^t = Σ_{j=1}^{N_P} w_j^t
for i = 1..N_P do
    w_i^t ← w_i^t / W^t  //normalization
end for
(X̂^t, Ĉ^t, τ^t) =setEstimate(P^t)
publish(X̂^t, Ĉ^t, τ^t)
P^t ← resampling(P^t)
```

Propagation this first step moves particle states following a probabilistic kinematic model of the platform with the current odometric observation, $\Delta\rho_U^t, \Delta\theta_U^t$ (see section 3.2). For each particle ($i = 1 \ldots N_P$) the state propagation is achieved by the following equations:

$$\begin{aligned}
\tilde{\Delta}\rho_i^t &= \mathcal{N}(\Delta\rho_U^t, \sigma_\rho^t);\ \sigma_\rho^t = \epsilon_\rho \Delta\rho_U^t \\
\tilde{\Delta}\theta_i^t &= \mathcal{N}(\Delta\theta_U^t, \sigma_\theta^t);\ \sigma_\theta^t = \epsilon_\theta \Delta\theta_U^t \\
x_i^t &= x_i^{t-1} + \tilde{\Delta}\rho_i^t \cos(\theta_i^{t-1} + \frac{\tilde{\Delta}\theta_i^t}{2}) \\
y_i^t &= y_i^{t-1} + \tilde{\Delta}\rho_i^t \sin(\theta_i^{t-1} + \frac{\tilde{\Delta}\theta_i^t}{2}) \\
\theta_i^t &= \theta_i^{t-1} + \tilde{\Delta}\theta_i^t
\end{aligned} \tag{6.4}$$

where variables $\tilde{\Delta}\rho_i^t$ and $\tilde{\Delta}\rho_i^t$ ara intermediate variables that expresses the odometric values with an added noise, hence the function $\mathcal{N}(\mu, \sigma)$ is a call to a random generator with a normal distribution centered at the current odometric data with standard deviation proportional to odometry increments. As described in section 3.2, noise parameters σ_ρ^t and σ_θ^t are proportional to odometry readings, so that large readings cause more uncertainty. Therefore, the user parameters to be adjusted are ϵ_ρ and ϵ_θ. Figure 6.1 shows the behavior of this model for a simple robot trajectory. Probabilistic propagation causes a dispersion of the particle set, allowing to explore state space regions where likely solutions can be found according the vehicle kinematics.

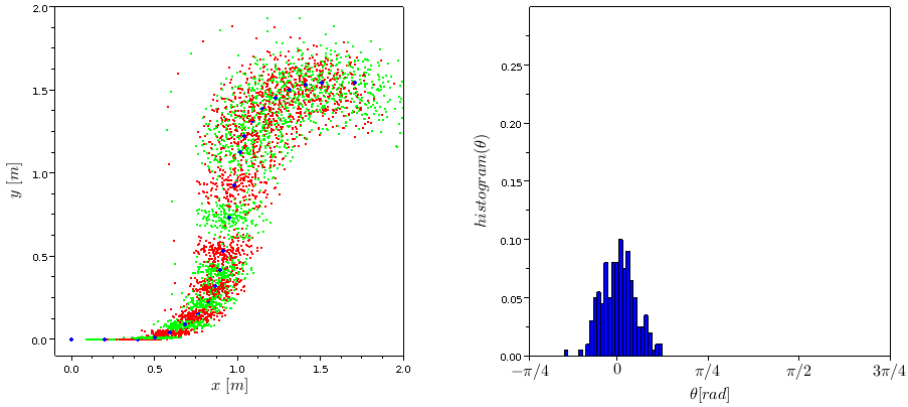


Figure 6.1: 2D probabilistic kinematic model. The robot moves straight ahead at translational speed of $1m/s$ but turns are done at translational speed of $0.5m/s$ and rotational speed of $0.2rad/s$ and $-0.2rad/s$ respectively. Left plot shows the sample distribution over the xy space, coloring alternatively each iteration in red and green. Right plot shows the probability density function over the θ component at the end of the trajectory. For this figure, $\epsilon_\rho = \epsilon_\theta = 0.2$.

Correction The correction loop integrates all available observations without taking into account time stamps. Assuming that conditional probabilities associated to each observation are independent between them, and using the approximation presented at equation 5.14 the correction step updates particle weights as follows:

$$w_i^t = \prod_{k=1}^{N_B} \mathcal{L}_k(O_k^t, O_k^s(X_i^t));\ \forall i = 1..N_P,\ \forall k | s_k^t = READY \tag{6.5}$$

The likelihood function between an actual laser scanner observation (see section 3.2) and an expected one (see section 4.2.3) has been implemented as the mean over the entire scan of the complementary error function, `erfc()`, of the difference between expected and actual ranges. Thus, given the range vectors of a real and an expected observations of a laser scanner, denoted respectively

as $\{r_j^t\}$ and $\{r_j^s\}$, $\forall j = 1 \dots N_L$, the likelihood function is:

$$\mathcal{L}_L(O_L^t, O_L^s(X_q^M)) = \sum_{j=1}^{N_L} \text{erfc}\left(\frac{|r_j^t - r_j^s(X_q^M)|}{\sigma_L\sqrt{2}}\right) \tag{6.6}$$

where σ_L is the standard deviation of the range observation. This likelihood function has the desired property that its value is limited to the $[0, 1]$ interval, evaluating to 1 for two identical scans and approaching 0 when the scans diverge, while the function depends only on the standard deviation associated to laser range measurements.

Normalization Next steps assumes that the sum of all weights will be 1, thus a normalization is required:

$$W^t = \sum_{j=1}^{N_P} w_j^t; \qquad w_i^t \leftarrow \frac{w_i^t}{W^t},\ \forall i = 1..N_P \tag{6.7}$$

Setting & Publishing the position estimate The setEstimate() function parameterizes the particle set as a Gaussian density function. This Gaussian estimation is computed in order to publish a close result ready to be used by other real-time processes, monitoring or analysis. However, the particle set remains the genuine output of the particle filter. This step also sets the time stamp of the current estimate, τ^t. The parameters of the Gaussian density function are:

$$\hat{x}_r^t = \sum_{i=1}^{N_P} x_i^t \cdot w_i^t; \quad (\hat{\sigma}_x^t)^2 = \sum_{i=1}^{N_P} (x_i^t - \hat{x}_r^t)^2 \cdot w_i^t \tag{6.8}$$

$$\hat{y}_r^t = \sum_{i=1}^{N_P} y_i^t \cdot w_i^t; \quad (\hat{\sigma}_y^t)^2 = \sum_{i=1}^{N_P} (y_i^t - \hat{y}_r^t)^2 \cdot w_i^t \tag{6.9}$$

$$\hat{\theta}_r^t = atan(\frac{\sum_{i=1}^{N_P} sin\theta_i^t \cdot w_i^t}{\sum_{i=1}^{N_P} cos\theta_i^t \cdot w_i^t}) \tag{6.10}$$

$$(\hat{\sigma}_\theta^t)^2 = \sum_{k=1}^{N_P} (acos(cos(\theta_i^t - \hat{\theta}_r^t)))^2 \cdot w_i^t \tag{6.11}$$

$$\hat{\sigma}_{xy}^t = \sum_{k=1}^{N_P} (x_i^t - \hat{x}_r^t) \cdot (y_i^t - \hat{y}_r^t) \cdot w_i^t;\ \hat{\sigma}_{x\theta}^t = \hat{\sigma}_{y\theta}^t = 0; \tag{6.12}$$

Once the Gaussian form of the position estimate is set, publish() function sends through an output TCP port this result. Processes requiring position data should connect to this port in order to receive it in real-time.

Resampling Finally, the resampling() step draws a new particle set resampling the current one. The general resampling solution detailed in subsection 5.2.1 has been used. However, a slightly modification has been applied with good practical results. This modification issued from the idea to resample best particles (highest w_i^t) with new positions close to them while resampling the

worst particles with the aim to explore possible new solutions in the surroundings of the robot quite more far than when resampling best particles. This idea is implemented by fixing a diferent resampling noise matrix, C_γ, as a function of the drawn value r of the algorithm 9. C_γ is set as a 3×3 diagonal matrix with its elements being the variance parameters $\sigma^2_{\gamma_x}$, $\sigma^2_{\gamma_y}$ and $\sigma^2_{\gamma_\theta}$. Standard deviations related to each matrix element j follow the function:

$$\sigma_{\gamma_j} = \begin{cases} \sigma^{low}_{\gamma_j} & \text{if } r < r_\gamma \\ \sigma^{high}_{\gamma_j} & \text{if } r \geq r_\gamma \end{cases} \tag{6.13}$$

Therefore a set of parameters have been adjusted during experimental sessions to achieve a robust behavior of the position tracking.

6.2.2 Field Results within the URUS project

Within the context of the european project URUS (Ubiquitous networking Robotics in Urban Settings) [102, 112], the basic particle filter for 2D position tracking has been extensively tested on the UPC campus area onboard three diferent platforms belonging to three partners of the project:

- Tibi & Dabo: Segway RMP200, two-wheel, self-balancing platforms, Universitat Politècnica de Catalunya (UPC).
- Romeo: Four-wheel electrical car-like vehicle, Asociación de Investigación y Cooperación Industrial de Andalucia (AICIA).
- Ben,Ced & Dan: Pioneer P3AT, four-wheel platforms, Universidad de Zaragoza (UniZar).

In all cases the filter required at its input at least an odometry observation and a front laser scanner observation. Each partner provided the low-level acquisition software modules while the localization module was exactly the same for all platforms. Further details on software integration can be found in chapter 8. Next paragraphes detail experimental results with the three different platforms types above mentioned. Due to working constraints of the URUS project, experiments carried out with platforms belonging to AICIA and UniZar partners are not so extensive as those performed with UPC robots. However, they are reported here for the interest of the reader.

Table 6.1 lists the parameters used in these experimental sessions, their associated values, a short comment and a reference to the related equation.

Results with Tibi and Dabo

Tibi and Dabo are two mobile robots, based on the Segway RMP200 two-wheeled, self-balancing platform. For localization purposes they integrate wheel odometry data provided by the platform, and two laser scanner data, one pointing frontward and a second pointing backwards, both scanning parallel planes to a flat floor. Such platforms are contiuoulsy pitching due to their self-balancing system, so they are challenging vehicles for perception and estimation algorithms such as localization. Further information on Tibi and Dabo can be found in appendix A.

Table 6.1: Parameters for 2D localization experimental sessions

N_P	50 particles	Particle set size, eq. 6.2
ϵ_ρ	0.2	Translational noise for kinematic model, eq. 6.4
ϵ_θ	0.2	Rotational noise for kinematic model, eq. 6.4
σ_L	5 cm	Standard deviation for laser likelihood, eq. 6.6
$\sigma_{\gamma_x}^{low}$	7.5 cm	x low resampling noise, eq. 6.13
$\sigma_{\gamma_y}^{low}$	7.5 cm	y low resampling noise, eq. 6.13
$\sigma_{\gamma_\theta}^{low}$	0.02 rad	θ low resampling noise, eq. 6.13
$\sigma_{\gamma_x}^{high}$	15 cm	x high resampling noise, eq. 6.13
$\sigma_{\gamma_y}^{high}$	15 cm	y high resampling noise, eq. 6.13
$\sigma_{\gamma_\theta}^{high}$	0.04 rad	θ high resampling noise, eq. 6.13
r_γ	0.9	low/high resampling threshold , eq. 6.13

A set of experimental sessions to test a full autonomous navigation system for Tibi and Dabo mobile robots were conducted during autumn 2009 at UPC Campus Nord setting, running a total length of $3.5Km$ at a mean travelling speed of $0.67m/s$ [25], (one of the sessions is reported in video 4, appendix E. During the experiments, the robot executed a serie of 35 *go to* requests providing a map destination. After indicating this destination point on the map, the robot started its autonomous navigation to drive itself up to the goal point. Localization failed five times over all the sessions. Figure 6.2 shows the path executed by the robot and indicates where failures happened. Failures took place mainly in zones featuring environment elements poorly modelled by the 2D environment model, as for instance in ramps. At a ramp entry the robot sees the ramp surface as a wall, but also, on the ramps, such Segway-based platforms try to compensate the floor tilt, resulting that the front/back laser scanner mainly perceives the ramp surface when robot goes up/down. Moreover, these experimental sessions avoided to navigate through the central part of the UPC Campus, where there is a square (see figures 4.1 and first picture of figure 4.3). This square features a set of four large obstacles for gardening purposes, poorly modelled with a 2D approach due to their geometric construction. This means that in this area the 2D localization approach was not enough robust for autonomous navigation purposes.

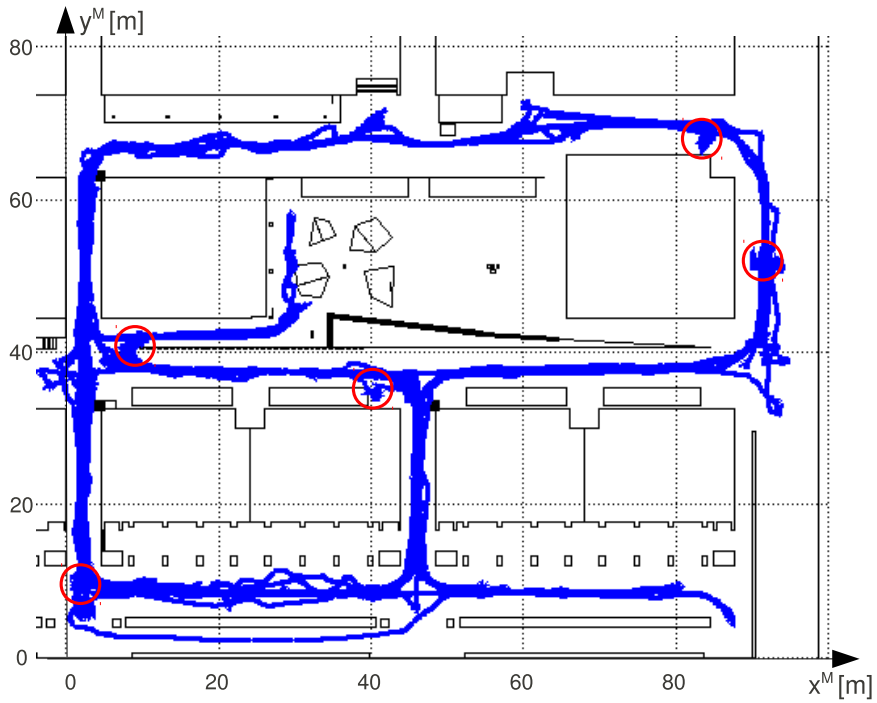


Figure 6.2: Path executed during the experimental sessions of autonomous navigation with Tibi and Dabo robots using 2D localization. Total travelled length was about $3.5Km$. Red circles indicate where the robot got lost.

Results with Romeo

Romeo robot is based on a four-wheel car-like platform, propulsed by an electrical motor. It belongs to AICIA. It is equipped with wheel encoders, an inertial unit, a GPS receiver, a frontal Sick laser scanner and a set of ultrasound beamers. For localization purposes, the AICIA colleagues provided an improved odometry thanks to the fusion of wheel encoders, inertial unit and GPS, and separately, they also provided raw laser scanner data. Figure 6.3 diagrams the filtering scheme, while figure 6.4 (left) shows a picture of the platform during an experimental session.

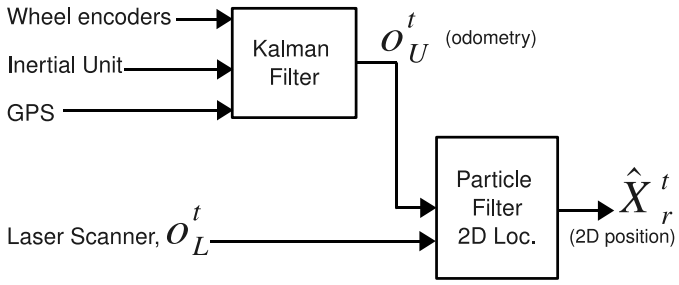


Figure 6.3: Filtering diagram for Romeo localization.

Romeo robot used the 2D position tracking for autonomous navigation. During an experimental session the robot executed a set of *go to* missions, completing a path of about 800 m. The entire path is shown in figure 6.4 (right). One of the key issues for the successful implementation of this approach on Romeo was the quality of the provided odometry data.

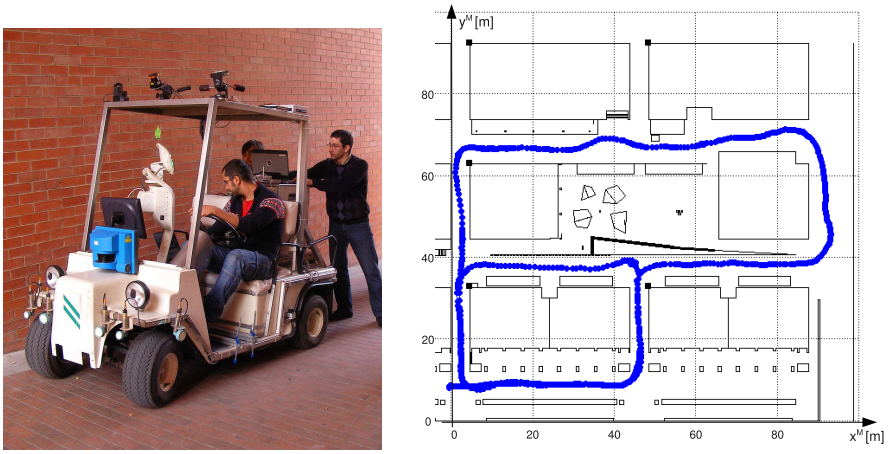


Figure 6.4: On the left, Romeo autonomous vehicle, from AICIA. On the right, travelled path of Romeo during an autonomous navigation session using the 2D position tracking.

Results with Ben, Ced and Dan

Ben, Ced and Dan are three Pioneer P3AT based platforms, belonging to UniZar. They were equiped with a sick laser scanner, mounted pointing forward. The platform itself also provides wheel odometry. Figure 6.5 shows a picture of the three robots running on the UPC Campus.

Figure 6.5: Ben, Ced and Dan robots during a demonstration of navigation in formation at UPC Campus.

In the context of the URUS project, the UniZar research team was performing a set of experiments showing cooperative issues on navigation in formation. They used the presented 2D position tracking module for real-time localization, fusing odometry and laser scanner data. Figure 6.6 shows a picture of the three robots and the path travelled by them during a project demonstration.

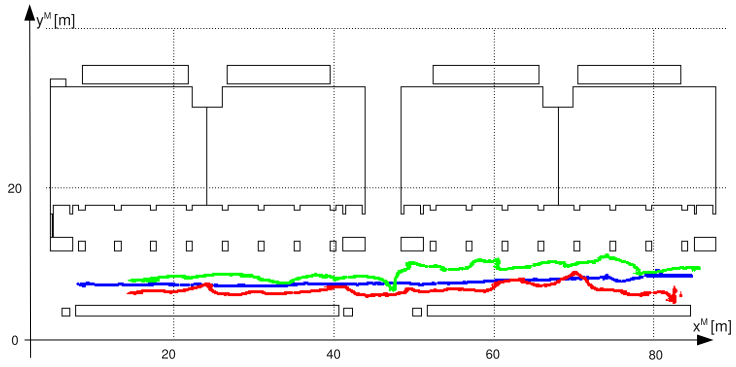


Figure 6.6: Path travelled by each robot during the demonstration of the navigation in formation. Pioneers were using the 2D position tracking.

6.3 Integrating asynchronous observations

This section describes an improvement of the basic 2D particle filter, consisting on modifying it to take into account the time stamp of real observations coming from sensor readings [22]. The approach is motivated mainly in two cases: fast vehicles and cooperative environments. In the former case little latencies of observation acquisition processes lead to observations related to past positions, so observation models should be computed also at past positions for a proper likelihood comparison. In the later case, even for slow platforms, communication delays can cause big latencies of the observations up to their availability at the filtering computer unit, so the same situation is found and observation models should be also computed taking into account observation time stamps.

In order to outline the proposed algorithm, some definitions are remembered or introduced:

- $O_k^t = (o_k^t, C_k^t, \tau_k^t, s_k^t)$ is an observation o_k^t, with covariance matrix C_k^t, arriving at the computing unit at iteration t, coming from the k^{th} observation process, made at continuous time τ_k^t and with status s_k^t (see section 3.1).
- Ω^t is the set composed by the observations available at the filtering computer unit at iteration t with the exception of the odometry (see section 5.1). These observations come from N_B observation processes. This set changes dynamically while filtering advances, since data acquisition processes run concurrently and asynchronously with the filtering process.
- $H^t = \{(\hat{X}^{t-\Delta}, \hat{C}^{t-\Delta}, \tau^{t-\Delta}), \ldots, (\hat{X}^{t-1}, \hat{C}^{t-1}, \tau^{t-1}), (\tilde{X}^t, \tilde{C}^t, \tau^t)\}$ is a set keeping the filter history of the Δ last posterior estimates and the last prior estimate made by the filter.

A single iteration of the proposed asynchronous particle filter integrates only those observations available at the filtering computer unit. Iteration t^{th} is outlined in algorithm 12.

For each observation process, the algorithm compares the observation time stamp with the last filtering time stamp, set inside the call *setEstimate()*. As a result of this comparison, two cases are identified whether if the observation is delayed or advanced with respect to the filtering time stamp. Figure 6.7 depicts the first case when $j < t$, thus the observation O_k^t is delayed with respect to the last prior time stamp τ^t. This figure shows how the particle X_i^t is backpropagated in order to compute observation models at positions where that particle was expected to be at the observation time τ_k^t. a, b and α are computed for linear interpolation between estimation points. The other case, evaluated in the algorithm with the statement **if** $j == t$, appears when the observation time stamp is advanced with respect the last prior time stamp. At this case, the filter propagates the particle set with the current odometry increments, and the advanced observation becomes a delayed one since the filtering time stamp is then updated at the *setEstimate()* calling.

6.3.1 Comparison between basic and asynchronous filters

In order to evaluate the performance of the proposed algorithm, it has been carried out a simulated experiment consisting on a mobile robot running on

Algorithm 12 Asynchronous particle filter iteration

INPUT: $P^{t-1}, H^{t-1}, \Omega^t$
OUTPUT: $P^t, (\hat{X}^t, \hat{C}^t, \tau^t), H^t_\Delta$

```
P^t=propagate(P^{t-1}, o_0^t)
(X̃^t, C̃^t, τ^t) =setEstimate(P^t)
H^t.pushBackCurrentEstimate((X̃^t, C̃^t, τ^t))
for k = 1..N_B do
    j =max ι ∈ {t − Δ, ..., t}|τ^ι ≤ τ_k^t
    if j == t then
        P^t=propagate(P^t, o_0^{t+})
        (X̃^t, C̃^t, τ^t) =setEstimate(P^t)
        H^t.replaceLastEstimate((X̃^t, C̃^t, τ^t))
        j = t − 1
    end if
    α = (τ^{j+1} − τ_k^t) / (τ^{j+1} − τ^j)
    X̂^H = αX̂^j + (1 − α)X̂^{j+1}
    ΔX = X̃^t − X̂^H
    for i = 1..N_P do
        X_i^{H,t} = X_i^t − ΔX
        p(X_i^{H,t}|o_k^t) = L_k(o_k^t, o_k^s(X_i^{H,t}))
        w_i^{t'} = w_i^{t'} · p(X_i^{H,t}|o_k^t)
    end for
end for
(X̂^t, Ĉ^t, τ^t) =setEstimate(P^t)
X̂^{t+}=propagate(X̂^t, o_0^{t+})
publish(X̂^{t+}, Ĉ^t, τ^t)
H^t.removeFormerEstimate()
H^t.replaceLastEstimate((X̂^t, Ĉ^t, τ^t))
resampling(P^t)
```

an environment of $10.000m^2$ at speed of about $2m/s$, completing a path of about $300m$. The simulated robot is equipped with two laser scanners, a compass and a GPS (coverage of about 60% of the path). Moreover, a camera network is deployed on the environment, covering about the 55% of the path and providing observations of the robot location (x, y). Table 6.2 summarizes the rates, latencies and the standard deviation of the simulated Gaussian noise for each observation process. These values were set taking into account real devices and systems.

The experimental testbench was composed by two computers. Computer 1 was executing the simulator, the basic and the asynchronous particle filters. Computer 2 executed the GUI and was saving the frames in order to produce the video (see video 1, appendix E). This scenario allows to compare the two filters in real-time with the same conditions since they are running on the same simulation execution. Further details on the software can be found in chapter 8. For both filters, the number of particles was set to $N_P = 100$. Figure 6.8 shows the map, the ground truth position in black, the basic filter estimate in blue, the asynchronous filter estimate in red and the odometric position in green. In

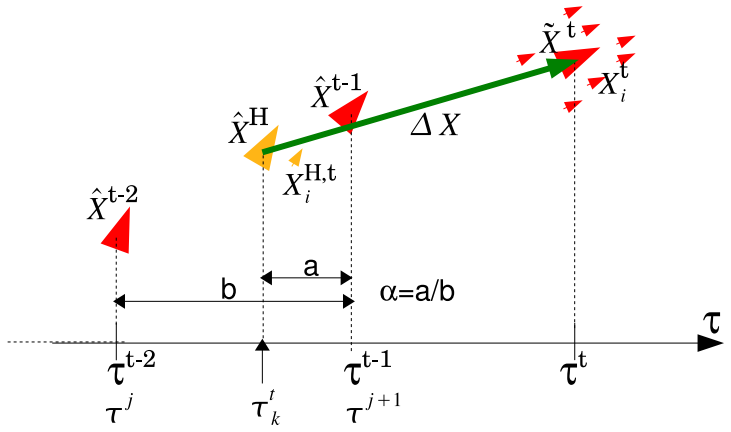


Figure 6.7: Backward propagation of particles when integrating observation o_k^t. Particle state X_i^t becomes $X_i^{H,t}$. Expected observations are computed from $X_i^{H,t}$.

Table 6.2: Rates and latencies of the observation process

	Observation process	Rate(Hz)	Latency(ms)	Std. dev. noise
o_0^t	odometry	20	∼0	$5\%(\delta XY), 10\%(\delta\theta)$
o_1^t	front laser	4	50	$5cm\ (range)$
o_2^t	back laser	4	50	$5cm\ (range)$
o_3^t	compass	5	20	$1.5°(\theta)$
o_4^t	GPS	1	20	$2m\ (x, y)$
o_5^t	CameraNet	1	500	$0.4m\ (x, y)$

this figure cameras are also drawn as small black squares.

Using this testbench three experiments are presented. Experiment A was switching off the camera network, thus both filters were integrating only the observations provided by onboard sensors. In experiment B we have switched on the camera network, thus both filters integrate also remote observations provided by these cameras. After a discussion, a third experiment demonstrates the potentialities of this approach in the case of a robot fusing only odometry and camera network detections.

To evaluate the performance of the filters we evaluate the following error figures:

$$\begin{aligned} e_x^t &= \hat{x}_r^t - x_r^\tau;\ e_y^t = \hat{y}_r^t - y_r^\tau;\ e_\theta^t = \hat{\theta}_r^t - \theta_r^\tau \\ e_{xy}^t &= \sqrt{(\hat{x}_r^t - x_r^\tau)^2 + (\hat{y}_r^t - y_r^\tau)^2} \end{aligned} \tag{6.14}$$

To compute this error we linearly approximate the simulated ground truth $(x_r^\tau, y_r^\tau, \theta_r^\tau)$ data at exact times where estimations are computed. This is done by considering the ground truth sample immediately before and after the estimate time stamp. The ground truth process was running at $20Hz$.

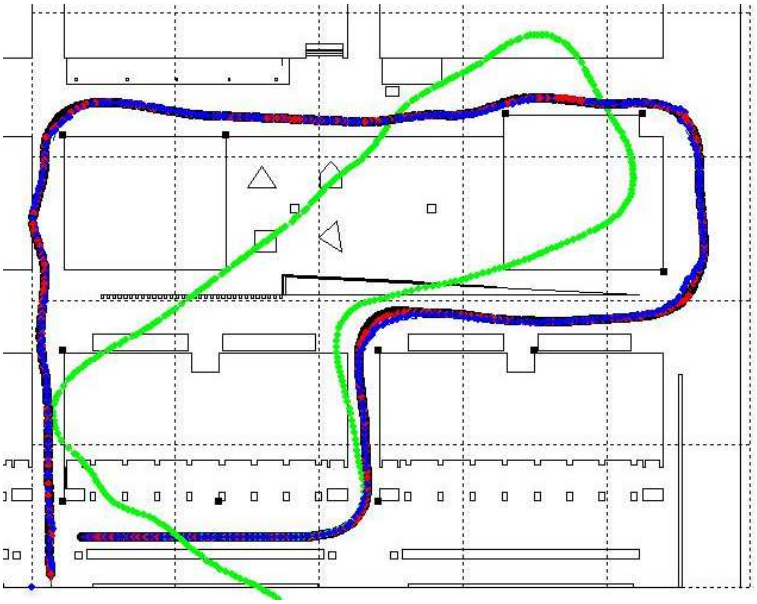


Figure 6.8: Ground truth (black), basic (blue) and asynchronous (red) particle filter estimates, and odometry (green) on the map. Cameras are drawn as little black squares. Dotted lines build a $20m \times 20m$ grid.

Experiment A: Camera Network Off

Figure 6.9 shows the error e^t_{xy} when the camera network was switched off. In this case the observation processes provide data with low latencies, therefore the asynchronous filter does not take clearly advantage of its properties. However, the proposed approach performs slightly better, since observations are integrated properly considering their time stamps.

Experiment B: Camera Network On

When a camera network is switched on, we put in the scenario a very accurate observation process that, however, provides observations at low rate and with large latencies. In this scenario, the proposed asynchronous filter performs much better than the basic one as it is shown in figure 6.10. The asynchronous filter outperforms the basic one with the exception of a short passage, where two filters have demonstrated a good recovery behaviour. This execution is recorded and presented in the video 1, appendix E, where the particle sets of each filter can be seen with the simulated ground truth position of the robot.

For this second experiment with the camera network switched on, the error for each estimated component of the state is presented, with the respective estimated standard deviation. Figure 6.11 shows how the filter error for each component remains most of the time within the uncertainty bound of 1σ.

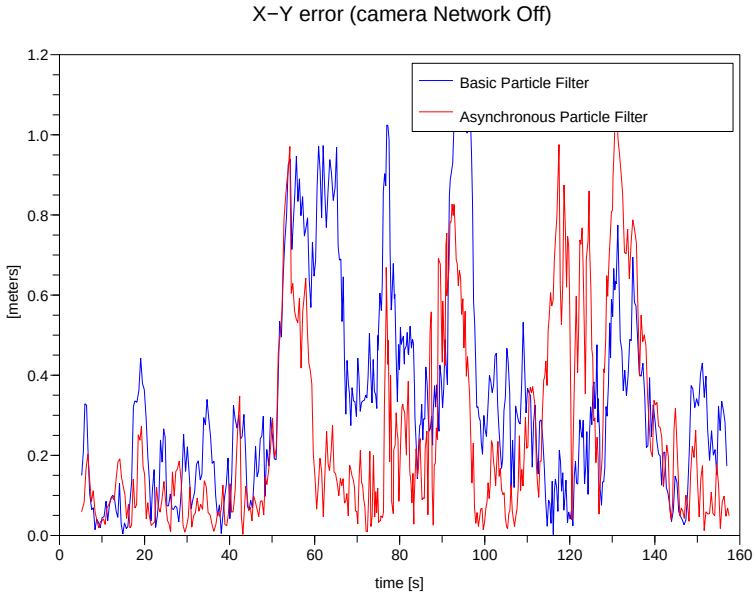


Figure 6.9: XY error, e^t_{xy}, of both filters. Camera network is switched off.

Discussion

Table 6.3 summarizes the mean errors for both filters and both experiments A & B. As expected, the proposed approach works much better, specially when an accurate but delayed observation process plays in the scene, as the case when the camera network is switched on. Table 6.3 also shows how the θ estimate does not improve its performance since it depends basically of the odometry and the compass, and these two observation process have high rates and low latencies.

Table 6.3: Mean errors of A & B experiments

	Basic PF		Asynchronous PF	
CamNet	$\mu(e_{xy})[m]$	$\mu(e_\theta)[rad]$	$\mu(e_{xy})[m]$	$\mu(e_\theta)[rad]$
OFF	0.36	0.013	0.28	0.012
ON	1.05	0.013	0.26	0.012

Table 6.3 also compares the asynchronous filter with and without the camera network and shows that only a little improvement appears in terms of position error, but gains in terms of robustness since another observation process is integrated on the filter. From this consideration, the next experiment evaluates the feasability of tracking the position of a robot fusing only the odometry and the camera network observations, in order to consider the proposed algorithm as a practical solution to be onboard of cheap robots running on environments where a camera network has been deployed. Experiment C investigates this issue.

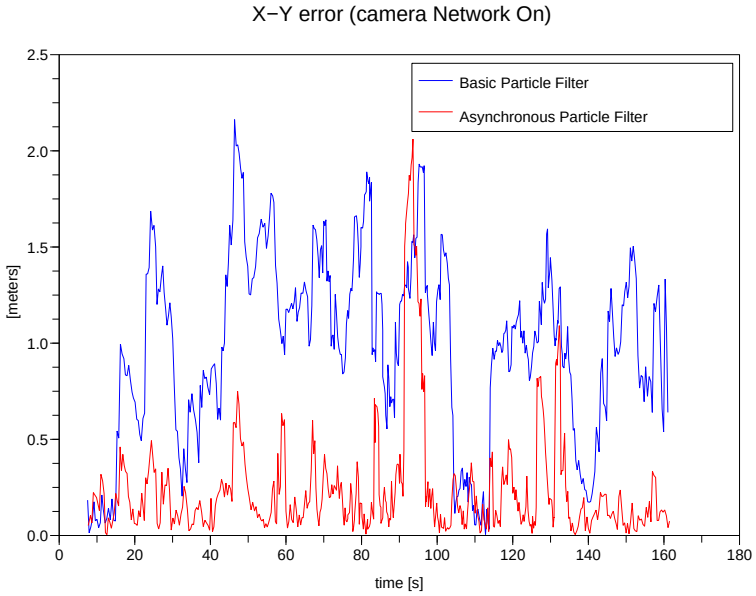


Figure 6.10: XY error, e^t_{xy}, of both filters. Camera network is switched on.

Experiment C: Position tracking with odometry and camera network

Once the asynchronous filter has shown good properties integrating observations with high latencies, this third experiment investigates a fusion scheme with only odometry and camera network data. These two observations are very complementary since odometry has a high rate, a small latency and a good accuracy in short displacements, while a camera network provides absolute and accurate (x, y) observations with a large latency, but does not suffer from accumulated drifts as odometry does. This experiment uses the same testbench as the previous ones, but only the asynchronous filter is executed since the basic filter was unable to track the robot position in a robust way. Video 2, outlined in appendix E, shows the evolution of the particle set during the experiment.

Figure 6.12 depicts the error of this experiment for the three estimated position components. This figure shows how the error increments when the robot is out of coverage of the camera network and redcuces when the robot integrates remote observations coming from the camera network. Even if the coverage of the camera network is partial (55% of the path) the proposed approach is able to track the position of the robot with an acceptable error. Mean errors for this experiment were $\mu(e_{xy}) = 0.7m$ and $\mu(e_\theta) = 0.07rad$. They can be compared with those errors of table 6.3 where the filters integrated all observations from all sensory subsystems. Obviously, the filter integrating all observations performs better, but the interest of this result lies in the fact that cheap robots with only wheel encoders could track its position taking benefit of observations coming from a deployed camera network with partial coverage of the environment.

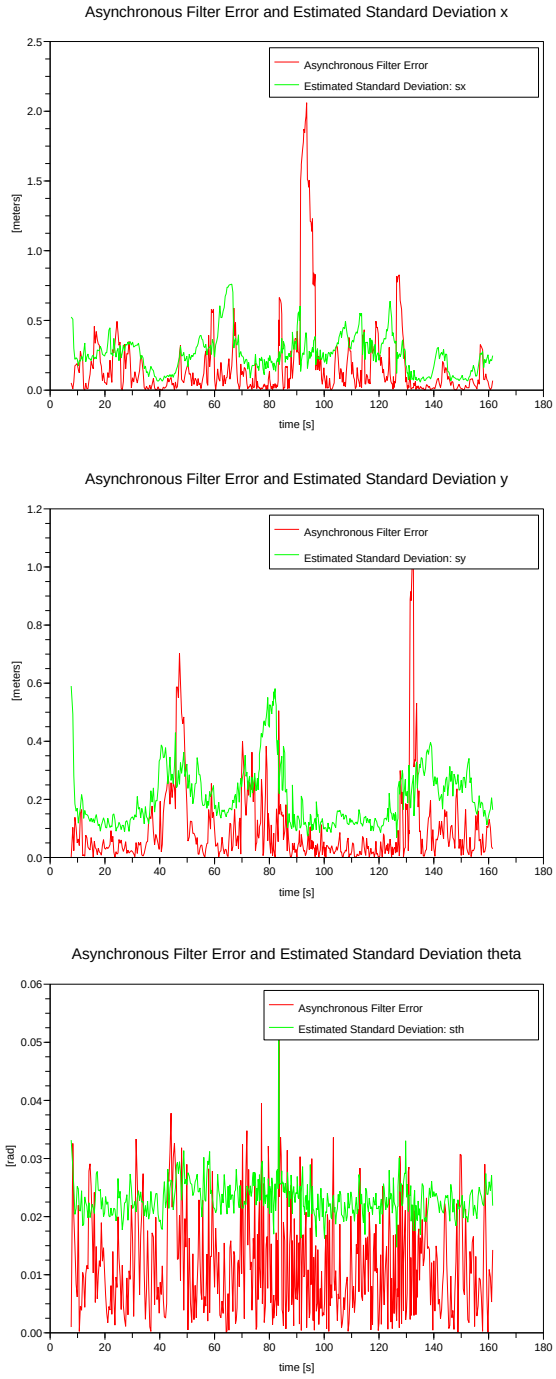


Figure 6.11: e_x^t (top), e_y^t (middle), e_θ^t (down) and respective estimated standard deviations $\hat{\sigma}_x^t, \hat{\sigma}_y^t, \hat{\sigma}_\theta^t$ for the asynchronous filter (experiment B).

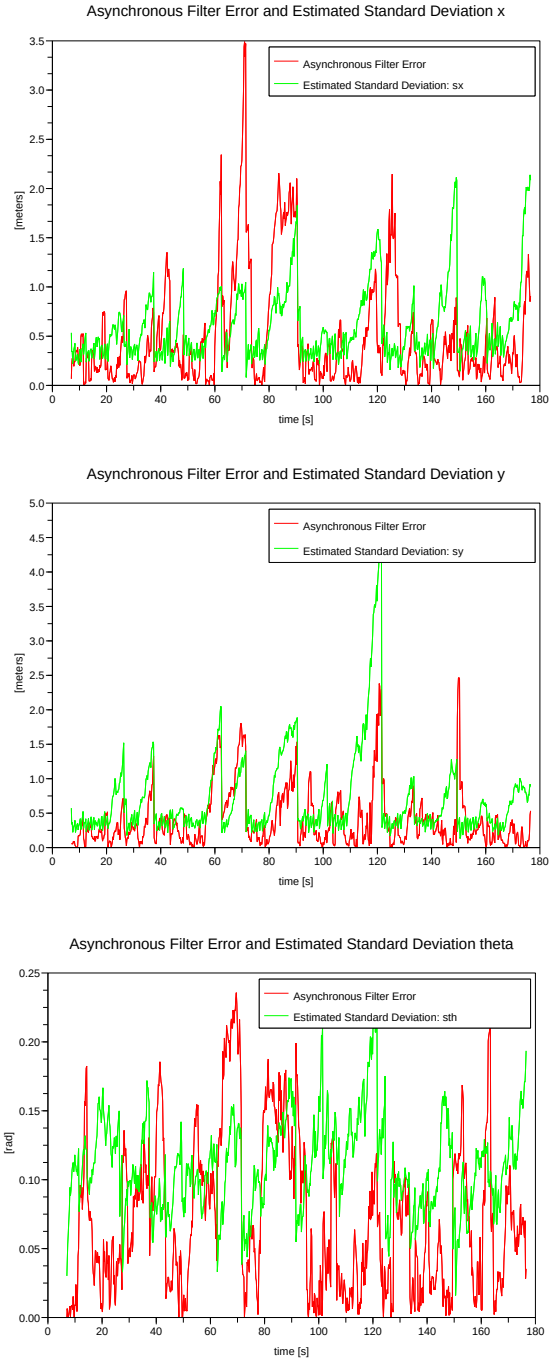


Figure 6.12: e_x (top), e_y (middle), e_θ (down) and respective estimated standard deviations $\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_\theta$ for the asynchronous filter only integrating odometry and camera network observations (experiment C).

6.4 3D Position Tracking

This section considers the robot moving in a 3D world, thus it represents a forward step from the previous 2D solution. The state space considered in this approach is that of 3D positions, parameterized as a (x, y, z) location referenced to the map frame, and three Euler angles, heading, pitch and roll, (θ, ϕ, ψ), defined following the $Z-Y-X$ Euler convention. In this section, all positions will be referenced to the map frame if no specific superindex or comment indicates otherwise.

At each iteration t, the filter produces a set of particles, P^t, where each particle is a pair formed by a sample of the state space (i.e. a position) and a weight:

$$P^t = \{(X_1^t, w_1^t), \ldots, (X_{N_P}^t, w_{N_P}^t)\};\ X_i^t = \{(x_i^t, y_i^t, z_i^t, \theta_i^t, \phi_i^t, \psi_i^t)\};\ w_i^t \in [0,1] \tag{6.15}$$

where the pair (X_i^t, w_i^t) is the i^{th} particle produced by the t^{th} iteration.

6.4.1 Basic Particle Filter

Kinematic Model In particle filtering, having a motion model allows to propagate the particle set, thus limiting the search space to positions satisfying the motion model constrained to given sensor inputs (see section 5.2). A probabilistic kinematic model, such as that proposed at [109], computes a new sample set called the *prior*, P^{t^-}, based on the previous set, P^{t-1}, constrained to the platform's motion. As defined in section 3.2, the odometry observation available at iteration t is:

$$o_U^t = (\Delta\rho_U^t, \Delta\theta_U^t) \tag{6.16}$$

where $\Delta\rho^t$ is the translational 2D increment in the local XY plane, and $\Delta\theta^t$ is the rotational increment around the local Z axis of the platform. Both increments are the accumulated odometry from iteration $t-1$ to iteration t. Moreover, for pitch-free platforms, such as the Segway RMP200 (see appendix A), an observation about the pitch motion gives to the filter an extra and useful information that can be also integrated in the kinematic model. This observation is provided by an inclinometer device and represents the pitch increment from iteration $t-1$ to iteration t:

$$o_I^t = \Delta\phi_I^t \tag{6.17}$$

With both odometry and pitch increments, and noise model parameters $(\epsilon_R, \epsilon_\theta, \epsilon_\phi)$, at the beginning of each iteration the state vector of the i^{th} particle is moved

according the following probabilistic kinematic model:

$$\begin{aligned}
\tilde{\Delta}\rho_i^t &= \mathcal{N}(\Delta\rho_U^t, \sigma_\rho^t);\ \sigma_\rho^t = \epsilon_\rho \Delta\rho_U^t \\
\tilde{\Delta}\theta_i^t &= \mathcal{N}(\Delta\theta_U^t, \sigma_\theta^t);\ \sigma_\theta^t = \epsilon_\theta \Delta\theta_U^t \\
\tilde{\Delta}\phi_i^t &= \mathcal{N}(\Delta\phi_I^t, \sigma_\phi^t);\ \sigma_\phi^t = \epsilon_\phi \Delta\phi_I^t \\
x_i^t &= x_i^{t-1} + \tilde{\Delta}\rho_i^t \cos(\theta_i^{t-1} + \frac{\tilde{\Delta}\theta_i^t}{2}) \\
y_i^t &= y_i^{t-1} + \tilde{\Delta}\rho_i^t \sin(\theta_i^{t-1} + \frac{\tilde{\Delta}\theta_i^t}{2}) \\
\theta_i^t &= \theta_i^{t-1} + \tilde{\Delta}\theta_i^t \\
\phi_i^t &= \phi_i^{t-1} + \tilde{\Delta}\phi_i^t
\end{aligned} \tag{6.18}$$

where the $\mathcal{N}()$ stands for a call to a random generator normally distributed, with its mean set to current observation data. Standard deviation for odometry increments are set proportional to data values, so that large odometry increments imply more uncertainty as presented in section 3.2.

The proposed kinematic model does not modify z_i^t and ψ_i^t since these two variables are constrained by gravity. These two coordinates will be set by using the ground constraints presented at section 4.3.3:

$$\begin{aligned}
z_i^t &= g_z(x_i^t, y_i^t) \\
\psi_i^t &= g_\psi(x_i^t, y_i^t, \theta_i^t)
\end{aligned} \tag{6.19}$$

Algorithm Outline Using the above presented probabilistic kinematic model, the procedure to compute ground constraints described in 4.3.3, the technique to compute expected laser observations in 3D models detailed in 4.3.5) and the likelihood function to compare laser scanner observations presented in 6.2.1, algorithm 13 overviews how to combine all these elements to build a particle filter to estimate the 3D position of the robot within the map coordinate frame. This localization algorithm fuses data from wheel encoders, platform inclinometers, front and back horizontal laser scanners and front vertical laser scanner. Details on sensor setup can be found on appendix A.

The filter is initialized with a 2D position provided by the user, (x_0, y_0, θ_0). The first particle set is initialized within a square of $4m^2$ around the (x_0, y_0) location, within heading range $\theta_0 \pm 3°$, and a pitch equal to zero. These initial values are based on the fact that the localization filter is initialized with the robot stopped in a roughly known position and on flat terrain. After propagation and correction steps, in order to output a close estimation of the filter, a gaussian parameterization of the particle set is performed. The robot position estimate, $\hat{X}_r^t$, is computed as the weighted mean of the particle positions, while the covariance parameters, $\hat{C}^t$, are computed as the weighted sample variance, in the same way as described in equations 6.2.1. In the last step of the filter, a resampling function draws a new particle set keeping the mean of the current one. Resampling is necessary to avoid particle depletion [28, 5], an undesired phenomenon of particle filters where the particle set collapses to a single state point rendering the filter no longer capable of exploring new solutions for the estimation, and therefore compromising its robustness (see subsections 5.2.1 and 6.2.1 for further details on resampling).

As an aside, the vertical laser is integrated into the correction stage only when appropriate. Most unmodeled obstacles, such as pedestrians or bicyclists, have a relatively small footprint on the XY plane, so that the horizontal lasers remain usable despite numerous occlusions. On the other hand, the vertical scanner can be nearly fully occluded by a single pedestrian a few meters in front of the robot. In that scenario the filter attempts to match actual and expected observations by pitching the robot forward, lifting the floor surface towards the part of the scan corresponding to the pedestrian, and thus increasing the similarity between scans. This is clearly inadequate and compromises the filter's performance, so vertical laser data is only integrated when the similarity between current real and expected observations, computed by the likelihood function, is greater than a threshold, λ_{L_V}, determined experimentally.

Algorithm 13 Particle filter localization algorithm

INITIAL INPUT: $(x_0, y_0, \theta_0), \mathcal{M}$
CONTINUOUS INPUT: $O_U^t, O_I^t, O_{L_F}^t, O_{L_B}^t, O_{L_V}^t$ //sensor observations
OUTPUT: $\hat{X}_r^t, \hat{C}_r^t, \tau^t$ //robot position, associated uncertainty and time stamp

$t = 0$; //iteration counter
$P^0 = initialization(x_0, y_0, \theta_0)$; //initialization with prior knowledge
while running **do**
  $t \leftarrow t + 1$
  for $i = 1 \dots N_P$ **do**
    $(x_i^t, y_i^t, \theta_i^t, \phi_i^t) = f(X_i^{t-1}, O_U^t, O_I^t)$; //kinematic model
    $z_i^t = g_z(x_i^t, y_i^t)$; //ground height constraint
    $\psi_i^t = g_\psi(x_i^t, y_i^t, \theta_i^t)$; //ground roll constraint
  end for
  $\tau^t = timeStamp(NOW)$;
  for $i = 1 \dots N_P$ **do**
    if $\mathcal{L}_L(O_{L_V}^t, O_{L_V}^s(X_i^t)) < \lambda_{L_V}$ **then**
      $\mathcal{L}_L(O_{L_V}^t, O_{L_V}^s(X_i^t)) = 1$;
    end if
    $w_i^t = \prod_{k=L_F, L_B, L_V} \mathcal{L}_L(o_k^t, o_k^s(X_i^t))$ //correction
  end for
  for $i = 1 \dots N_P$ **do**
    $w_i^t \leftarrow \frac{w_i^t}{\sum_{j=1}^{N_P} w_j^t}$; //normalization
  end for
  $(\hat{X}_r^t, \hat{C}_r^t) = gaussianParameters(P^t)$;
  $publish(\hat{X}_r^t, \hat{C}_r^t, \tau^t)$; //publish produced data
  $P^t \leftarrow resampling(P^t)$; //redraw a new particle set by resampling
end while

6.4.2 Field results during autonomous navigation sessions

The 3D position tracking was validated in the context of a full autonomous navigation system over the course of four experimental sessions [111], three at the UPC Campus area and one on a public avenue situated at the Gràcia district of Barcelona, using Tibi and Dabo robots without distinction (see appendix A).

All navigation processes run concurrently on a single laptop while an external computer, connected to the on-board computer via wireless, was used for on-line monitoring and to send arbitrary *go-to* requests (XY coordinates over the map) to the path planning module. Localization process executed at $5Hz$ with $N_P = 50$ particles. Runtime for all experiments added up to 2.3 hours, with over $6km$ of autonomous navigation. We set a speed limit of $0.75m/s$ for the first session, and increased it to $0.85m/s$ for the following three sessions, being this a soft limit since the robot often travels faster due to its self-balancing. One of the experimental sessions in the UPC Campus is reported in video 5, appendix E.

Table 6.4 lists the parameters used in these experimental sessions, their associated values, a short comment and a reference to the related equation.

Table 6.4: Parameters for 3D localization experimental sessions

N_P	50 particles	Particle set size, eq. 6.2
ϵ_ρ	0.2	Translational noise for kinematic model, eq. 6.18
ϵ_θ	0.2	Rotational noise for kinematic model, eq. 6.18
ϵ_ϕ	0.2	Pitch noise for kinematic model, eq. 6.18
σ_L	$5cm$	Standard deviation for laser likelihood, eq. 6.6
λ_{L_V}	0.1	Threshold to discard vertical scans, alg. 13
$\sigma_{\gamma_x}^{low}$	$10cm$	x low resampling noise, eq. 6.13
$\sigma_{\gamma_y}^{low}$	$10cm$	y low resampling noise, eq. 6.13
$\sigma_{\gamma_\theta}^{low}$	$0.02rad$	θ low resampling noise, eq. 6.13
$\sigma_{\gamma_x}^{high}$	$30cm$	x high resampling noise, eq. 6.13
$\sigma_{\gamma_y}^{high}$	$30cm$	y high resampling noise, eq. 6.13
$\sigma_{\gamma_\theta}^{high}$	$0.06rad$	θ high resampling noise, eq. 6.13
r_γ	0.9	low/high resampling threshold , eq. 6.13

Results are displayed in table 6.5. For each experimental session, table 6.5 lists the number of requests sent to the robot, the navigation distance D, as estimated by the localization module, the total navigation time t_{nav} (understood as that spent by the robot attending a *go-to* request), an estimation of the average translational speed,$\hat{v}$, computed using the previous values, $\hat{v} = D/t_{nav}$, the number of navigation errors and the success rate of these errors over the total *go-to* requests queried to the robot.

Table 6.5: Experimental results

Site & Date	Requests [#]	D [m]	t_{nav} [s]	$\hat{v}$ [m/s]	Errors [#]	Success [%]
Gràcia, 20-May-2010	33	777.7	1108	0.71	0	100
Campus, 3-Jun-2010	23	858.5	1056	0.81	0	100
Campus, 22-Jun-2010	55	2481.8	3426	0.72	0	100
Campus, 23-Jun-2010	60	2252.5	2727	0.83	2	96.7
Accumulated	171	6370.5	8317	0.77	2	98.8

We were allowed three days (mornings only) to conduct experiments at the Gràcia site, the last of which was dedicated to a public demonstration, and

so the scope of that session is limited, totaling less than $1Km$ of autonomous navigation. Even so it must be noted that, due to time constraints, these experiments were conducted with little to no prior in-site testing. Moreover, while part of the area was fenced, many pedestrians and bicyclists disregarded instructions and crossed the area anyway. This fact proves the robustness of the localization approach in new environments and under different conditions.

The four sessions are plotted in figure 6.13. For the session at the Gràcia site, the rightmost passageway was fenced while pedestrians and bicyclists were allowed to use the one on the left. The rest of the area was left as-is, except for four fences placed below the monument, at $y = 20m$ (fig. 6.13, top-left plot), as a safety measure. The second session, already at the Campus site, starts at $(90, 38)m$, and ends at $(17, 69)m$ when the robot encounters a large section occupied by public works and thus unmapped. In the third session the robot moved successfully through the passageway between C and D buildings, which is on the verge of the experimental area and was roughly mapped, and hence did not revisit. The robot also had the opportunity to navigate the narrow passageway to the right of the FIB square, which is usually occupied by the cafeteria's terrace. Please note that areas where the localization estimate is within a building, such as for A5, A6 and C6, are outdoor covered passages like that shown in the bottom of picture 4.3.

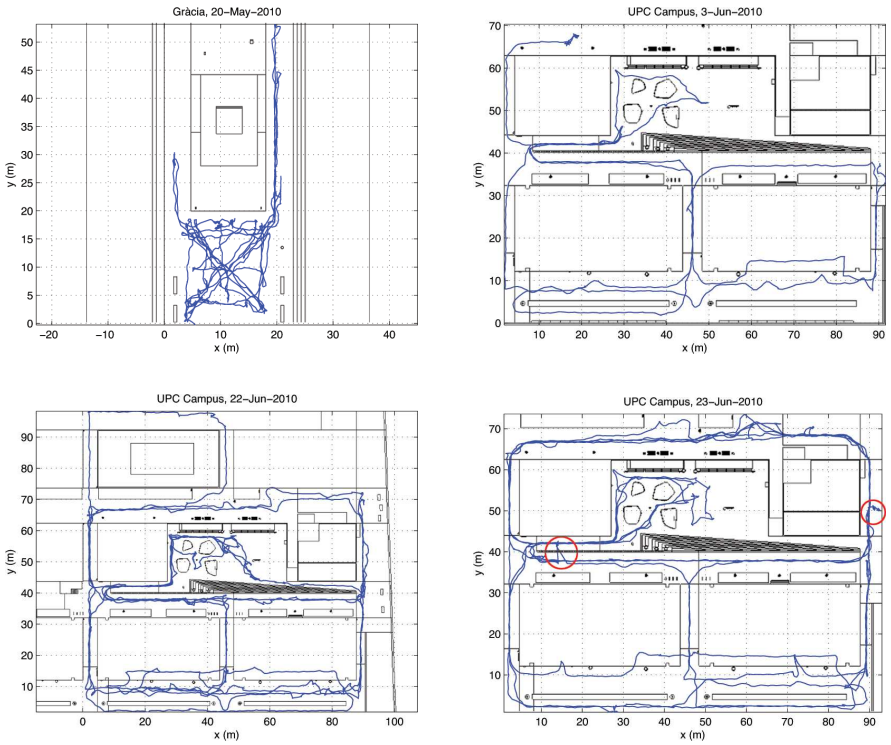


Figure 6.13: Localization results for the four experimental sessions. Red circles in the bottom left figure mark failure points.

The fourth run contains the only two errors we encountered. Both are related to the localization algorithm, and were mainly due to features of the terrain.

Having failures gives us the chance to learn, advance and improve our system. Therefore, the following paragraphs provide insights to the two localization failures that occurred during the last session, at the Campus site. This analysis was made possible by the off-line study of the logged data for that session, automatically stored by our software. Our localization module can be executed off-line using dummy sensors, which publish logged sensor data under the same interfaces as during on-line executions, while keeping synchrony. This allows us to run new real-time, off-line executions of the localization process with the data collected during on-line executions.

The first failure happened approximately at XY point $(90, 50)m$. The robot was traveling from left to right along $y = 38m$ and turned to its left to go up the ramp at $x = 90m$ (fig. 6.13). After turning, the localization uncertainty grew up, while the position estimate deviated very clearly from the true position as perceived by the team members, finally causing a navigation error. This was due to two causes. Firstly, the robot passed over a big terrain irregularity (a crack on the pavement) just before starting the turning maneuver, reported by odometry data as a period with high roll oscillations (fig. 6.14) and noisy heading increments. This approach constrains the roll component to the 3D model, assuming that the platform is a rigid body, so that roll oscillations caused by surface irregularities are not tracked well by the filter, as can be seen in figure 6.14. Secondly, this happened around the bottom-right corner of the B6 building, which has large, floor-to-ceiling glass windows, modeled in the 3D map as walls. Off-line inspection of the front laser data shows how in many instances the laser beam goes through the windows before the robot turns to face the ramp (fig. 6.14). Modeling this behavior would require a complex observation model, since it depends on outdoor and indoor window lighting, as well as on ray incidence, thus this being one of the main limitations for laser devices. Figure 6.14 also shows the presence of three pedestrians (team members) close to the robot, blocking three important sectors of the back laser scan. The odometry issue led to noisy particle propagation, while the laser issue led to poor filter correction. The combination of both events caused a localization error. 20 real-time, off-line executions of the localization filter have been performed at this point with the logged data, resulting in a failure ratio of 45%, clearly indicating that it was a challenging situation.

The second localization failure was due to faulty odometry data, again after passing over a big terrain irregularity. Our localization approach can filter noisy data peaks, but this case was extreme as odometry data was clearly incorrect for both translation and rotation for approximately 1.2 seconds, providing odometry increments around $0.4m$ and $-8°$ while the odometry acquisition period was $T_{odo} = 0.1s$. This data is clearly erroneous as the robot was at this time moving straight ahead at a speed of approximately $1m/s$ (fig. 6.15). This was the first time that such an error was reported on our platforms, and the localization filter did not check the coherence of odometry data before using it for particle propagation (see equation 6.18). Using faulty data for about 6 consecutive iterations caused the localization estimate to advance and turn right with no chance for recovery in the filter's correction step. This can be clearly seen in figure 6.13, where the robot jumped from (16,37) to (13,42). After acknowledging the error we relocalized the robot manually and resumed the experiments, hence the second jump to the correct position. The terrain irregularity that caused this error was another crack in the pavement. These are frequent throughout the campus

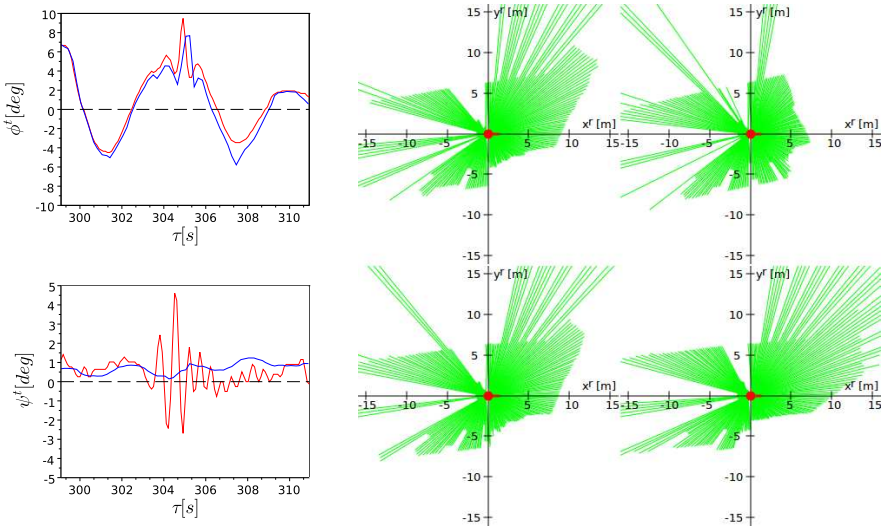


Figure 6.14: Left: pitch (top) and roll (bottom) data provided by the platform (red) and estimated (blue) at the surroundings of the first failure. Right: four consecutive laser scans just before the failure. Note how the front laser beams penetrate in many instances the windows at $y^R \sim 6m$, and also the presence of some pedestrians blocking three large sectors of the back laser scan.

and occasionally cause the robot to take a small jump and thrash sideways.

6.4.3 Filter improvements

These two failures teach that robust navigation is still an open issue for mobile robots operating in urban pedestrian areas. Three improvements of the filter have been applied: (1)using velocity data from the Segway platforms, (2) integrate roll increment in the propagation step and compute a likelihood in the correction step comparing particle roll with expected roll, the last given by the ground constraint and (3) a new approach on the resampling method taking into account odometric increments. These improvements have produced a new version of the particle filter 3D localization that has been used in the following sections of this chapter.

Integrating velocities Velocities are instantaneous observations computed by the Segway platform embedded microprocessor, so that at each reception of platform data by the computer running localization, a new velocity pair is also available:

$$o_v^{t_r} = (v_\rho^{t_r}, v_\theta^{t_r}) \tag{6.20}$$

where t_r stands for the iteration of the reception process but not of the localization filter. This velocity pair is accompained with a time stamp τ^{t_r} measured by the platform acquisition process. The reception process is in charge of integrating these velocities to update a pair of odometric increments, by measuring the time ellapsed from the last reception up to the current one. After each use of these odometric increments by the localization filter, they are reset, thus the

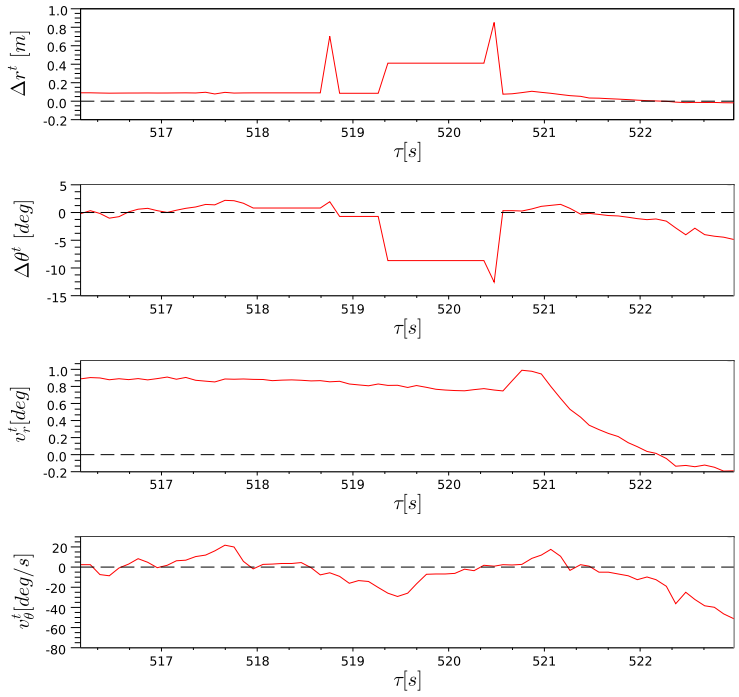


Figure 6.15: Odometry increments and velocities provided by the Segway platform during the second localization failure. Acquisition period is $T_{odo} = 0.1s$. After a first peak in translation at $\tau = 518.8s$, odometry increments are faulty during the time period $[519.3, 520.5]s$, while platform velocities remain coherent. The robot was moving straight ahead at approximately $1m/s$.

accumulation restarts. Once the odometric increments are updated using velocity data, the localization filter uses them as it used the odometric increments provided directly from encoders (see equation 6.18).

Roll data integration Segway embedded inclinometers also provide roll data. This leads to reformulate equation 6.17 as follows:

$$o_I^t = (\Delta\phi_I^t, \Delta\psi_I^t) \tag{6.21}$$

This improvement consists on using platform roll increments in the same way than the pitch increments were used in equation 6.18. Moreover, as discussed in section 4.3.3, roll component is mainly constrained by the gravity and the environment model. Taking benefit of this remark, in the correction step of the filter, the i^{th} particle is also scored with the result of a likelihood between its actual roll and the expected one according the ground constraint:

$$L_\psi(\psi_i^t, g_\psi(x_i^t, y_i^t, \theta_i^t)) = \text{erfc}\left(\frac{|\psi_i^t - g_\psi(X_i^t)|}{\sigma_\psi\sqrt{2}}\right) \tag{6.22}$$

Adaptative resampling with odometry increments Resampling method proposed in section 6.2.1 was also used in 3D approach. However, setting fixed

noise parameters for resampling causes problems when the filter iterates at higher rates, since odometry increments are smaller and resampling noise can overpass them, leading to a divergence of the particle set. Moreover, if the filter rate is higher than acquisition rate of horizontal lasers, there are iterations where resampling is executed with only the vertical laser correction, which is not so informative than horizontal ones, so that particle set dispersion grows up even more.

To overcome this effect, an alternative way to set resampling noise values is proposed, based on using current odometry increments to set the matrix C_γ^t, which now depends on the iteration index t. Resampling noise values are set as:

$$\begin{aligned} \sigma_{\gamma_x}^t &= K_\gamma \Delta\rho_U^t; \quad \sigma_{\gamma_y}^t = K_\gamma \Delta\rho_U^t \\ \sigma_{\gamma_\theta}^t &= K_\gamma \Delta\theta_U^t; \quad \sigma_{\gamma_\phi}^t = K_\gamma \Delta\phi_I^t \end{aligned} \tag{6.23}$$

Following this approach, the only parameter to be adjusted is the constant K_γ. However, for high rate filters, or in situations where the platform is at low speeds or stopped, increments are so small that resampling noise approaches to zero, thus particle depletion arises, causing that a large mass of the particle set samples the same state point. To avoid this situation minimum values are set for each noise component: $\sigma_{\gamma_x}^{min}, \sigma_{\gamma_y}^{min}, \sigma_{\gamma_\theta}^{min}, \sigma_{\gamma_\phi}^{min}$

Testing new filter version To check these proposed improvements in challenging situations such as ones causing failures, a new version of the filter has been implemented incorporating the above presented improvements. Table 6.6 lists the parameters used in these experimental sessions, their associated values, a short comment and a reference to the related equation.

Table 6.6: Parameters for 3D localization off-line experimental sessions

ϵ_ρ	0.2	Translational noise for kinematic model, eq. 6.18
ϵ_θ	0.2	Rotational noise for kinematic model, eq. 6.18
ϵ_ϕ	0.2	Pitch noise for kinematic model, eq. 6.18
ϵ_ψ	0.2	Roll noise for kinematic model, eq. 6.18 and eq. 6.21
σ_L	$5cm$	Standard deviation for laser likelihood, eq. 6.6
σ_ψ	$0.01rad$	Standard deviation for roll likelihood, eq. 6.22
λ_{L_V}	0.1	Threshold to discard vertical scans, alg. 13
$\sigma_{\gamma_x}^{min}$	$5cm$	minimum x resampling noise, eq. 6.23
$\sigma_{\gamma_y}^{min}$	$5cm$	minimum y resampling noise, eq. 6.23
$\sigma_{\gamma_\theta}^{min}$	$0.02rad$	minimum θ resampling noise, eq. 6.23
$\sigma_{\gamma_{phi}}^{min}$	$0.02rad$	minimum ϕ resampling noise, eq. 6.23
K_γ	1	Constant gain for resampling noise, eq. 6.23

A set of 12 off-line executions has been performed passing through each of both situation where failures occurred, keeping the particle set size to $N_P = 50$. This new version of the filter has been demonstrated successful rates of 83% passing through the failure 1 and 100% passing through the failure 2.

6.5 High accuracy 3D localization

Ground truth position data is useful to compare localization methods in terms of accuracy. However, rigorous ground truth data have to be computed by alternative means than those used for the localization approach being investigated, thus implementing a ground truth method in an urban outdoor scenario is not a trivial task. Unfortunately, and due to time and working constraints, field experiments reported in this thesis do not provide ground truth data. However, this section describes a means to compute a high precision localization estimate based on executing off-line the 3D localization process but using logged sensor data instead of on-line sensor data. In order to keep synchrony of sensor data as it was during the acquisition there is a process managing the synchrony of the data published of each sensor, thanks to the registered time stamps of all readings (see details on section 8.4). This process can slow down the off-line time scale by a factor, thus the localization filter can be executed with a large number of particles to find an accurate estimate. Figure 6.16 plots the localization results computed using the sensor data of the experimental session held during 22th June 2010 at UPC Campus, but slowing down the off-line time by a factor of 10 and using $N_P = 1000$ particles for the filter. This position estimate, denoted as $\hat{X}_h$, will be used in section 6.6 to evaluate the real-time performance of the proposed particle filter 3D localization while modifying the particle set size.

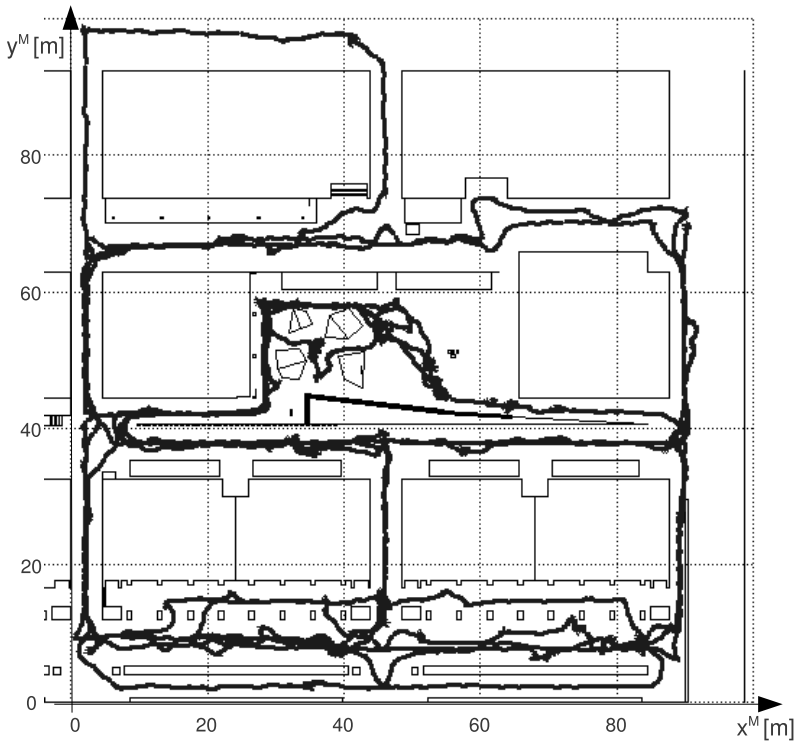


Figure 6.16: High accuracy localization estimate from off-line execution of the filter with $N_P = 1000$ particles. Filter fuses raw data from experimental session on 22th June 2010 at UPC Campus.

6.6 Performance of 3D localization filter

Due to the important role of localization in the autonomous navigation loop (see section 1.2) it is relevant to have a way for a good high-level description of a localization module, specially for system integration purposes. This section provides a set of figures describing three key aspects of the localization performance: output rate, localization error and quality of uncertainty estimate. These figures result from a set of real-time, off-line executions using the first 9 minutes of the data set of the experimental session carried out on 22th June of 2010 at UPC Campus. This subset is a complete tour on the campus, starting at position $(90, 70)m$. For error and uncertainty analysis, the high accuracy localization data presented at section 6.5 has been used as a ground truth, not with the aim of providing absolute error values but to compare the performance when modifying the number of particles of the filter. All figures are given as curves depending on the number of particles, N_P, which takes the following values $60, 70, 80, 90, 100, 110, 120, 130, 150, 175, 200$. The machine used to execute the filtering process during these real-time, off-line, experiments was a desktop computer with two Intel processors Core 2 Duo @ 3.16GHz equipped with a graphics card NVIDIA GeForce 9500 GT. The operating system was Ubuntu 10.04. The robot platform moved at speeds up to $1m/s$. In the following figures, plotted data points are the mean over five executions, and bars indicate the maximum and minimum values over these five executions.

Output rate For a given particle filter, output rate strongly depends on the performance of the machine that executes the filter. However, it is relevant to provide this figure for two reasons: first, to understand the overall shape of the curve, and, second, to have absolute values of the rate at which the filter iterated. Knowing the filter rate will help on the analysis of the following figures. Figure 6.17 shows the localization filter rate as a function of N_P.

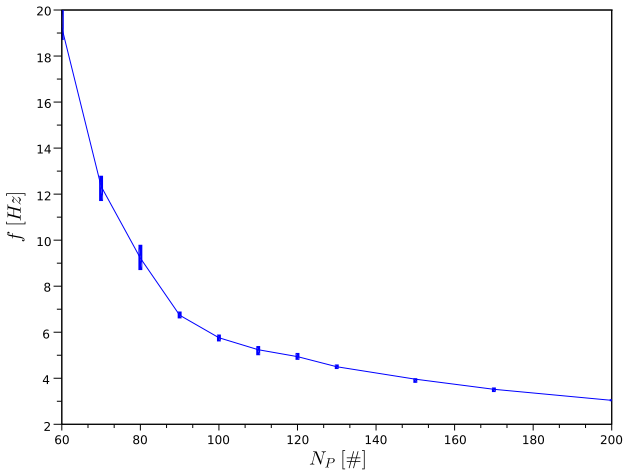


Figure 6.17: Localization filter rate as a function of the number of particles.

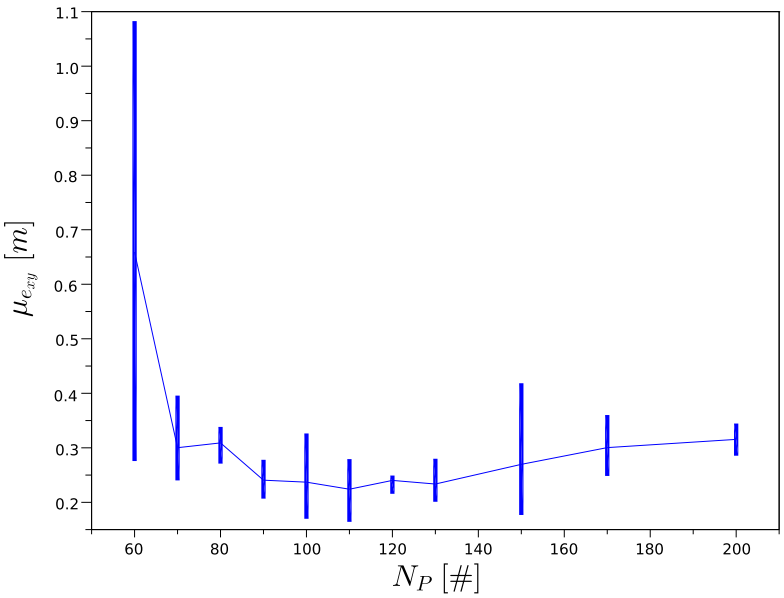


Figure 6.18: Mean xy error of the localization estimate as a function of the number of particles.

Localization accuracy Figure 6.18 shows the mean of the xy error, μe_{xy}, over the whole execution, while incrementing the number of particles of the real-time localization filter. The xy error, e_{xy}, is computed as the euclidean distance between real-time filter estimate and the high accuracy localization estimate, both projected on the xy plane.

Figure 6.19 plots the mean of heading, pitch and roll errors over the whole execution, while incrementing the particle set size. These errors are calculated as the angular difference between real-time filter estimate and high accuracy estimate of either component.

Quality of uncertainty estimate In addition to evaluate the robot position error, it is also interesting to evaluate and provide results on the uncertainty estimate. To derive the following plots, after particle correction in the filter loop, uncertainty is parameterized as a matrix representing the covariance values of the particle set. At iteration t, the confidence level where the error e^t is encountered, taking into account the estimated covariance C_r^t, is:

$$K^t = (e^t)' \hat{C}_r^t e^t \tag{6.24}$$

After the filter execution, the idea is to show the ratio of iterations in which the error is inside the estimated confidence level of $K = 1$. Figure 6.20 illustrates the confidence level concept on the XY plane, while figure 6.21 provides results of this ratio for xy components together and also for θ, ϕ, ψ components separated.

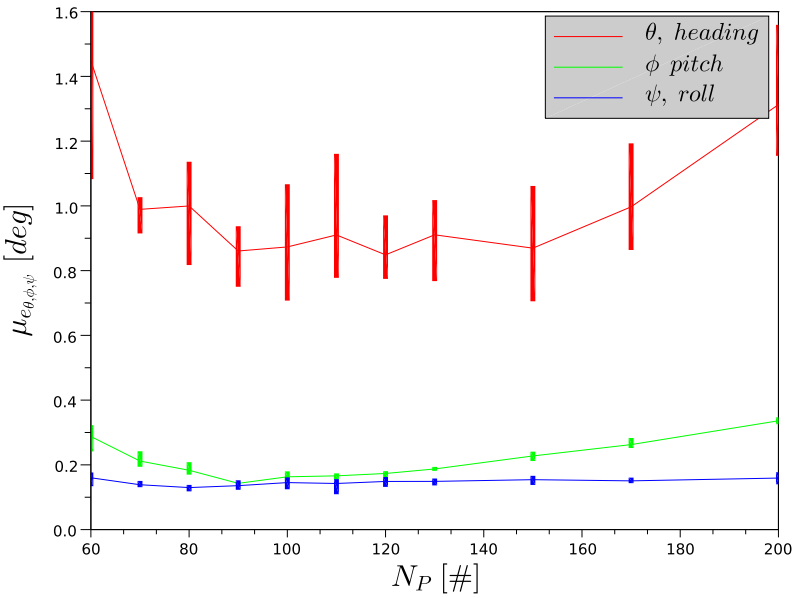


Figure 6.19: Mean θ, ϕ, ψ errors of the localization estimate as a function of the number of particles.

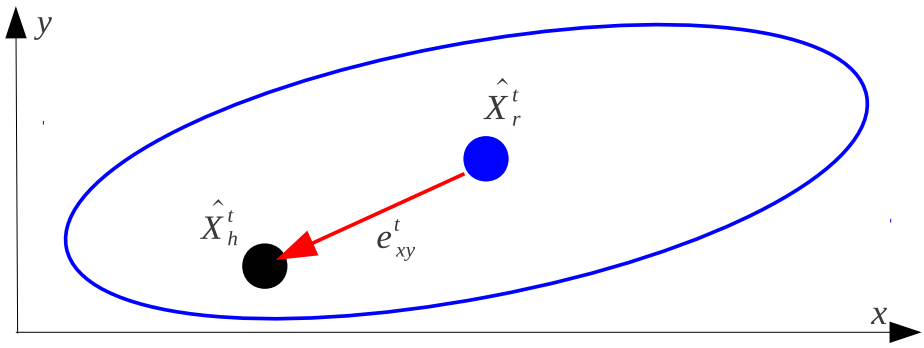


Figure 6.20: K^{th} confidence level. Filter estimate in blue, high accuracy estimate in black and error vector in red (all drawn on the xy plane). Estimated covariance ellipse for a given confidence level K is drawn in blue. In the depicted case (iteration t) the error is inside that K^{th} confidence level.

6.6.1 Discussion

Previous figures presented in this section show how the particle filter presents a fuzzy-defined range on N_P where both xy and θ, ϕ, ψ errors are minimal. For the particular experimental conditions such as the performance of the machine used to execute the filter, the acquisition rates of the used sensors and the platform speeds, this range is centered approximatively at $N_P = 110$. However,

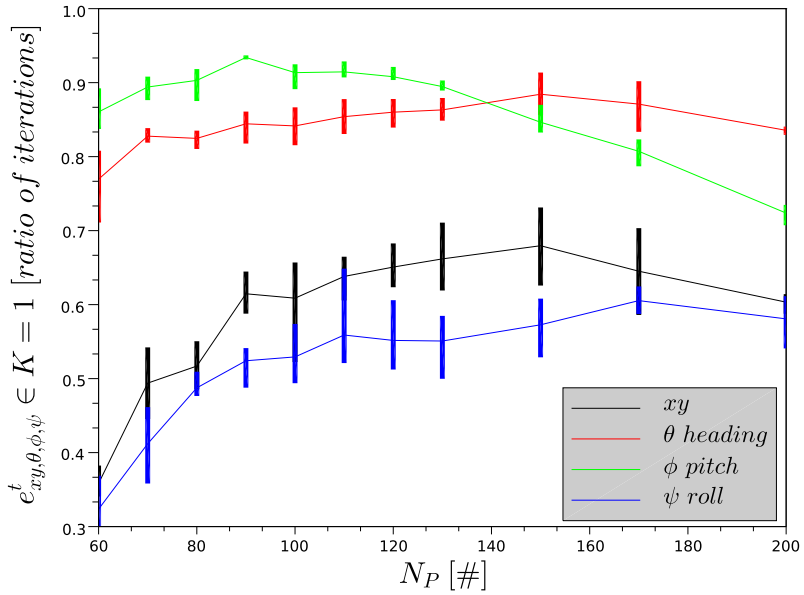


Figure 6.21: Ratio of iterations at which xy, heading, pitch and roll errors are inside a confidence level of $K = 1$.

real-time particle filters present intrinsic characteristics that cause the presence of optimal N_P, so that the following considerations presented below are valid for a generic particle filter localization approach.

For $N_P = 100$, the filter runs at about $6Hz$. This rate coincides with the acquisition rate of the two main sensors used for particle correction (front and back horizontal leuze laser scanners), thus running at this rate implies that no observations will be discarded, as it is done by slower filters such as those with $N_P > 125$ particles. Moreover, running at $6Hz$ does not cause a large error in the kinematic model that implicitly propagates particle states following lineal piecewise displacements. Platform speeds are limited to $1m/s$ and $\pi rad/s$, therefore curve trajectories are well approximated by piecewise displacements computed using increments accumulated during periods of about $165ms$, leading to small error on the state propagation.

The two above mentioned aspects could justify a lower number of particles. However, filters with $N_P < 100$ do not take benefit of their speed since they can not integrate more sensor observations, simply because these observations are not available at higher rates. So, these filters do not integrate more information, and, moreover, they suffer of a less dense sampling of the state space and thus the optimal solution can be usually skipped by the particle set.

These considerations are just presented here as preliminar results in order to motivate further studies that should investigate relations between observation uncertainties, observation rates, vehicle maximum speeds and the optimal number of particles. Section 9.3 dsicusses future works in this line.

Chapter 7

Active Global localization

A fully autonomous robot should be capable not only of tracking its position during navigation but also of solving the global localization problem, that is estimate its position within the map frame with no previous knowledge about it. In the particle filter framework this condition leads to an uniform initialization of the filter through the whole state space. Once the filter iterates, a multi-peak density can arise indicating a multi-hypothesis situation that needs to be disambiguated. This chapter presents an approach to select the best motion action the robot should execute to solve this ambiguous situation, based on probabilistic measures about the expected number of remainig hypotheses after executing the action. The approach is also extended in cooperative situations where robots can receive information from remote observation processes, like a sensor network or other well localized mobile robots, or also when two lost robots select joint actions to solve cooperatively the global localization. This chapter is mainly based on two publications of the thesis author [21, 23].

7.1 Basic Assumptions and Definitions

This section provides the basic assumptions as well as the definitions of concepts that will be used all throughout the chapter to formalise the proposed active strategy. Some of the assumptions are common to other existing global localization approaches, or, may be, they have been already mentioned in previous chapters. However all the assumptions are listed here for the completeness of the chapter. Please note that the points listed below are further discussed in section 7.4.

7.1.1 Basic assumptions

- A sensor network, when it is present in the environment, does not cover all the working area. However, the robots are considered to have always communication coverage, so data communication service is always available. Position of the sensor network devices is known with enough precision relative to the required localization accuracy.

- A sub-system for robot identification and relative localization is assumed to be onboard the robots and also implemented by the sensor network. If a robot is within an area covered by the sensor network or in the line of sight of another robot, the first one can request for remote observations about its position. Examples of such a sub-system can be found in [92, 96, 97].

- A robot can process *real* observations coming either from its onboard exteroceptive sensor readings, from other robots, or from the sensor network server, thanks to an implemented data communication service. Communications are considered always available.

7.1.2 Definitions

- As in chapter 6, a 2D position p in the map frame coordinates is $X_p^m = (x_p^m, y_p^m, \theta_p^m)$. The *true* state of the r^{th} robot is defined by its position on the 2D plane, $X_r^m = (x_r^m, y_r^m, \theta_r^m)$. The *estimated* state of the r^{th} robot is $\hat{X}_r^m = (\hat{x}_r^m, \hat{y}_r^m, \hat{\theta}_r^m)$. The state space is given by $X^m = \{[x_{min}^m, x_{max}^m], [y_{min}^m, y_{max}^m], (-\pi, \pi]\}$. Even if the approach could be scalable to 3D positions, the work presented in this chapter treats the space of 2D positions.

- The approach uses the explicit observation models and likelihoods functions presented in chapter 4 and 6. As it is done in the position tracking case (see chapter 6), these models are used to compute the conditional probability for a real observation o_n^t, given that the robot state is X_p^m, $p(o_n^t|X_p^m)$. This conditional probability is approximated with a likelihood function:

$$p(o_n^t|X_p^m) \sim L_n(o_n^t, o_n^s(X_p^m)) \ \in \ [0, 1] \tag{7.1}$$

- Moreover, this conditional probability can be also computed for an expected observation, instead of for a real observation, thus indicating how distinctive is the position X_q^m to the position X_p^m from the point of view

of the n^{th} observation model. This fact is the core of the herein proposed active strategy and is formally defined as:

$$p(o_n^s(X_q^m)|X_p^m) \sim L_n(o_n^s(X_q^m), o_n^s(X_p^m)) \ \in \ [0,1] \tag{7.2}$$

- $H = \{h_1, \ldots, h_{N_H}\}$ is the set of N_H robot position hypotheses, where the i^{th} hypothesis is defined as a position on the map coordinate frame, $X_{h_i}^m = (x_{h_i}^m, y_{h_i}^m, \theta_{h_i}^m)$, a covariance matrix, C_{h_i}, and a probability associated to that hypothesis such that for the robot position, p_{h_i}:

$$h_i = \{X_{h_i}^m, C_{h_i}, p_{h_i}\}, \ \forall i = 1..N_H; \ \sum_{i=1}^{N_H} p_{h_i} = 1 \tag{7.3}$$

  Different approaches can be found in the literature providing this hypothesis set explicitly [43, 3, 117], or, alternatively, clustering a particle set such as the provided by the particle filter localization methods [110].

With all these assumptions and definitions, the problem to be solved by an active strategy is where to move a lost robot in order to reduce the hypotheses set. The proposed strategy exploits the map and the cooperative environment, selecting actions that drive the robot where *distinctive* observations are expected among the hypotheses. The proposed active approach is formulated in a general way but section 7.4 discusses practical issues when implementing the above listed requeriments in order to obtain the illustrative results of section 7.6.

7.2 Active Strategy. Non Cooperative Environment.

This section formulates the active strategy for the single robot case operating in a non cooperative environment, therefore only observations coming from its own sensors are available. The proposed active strategy is divided in three steps, and only one action can be selected. The first step consists in randomly generating a set of *exploration particles* in the robot coordinate frame, as robot candidate destinations (candidate actions). The second step validates these exploration particles if a *multi-hypothesis path* exists between the robot and the given exploration particle. The third step computes, for each validated exploration particle, the *expected number of remaining hypotheses* given that the robot goes to that exploration particle. The exploration particle, as a position in the robot coordinate frame, with minimum expected number of remaining hypotheses is the selected one to drive the robot.

7.2.1 Generating Exploration Particles

Let's call the k^{th} *exploration particle*, ϵ_k^r, as a random position in the robot coordinate frame generated within a given disk of radius R_ϵ around the robot. R_ϵ is called the *exploration radius*.

$$\epsilon_k^r = X_{\epsilon_k}^r = (x_{\epsilon_k}^r, y_{\epsilon_k}^r, \theta_{\epsilon_k}^r) \tag{7.4}$$

Under the assumption that h_i is the true hypothesis, ϵ_k^r can be expressed in the map coordinates frame as:

$$\epsilon_{ki}^m = \epsilon_k^r | h_i = \begin{bmatrix} x_{h_i}^m \\ y_{h_i}^m \\ \theta_{h_i}^m \end{bmatrix} + \begin{bmatrix} cos(\theta_{h_i}^m) & -sin(\theta_{h_i}^m) & 0 \\ sin(\theta_{h_i}^m) & cos(\theta_{h_i}^m) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\epsilon_k}^r \\ y_{\epsilon_k}^r \\ \theta_{\epsilon_k}^r \end{bmatrix} \tag{7.5}$$

Please note that $\epsilon_{ki}^m \in X^m$ and equation 7.5 shows that a single exploration particle ϵ_k^r becomes a set of N_H positions in the map when it is translated to the map coordinates frame, since all hypotheses should be considered and, therefore, each ϵ_k^r should be translated for each hypothesis $h_i, i = 1..N_H$. Fig. 7.1 illustrates this observation.

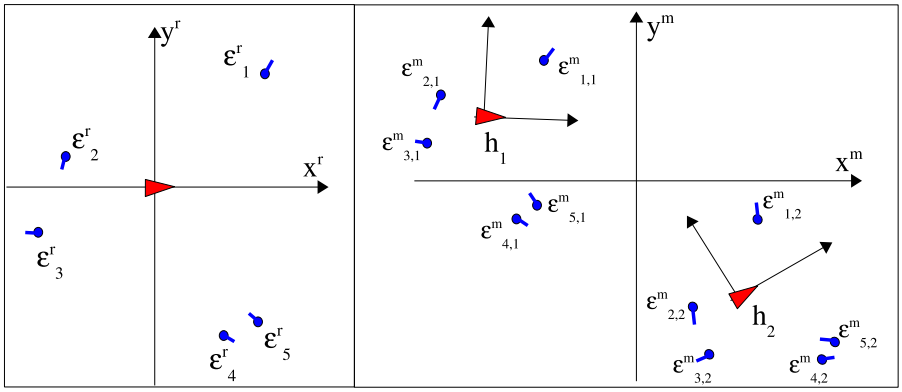


Figure 7.1: A set of 5 exploration particles in the robot coordinates frame (left) and their transformation to map coordinates frame (right) when $N_H = 2$. Each exploration particle is represented as a point with a short line indicating its orientation.

7.2.2 Multi-hypothesis Path Planning

Even if ϵ_k^r is expressed in the robot coordinate frame and, therefore, the robot knows where the exploration particle is positioned, since ϵ_k^r can be beyond the sensor horizons, the existence of a free path between the robot and ϵ_k^r for all hypotheses have to be checked. This step is called *multi-hypothesis path planning* (MHPP), as the planning of a path expressed in the robot coordinate frame using all hypotheses constraints. Figure 7.2 draws the MHPP approach in an illustrative geometric world. For this step, C_{h_i} can be used as a clearance factor. If a multi-hypothesis path (MHP) exists between the robot and the ϵ_k^r, then ϵ_k^r will be labelled as a valid candidate destination, e_k^r, to drive the robot, and this candidate will be add to the set of all valid exploration particles E. Summarizing, the output of the first and second steps of the active strategy will be a set E of N_E exploration particles $E = \{e_1^r \dots e_{N_E}^r\}$ that are connected to the robot with a MHP. This set E is the action set to be evaluated, since each e_k^r is considered as an action *go to* e_k^r. This action set has been automatically generated and it is adapted to the current situation of the robot. Note that E is not a fixed action set such as most of the previous works proposed in

the literature. Please, the reader should remind that an exploration particle is expressed in the robot frame and it can be translated to the map frame if a given hypothesis h_i is assumed as being true:

$$e^m_{ki} = e^r_k|h_i,\ e^m_{ki} \in X^m; \quad \forall k = 1..N_E, \forall i = 1..N_H \tag{7.6}$$

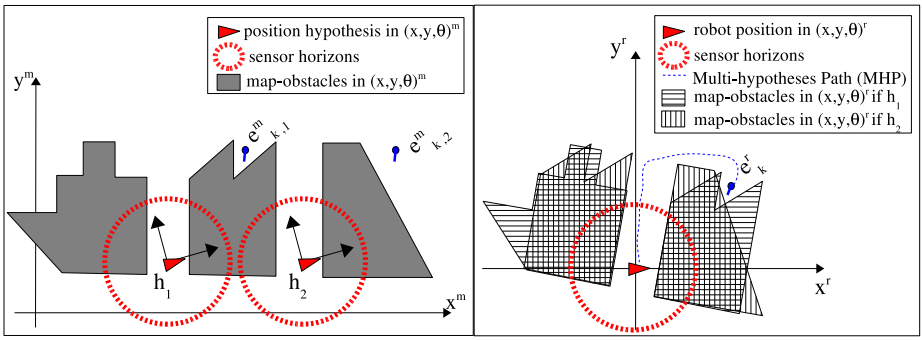


Figure 7.2: Multi-hypotyhesis Path (MHP) in an illustrative geometric world. Map coordinate frame on the left and robot coordinate frame on the right.

7.2.3 Computing Hypotheses Reduction

The goal of this third step is to compute $\hat{N}_H(e^r_k)$, as the expected number of remaining hypotheses given that the robot goes to e^r_k and senses the environment. Prior to compute $\hat{N}_H(e^r_k)$, $\hat{N}_H(e^r_k|h_i)$ will be calculated as the expected number of remaining hypotheses assuming h_i as the true position hypothesis, and given that the robot will execute the action *go to* e^r_k. Using equation 7.2 and considering that only one exteroceptive observation is used ($N_B = 1$), it can be formulated that:

$$\hat{N}_H(e^r_k|h_i) = \sum_{j=1}^{N_H} p(o^s_1(e^m_{kj})|e^m_{ki}) \tag{7.7}$$

If the perception module of the robot provides N_B exteroceptive observations, and independency between them is assumed, equation 7.7 is generalized as:

$$\hat{N}_H(e^r_k|h_i) = \sum_{j=1}^{N_H} \prod_{n=1}^{N_B} p(o^s_n(e^m_{kj})|e^m_{ki}) \tag{7.8}$$

Now, it can be formalized the $\hat{N}_H(e^r_k)$ as the sum of each $\hat{N}_H(e^r_k|h_i)$ weighted by the probability of the i^{th} hypothesis being true, p_{h_i}:

$$\hat{N}_H(e^r_k) = \sum_{i=1}^{N_H} \hat{N}_H(e^r_k|h_i) \cdot p_{h_i} \tag{7.9}$$

Please note that $\hat{N}_H(e^r_k) \in [1, N_H]$ since $p(o^s_n(e^m_{kj})|e^m_{ki}) \in [0, 1]$ as stated in equation 7.2. For an exploration particle e^r_k having similar synthetic observations $\forall h_i$, all the probabilities $p(o^s_n(e^m_{kj})|e^m_{ki})$ will be close to 1 and, therefore,

$\hat{N}_H(e_k^r)|h_i \approx N_H$. Given the assumption of equation 7.3, $\hat{N}_H(e_k^r)$ will also result in $\approx N_H$. This case implies that the position of e_k^r has expected observations too similar for all position hypotheses, and, therefore, it is an exploration particle that will not disambiguate at all the situation. On the other hand, when an exploration particle has completely different synthetic observations $\forall h_i$, the probability $p(o_n^s(e_{kj}^m)|e_{ki}^m)$ will be close to zero $\forall i \neq j$, but it will take one for $i = j$. Again, given the assumption of equation 7.3, $\hat{N}_H(e_k^r) \approx 1$. In this case, the exploration particle e_k^r is expected to completely disambiguate the situation since all synthetic observations are entirely different for each h_i.

With this well delimited results, the $\hat{N}_H(e_k^r)$ can be used as the expected number of remaining hypotheses given that the robot goes to e_k^r, so the robot will start path execution driving itself to the position e_k^r with minimum $\hat{N}_H(e_k^r)$.

7.3 Active Strategy. Cooperative Environment.

This section formulates the previous strategy for a cooperative context in which different robots work in a network robot environment. A network robot environment is formed by a sensor network of N_C sensors and a group of N_R robots. The formulation is presented for the two ways of cooperation: sharing information and selecting joint actions.

7.3.1 Single Lost Robot in a Sensor Network: Sharing Information

This subsection analyses the particular case of a lost robot which is a member of a network robot system. In this situation the active strategy selects, as in section 7.2, one action exploiting its onboard sensors and the map, but also uses the potentialities of integrating remote observations. Let's define the coverage space of the sensor network, which does not depend on time, as:

$$C_{CN} = \bigcup_{c=1}^{N_C} C_c \ , \quad C_{CN} \subset X^m \tag{7.10}$$

where C_c is the coverage area of the c^{th} sensor of the network. The coverage space of the robots, which is time depending, is also defined as:

$$C_{RN}^t = \bigcup_{r=1}^{N_R} C_r^t \ , \quad C_{RN}^t \subset X^m \tag{7.11}$$

where C_r^t is the coverage area of the r^{th} robot at time t. For a lost robot, $C_r^t = \emptyset$.

In the proposed network robot system, both C_{CN} and C_{RN}^t are data available on the central server, since it knows where the sensors are deployed and where the non lost robots are. Note that a robot can request both coverage spaces at a given time and, therefore, a lost robot can use this data for local processing when it is executing the active global localization strategy.

In this context, the active strategy will be the same that the one exposed in section 7.2. Evaluation of actions will be done by equations 7.8 and 7.9, but considering that the robot can use external observations done by other observation process such as a camera network or well localized robots. In equation 7.8, and

in order to consider remote observations for the active strategy, the lost robot has to evaluate if e^m_{kj} is in $C_{CN} \bigcup C^t_{RN}$. If this is the case, a remote observation for that position is available and the $p(o^s_n(e^m_{kj})|e^m_{ki})$ can be computed where o^s_n is the model for that remote observation.

The effect of this is that exploration particles expected to be in the coverage space, $C_{CN} \bigcup C^t_{RN}$, will be attractive to move the robot since disambiguation can be done via remote observations instead of only considering the robot exteroceptive observations. Therefore, this is a situation of an active approach considering the potentialities of a cooperative environment, taking advantage of information sharing.

As an illustrative example, the GPS system can be considered as a particular case of this cooperative context since the GPS satellite network acts as a sensor network. Assuming that we have a map of the GPS coverage in our environment, a lost robot equipped with a GPS receiver, out of satellite coverage, will be attracted by actions driving the robot to areas where GPS is available.

7.3.2 Two Lost Robots: Selecting Joint Actions

This subsection addresses the case where two lost robots are performing the global localization task, hence trying to locate themselves in the environment. In this case the output of the active strategy should be two joint actions, each one to be executed by each robot.

In this scenario two lost robots, r^{th} and ρ^{th}, are assumed to be in line of sight of each other. It is also assumed that they can be detected between them by means of an active or passive beacon system like the ones reported in [92, 96, 97]. Therefore, they can build up a commom frame in the middle point of this line of sight, called the cooperative robot frame and denoted by $(x^{r\rho}, y^{r\rho}, \theta^{r\rho})$(see figure 7.3). From the observation made by the r^{th} robot to the ρ^{th} robot, $(\delta^r_\rho, \alpha^r_\rho, \phi^r_\rho)$, the cooperative frame is placed in terms of the r^{th} robot coordinates as:

$$x^r_{r\rho} = \frac{\delta^r_\rho}{2} cos\alpha^r_\rho; \quad y^r_{r\rho} = \frac{\delta^r_\rho}{2} sin\alpha^r_\rho; \quad \theta^r_{r\rho} = \frac{\phi^r_\rho}{2} \tag{7.12}$$

Initially, the robots translate their own observations to the cooperative frame $(x^{r\rho}, y^{r\rho}, \theta^{r\rho})$ and an hypotheses generation step to localize its center is executed integrating all available observations. Once a set of position hypotheses for

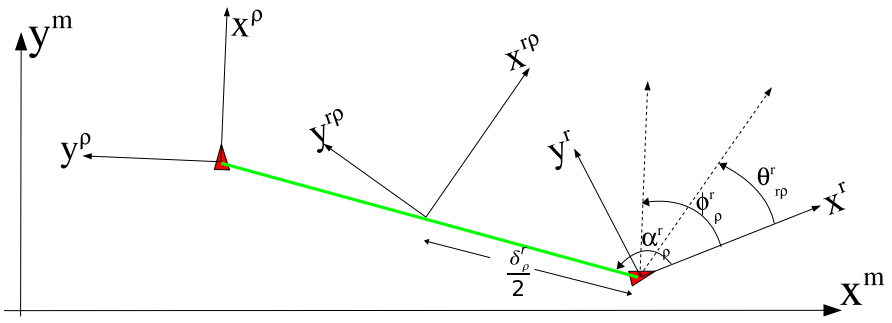


Figure 7.3: Single robot frames (r and ρ) and the cooperative robot frame ($r\rho$).

the cooperative frame is obtained, the active strategy could run as exposed previously and the robots could perform the two best actions, a different action for each robot. However, this approach is not suitable since this couple of actions could disambiguate the same subset of hypotheses and, therefore, the two decided actions would be redundant. Instead, it would be desirable that the robots take complementary actions, complementary in the sense that each action disambiguates a different subset of hypotheses. Previous strategy of estimating $\hat{N}_H(e_k^r)$ gave a *quantitative* criterion but for the multi-action problem it is also needed a *qualitative* criterion to evaluate the actions.

To formulate a qualitative criterion, a N_H-dimensional vector is defined, $V(e_k^{r\rho})$, for each exploration particle in the cooperative robot frame. The i^{th} component of such vector is defined using the equation 7.8 as:

$$V(e_k^{r\rho})_i = \hat{N}_H(e_k^{r\rho}|h_i) - 1,\ \forall i = 1 \dots N_H,\ \forall k = 1 \dots N_E \tag{7.13}$$

This vector indicates at its i^{th} component how the k^{th} exploration particle resolves the hypothesis h_i. Substraction of 1 removes the contribution of $i = j$ of equation 7.8, which is always 1, and leads to a more sparse vector set, more suitable for computations presented below. A vector defined by the above equation will have the following properties inherited from properties of the likelihood function:

- $V(e_k^{r\rho})_i \in [0, N_H - 1]\ ,\ \forall i = 1..N_H,\ \forall k = 1..N_E$
- The ideal exploration particle is that $e_k^{r\rho}$ which fully disambiguates the situation. It has a vector $V(e_k^{r\rho}) = [0]$.
- An useless exploration particle has completely ambiguous synthetic observations. It has a vector $V(e_k^{r\rho}) = [N_H - 1]$.

A weighted scalar product between two of these vectors is defined as:

$$< V(e_k^{r\rho}), V(e_q^{r\rho}) >= \sum_{i=1}^{N_H} V(e_k^{r\rho})_i \cdot V(e_q^{r\rho})_i \cdot p_{h_i},\quad \forall k, q = 1 \dots N_E \tag{7.14}$$

This weighted scalar product provides a measure of the complementariness of these two exploration particles. The scalar product will be maximized when two particles disambiguate the same subset of hypotheses and, therefore, their vectors will be colinear. Otherwise, when two exploration particles disambiguate different subsets of hypotheses, $V(e_k^{r\rho})$ and $V(e_q^{r\rho})$ are close to be orthogonal and the scalar product approaches to zero.

For this case, the strategy initially selects the best exploration particle in the sense of minimum $\hat{N}_H(e_k^{r\rho})$, as it was done for the single robot case, which will be called the primary one, $e_{s1}^{r\rho}$. Then, the strategy will search a support action, that is, going to the exploration particle $e_k^{r\rho}$ minimizing the weighted scalar product with $e_{s1}^{r\rho}$. This support exploration particle will be labelled as $e_{s2}^{r\rho}$. Finally, $e_{s1}^{r\rho}$ and $e_{s2}^{r\rho}$ have to be executed by the robots so it is necessary to compute the translation of both actions from the cooperative robot frame $(x, y, \theta)^{r\rho}$ to each robot frame $(x, y, \theta)^r$ and $(x, y, \theta)^\rho$.

7.4 Implementation of the Active Strategy

This section details the implementation of the proposed active strategy, which is intended to be independent from the origin of data, whether coming from a real platform and sensors or from a simulated environment.

7.4.1 Environment Model

The environment model, $\mathcal{M}$, used to implement the active strategy has been that described in section 4.2.1. However it is noteworthy to remark that the proposed active method only requires an environment model that expected observations could be calculated given robot positions.

7.4.2 Observation Models

For the active strategy we take into account a laser scanner, an electronic compass and a set of omnidirectional cameras deployed on the environment. The relative localization between robots is only considered to build the cooperative frame for the two lost robots case. This subsection describes the observation models used to compute the expected observation $o_n^s(X_p^m)$. These models are used to compute the $p(o_n^s(e_{kj}^m)|e_{ki}^m)$ of equation 7.8.

The observation model for the **laser scanner**, $o_L^s(X_p^m)$, is a vector of $N_L = 133$ ranges over the scan aperture of $(-95, 95)$ degrees. The maximun laser range is limited to $r_{max} = 15m$. To compute this observation model, we compute the ray tracing function from the position X_p^m, as described in section 4.2.3. Summarizing, the output of the observation model of the laser scanner is, given the X_p^m position:

$$o_L^s(X_p^m) = (o_{L,1}^s(X_p^m)...o_{L,N_L}^s(X_p^m));\ o_{L,i}^s(X_p^m) = rayTracing(X_p^m, \mathcal{M}, i) \tag{7.15}$$

Expected observations for the **compass** are much more simple to compute. Given a position X_p^m,the observation model, $o_C^s(X_p^m)$, is directly the heading of that given position:

$$o_C^s(X_p^m) = \theta_p^m, \quad \in (-\pi, \pi] \tag{7.16}$$

The **sensor network** is modelled as a set of N_C omnidirectional cameras deployed at known positions of the environment. The implemented observation model, $o_N^s(X_p^m)$, outputs a triplet containing range, bearing and heading measures (see figure 7.4). The coverage area for the c^{th} camera, C_c, is modelled as the set of positions of the state space that are in a line of sight of length less than $R_C = 7m$ with the position of the c^{th} camera. Given a position X_p^m, if $X_p^m \in C_c$ the observation model is computed as:

$$o_N^s(X_p^m) = \begin{bmatrix} \delta_{c,p} \\ \alpha_{c,p} \\ \phi_{c,p} \end{bmatrix} = \begin{bmatrix} \sqrt{(x_p^m - x_c^m)^2 + (y_p^m - y_c^m)^2} \\ atan(\frac{y_p^m - y_c^m}{x_p^m - x_c^m}) \\ \theta_p^m \end{bmatrix} \tag{7.17}$$

otherwise, when $X_p^m \notin C_c$, $\forall c = 1..N_C$ the output of the model is $o_N^s(X_p^m) = (-1, NaN, NaN)$, indicating that the position is not seen by the camera network.

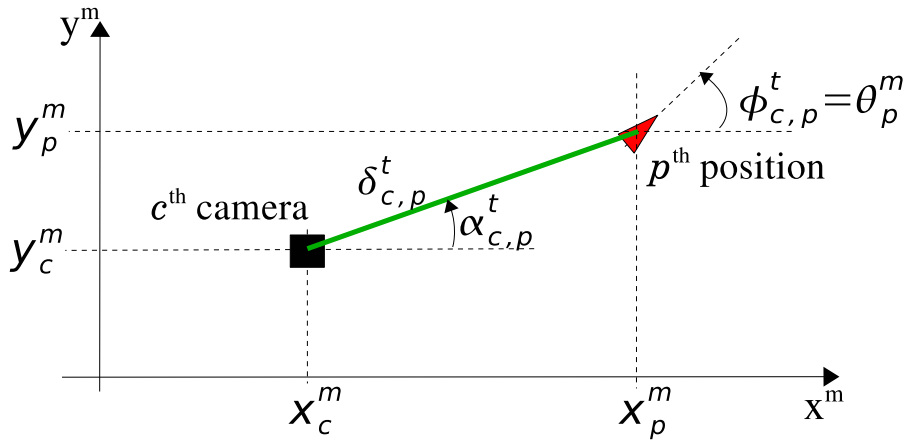


Figure 7.4: The observation model for the c^{th} camera seeing the position X_p^m.

This implementation does not integrate potential **relative localization** between robots in equation 7.8 as observations. This could be done in a similar way such that implemented by the cameras since a well localized robot can be considered as an static camera of the camera network.

7.4.3 Likelihood Functions

In order to compute the conditional probabilities of equation 7.8, and as stated in equations 7.1 and 7.2, we need to implement likelihood functions $L_n(\cdot)$ for each of the above mentioned observations models ($n = L, C, N$). All the implemented $L_n(\cdot)$ are based on the complementary error function $erfc()$, already presented and used in chapter 6. In equations presented below, these likelihoods are written as functions between a real and an expected observation but, as discussed in equation 7.2, they can be also computed for two expected observations.

The likelihood function for the laser observations, where $\sigma_L = 0.05m$, is:

$$L_L(o_L^t, o_L^s(X_p^m)) = \frac{1}{N_L}\sum_{i=1}^{N_L} erfc(\frac{|o_{L,i}^t - o_{L,i}^s(X_p^m)|}{\sigma_L\sqrt{2}}) \tag{7.18}$$

For the compass observations, the $L_C(\cdot)$, where $\sigma_C = 0.08rad$, is:

$$L_C(o_C^t, o_C^s(X_p^m)) = erfc(\frac{|o_C^t - o_C^s(X_p^m)|}{\sigma_C\sqrt{2}}) \tag{7.19}$$

The likelihood function for the camera network observation, where $\sigma_{C,1} = 0.5m$, $\sigma_{C,2} = 0.05rad$ and $\sigma_{C,3} = 0.1rad$, is:

$$L_N(o_N^t, o_N^s(X_p^m)) = \frac{1}{3}\sum_{i=1}^{3} erfc(\frac{|o_{N,i}^t - o_{N,i}^s(X_p^m)|}{\sigma_{N,i}\sqrt{2}}) \tag{7.20}$$

For the $L_n(\cdot)$ involving two synthetic observations $o_n^s(e_{ki}^m)$ and $o_n^s(e_{kj}^m)$, parameters σ_n can be calculated using the matrices C_{h_i} and C_{h_j}, $\forall i, j, k, n$.

7.4.4 Hypotheses Generation: Particle Filtering and Clustering

A particle filter has been implemented to perform an initial search as previous step of the active strategy, following the well stablished framework of [110] (see chapter 5 for further details). The filter represents the position belief with a set of N_P particles, $s_i^t = \{X_i^{m,t}, w_i^t\}, \forall i = 1..N_P$. It initializes generating the set of position particles sampling randomly the $(x, y)^m$ space, but sampling the θ^m dimension with a normal law of $\mathcal{N}(\theta^0, \sigma_{\theta^0})$, being θ^0 the first available compass observation. Weights of each particle are initialized to $w_i = 1/N_P$. A propagation step is performed to move each particle's state, using the odometric obervation o_0^t and the kinematic model $f(\cdot)$. The above described likelihood functions integrate real observations to correct the weights of each particle position as:

$$w_i^t = \prod_{n=1}^{N_B} L_n(o_n^t, o_n^s(X_i^{m,t})) \ , \ \forall i = 1..N_P \ , \ \forall t > 0 \tag{7.21}$$

The resampling step generates a new particle set sampling the old one according to the particle weights, so likely areas are successively more sampled. Section 5.2.1 provides details on the resampling method. To perform experiments, $N_P = 5000$ particles have been used. After several iterations, particles are concentrated in several subsets, so a clustering step is executed in order to generate a reduced hypotheses set H. Clustering is implemented using a recursive routine that starts with the set of position particles ordered by their weights $w_{s_i} \geq w_{s_j}$ when $i < j$. Let K_k be a cluster and $c(K_k)$ be the centroid of it. Initially, the routine creates the first cluster using the first particle $c(K_1) = X_{s_1}^m$. The rest of the particles will join to an already created cluster if $d(X_{s_i}^m, c(K_k)) < R_K$ or, otherwise, will create a new cluster. R_K is the parameter fixing clustering size, set to $R_K = 3m$. Each time that a particle joins to an already created cluster, the centroid is updated, using all the particles in that cluster, as a weighted mean of their positions. The fact that, in a particle filter, the more likely particles are usually at the center of the clusters improves the performance of this simple method. Finally, a covariance matrix is computed with the particle subset of each cluster, and clusters become the position hypotheses, so the system is ready to perform the active strategy.

7.4.5 Multi-hypothesis Path Planning

The Rapidly-Exploring Random Trees (RRT) approach [53] has been implemented. In the case of the multi-hypothesis path planning, the tree is computed in the robot coordinate frame translating the map obstacles to this frame for each hypotheses in a similar way as equation 7.5 does. When planning paths in the robot coordinates to reach e_k^r goals, randomly points that build iteratively the tree are generated in the surroundings of the robot, instead of on the whole map in order to improve the efficiency of the RRT. Details on the implementation of RRT's with that environment model can be found in [19]. However, we have seen that generation and validation of the exploration particles can be collapsed in a single step by means of building a single RRT bounded in the exploration area. The RRT is also generated in the robot frame, taking into account the constraints of all hypotheses. The nodes of the RRT will be

directly the exploration particles. This computes a single but 'big' RRT just once, avoiding the computation of a RRT to validate each exploration particle. Exploration radius R_ϵ is set to $20m$.

7.5 Comparative Analysis on Computational Complexity

This section discusses the computational cost of the proposed active approach, both in time and memory, and compares it to that of the existing entropy-based methods [34, 91]. In order to compare different methods, the notation N_A is introduced, indicating the number of actions to be evaluated, which in the proposed method, [23], coincides with N_E since each exploration particle supposes an action.

As equations 7.8 and 7.9 suggest, the time complexity to evaluate a single action in the proposed active strategy is $\mathcal{O}(N_H^2 \cdot N_B)$. Therefore, the time complexity of evaluating a set of N_A actions results on $\mathcal{O}(N_H^2 \cdot N_A \cdot N_B)$. In the particular case of the implementation presented in section 7.4, and, since the observations are computed on-line, N_B becomes $N_L + N_C$, which refers to the laser scanner number of points and the number of cameras respectively. In terms of memory complexity, the presented implementation is extremely efficient since the spatial representation is based on the compact GIS vector format and no sensor-appearance data is stored in the map database, thus avoiding space discretization and huge representations. The memory complexity of this spatial representation has not been analyzed in this work but the real environment of about $10.000m^2$, used in this chapter as a testbench area, is represented with a map of about $40KBytes$, supposing a very efficient map model ($4Bytes/m^2$).

For the work in [34], based on the Markov framework, time complexity behaves as $\mathcal{O}(N_X^2 \cdot N_A \cdot N_S)$, where N_X is the number of all possible position states, N_A the size of the action set and N_S the number of sensings at each state. In order to reduce the computational cost, authors precompute the sensor model and cluster the belief distribution, forming a set of N_{Xg} Gaussians in a runtime step, reducing time complexity to $\mathcal{O}(N_X \cdot N_{Xg} \cdot N_A)$, with $N_{Xg} \ll N_X$. This clustering step is similar to that performed by the proposed strategy in the sense of creating a set of Gaussians instead of having a complete sampled belief distribution. Therefore, it can be supposed that $N_{Xg} \sim N_H$. However, the term N_X remains in the time complexity expression for this approach. Due to the complete discretization of the state space, N_X grows up with the environment size and this approach remains too expensive in large areas.

Using the same entropy-based approach but based on the particle representation of the uncertainty, the work presented on [91] has a time complexity of $\mathcal{O}(N_P^2 \cdot N_A \cdot N_J)$, where N_P is the number of particles representing the belief, N_A the number of actions to be evaluated, and N_J an observation model parameter. Authors precompute the observation model, reducing the time complexity to $\mathcal{O}(N_P^2 \cdot N_A)$ but incrementing the memory complexity since the precomputations have to be stored in an appearance map. Since $N_P \ll N_X$ is a general case, this work drastically reduces the time complexity in comparison with [34]. However, the complexity remains high, specially for large environments where the amount of particles need to global localize the robot is also large. In the

practical experimentation, authors in [91] report the requirement to reduce the action set and the size of the environment in order to achieve acceptable computation delays.

Table 7.1 summarizes this discussion. Theoretical time complexities of the considered frameworks are of the form of $\mathcal{O}(N^2_{X,P,H} \cdot N_A \cdot N_{S,J,B})$. Therefore the quadratical terms N_X^2, N_P^2, N_H^2 are the critical ones to be analyzed. In large environments, such as the one of $10.000m^2$ used as a test bench of this experiment, a complete discretization, with discretization steps of $\Delta xy = 0.5m$, and $\Delta\theta = 5^\circ$, would result in $N_X \sim 3 \cdot 10^6$ states. In the proposed implementation described in section 7.4, the particle filter localization needs about $N_P \sim 5000$ particles. Several executions of the particle filter with the clustering step have resulted in a number of hypotyheses of about $N_H \sim 20$ in the testbench environment, thus $N_H \ll N_P \ll N_X$ will be generally satisfied, indicating that the presented approach entails a significant improvement in time complexity, a key requeriment in large environments. For the practical implementation it exists a trade-off between pre-computation of observation models, which increases memory complexity, versus on-line computation of these models, which increases the time complexity. In the proposed implementation we have choosen to compute on-line the observation models. This choice was motivated by the fact that memory resources required to store precomputed values of these models grows up with the number of states N_X, which increases with the environment size, therefore being critical in large evironments.

Table 7.1: Comparison of computational complexities between existing active methods

	Theoretical Time Complexity	Practical Time Complexity	Practical Memory Complexity
[34]	$\mathcal{O}(N_X^2 \cdot N_A \cdot N_S)$	$\mathcal{O}(N_X \cdot N_{Xg} \cdot N_A)$	$\mathcal{O}(N_X \cdot N_S)$
[91]	$\mathcal{O}(N_P^2 \cdot N_A \cdot N_J)$	$\mathcal{O}(N_P^2 \cdot N_A)$	$\mathcal{O}(N_X \cdot N_J)$
[23]	$\mathcal{O}(N_H^2 \cdot N_A \cdot N_O)$	$\mathcal{O}(N_H^2 \cdot N_A \cdot (N_{LS} + N_C))$	$(\sim 4Bytes/m^2)$

So, the proposed technique offers better computational performance with respect to existing ones, specially thanks to the clustering step, thus it can be considered more appropiate in large environments where N_X and N_P become large magnitudes. Obviously, the environment size will also influence in N_H, but the relation $N_H \ll N_P \ll N_X$ will be accomplished in most common urban environments. For illustrative purposes, some numerical results about the real delays obtained when computing the active strategy are provided. A run of the active stategy, with $N_H = 19$ position hypotheses and $N_E = 40$ exploration particles, needs about 87 seconds in a standard 2GB RAM, 1.86-GHz Core 2 Duo PC, running Linux Ubuntu kernel 2.6.17. Only the 1.8% of this delay is for the generation of the E set, which involves multi-hypothesis path planning. The 97% of this execution time has been to compute the expected observations, $o_L^s(e_{kj}^m)$ and $o_L^s(e_{ki}^m)$ of the laser scanner. Authors are confident that the proposed implementation can be fairly optimized to reduce these delays, for instance using optimized techniques to compute range observation models such as [24], also described in section 4.3 of this thesis.

7.6 Simulation Results

This section presents results on the action selection for both single robot and cooperative cases. First subsection details how each real observation is obtained in the simulated environment used to obtain the results. More details on the simulator or on the programming can be found in chapter 8.

7.6.1 Simulated Real Observations

Robot onboard sensors and other sensors and robots of the network provide real observations, o_n^t, computed given a robot position, to be integrated by the localization algorithm, in the hypotheses generation step. Even if these observations are simulated, they are labelled as *real* since they play the role of actual observation data.

Each platform is equipped with **wheel encoders** that provide, at iteration t, the odometric observation o_0^t. This observation is composed by the increment in translational motion, ΔR^t, and the increment in rotational motion, $\Delta\theta^t$. We add to these increments a simulated normal noise with standard deviation of 5% in translation and 10% in rotation:

$$o_0^t = (\Delta R^t + \mathcal{N}(0, 0.05 * \Delta R^t), \Delta\theta^t + \mathcal{N}(0, 0.1 * \Delta\theta^t)) \tag{7.22}$$

where,

$$\Delta R^t = \sqrt{(x_r^{m,t} - x_r^{m,t-1})^2 + (y_r^{m,t} - y_r^{m,t-1})^2}; \quad \Delta\theta^t = \theta_r^{m,t} - \theta_r^{m,t-1} \tag{7.23}$$

Each simulated robot has also a simulated **laser scanner** RS4 (Leuze corp.), providing, at iteration t, a real observation o_L^t; computed from the simulated robot position, $X_r^{m,t}$, following the model described in the section 7.4. However, to simulate a real observation, we add normal noise with $5cm$ standard deviation.

$$o_L^t = (o_{L,1}^t, .., o_{L,N_L}^t); \quad o_{L,i}^t = rayTracing(X_r^{m,t}, \mathcal{M}, i) + \mathcal{N}(0, 0.05) \tag{7.24}$$

An **electronic compass** TCM2 (PNI corp.) is also simulated to be onboard of each robot and provides, at iteration t, a real observation o_C^t. This observation is directly computed as the heading of the simulated robot position with a normal noise of standard deviation of $0.05rad$:

$$o_C^t = \theta_r^{m,t} + \mathcal{N}(0, 0.05) \tag{7.25}$$

In the simulator, compass observations are always available. However, compass measurements are usually corrupted by magnetic distortions. For the local distortions, the TCM2 device provides an automatic routine to calibrate itself in a given position on the robot. For the external distortions, this device has a magnetic alarm to detect corrupted readings.

The sensor network is modelled as a set of N_C **omnidirectional cameras** deployed at known positions. This camera network provides to the robot, at iteration t, the observation o_N^t. The model to compute real observations of the camera network is the same than the one exposed in section 7.4, but adding normal noise to the measurements. If a robot is in the coverage area of the c^{th}

camera, $X_r^m \in C_c$, a real observation of the camera network is available as:

$$o_3^t = \begin{bmatrix} \delta_{c,r}^t \\ \alpha_{c,r}^t \\ \phi_{c,r}^t \end{bmatrix} = \begin{bmatrix} \sqrt{(x_r^{m,t} - x_c)^2 + (y_r^{m,t} - y_c)^2} + \mathcal{N}(0, 0.5)) \\ atan(\frac{y_r^{m,t} - y_c}{x_r^{m,t} - x_c}) + \mathcal{N}(0, 0.02 * \delta_{c,r}^t) \\ \theta_r^{m,t} + \mathcal{N}(0, 0.2) \end{bmatrix} \tag{7.26}$$

When the ρ^{th} robot sees the r^{th} one in a line of sight of length less than $R_R = 10m$, the observation made by ρ at time t, in terms of the ρ^{th} frame, is:

$$o_4^t = \begin{bmatrix} \delta_r^{\rho,t} \\ \alpha_r^{\rho,t} \\ \phi_r^{\rho,t} \end{bmatrix} = \begin{bmatrix} \sqrt{(x_r^{m,t} - x_\rho^{m,t})^2 + (y_r^{m,t} - y_\rho^{m,t})^2} + \mathcal{N}(0, 0.05)) \\ atan(\frac{y_r^{m,t} - y_\rho^{m,t}}{x_r^{m,t} - x_\rho^{m,t}}) + \mathcal{N}(0, 0.01 * \delta_r^{\rho,t}) \\ \theta_r^{m,t} - \theta_\rho^{m,t} + \mathcal{N}(0, 0.1) \end{bmatrix} \tag{7.27}$$

7.6.2 Single Lost Robot. Non Cooperative *vs* Cooperative Environments

In this section we present the results of the active strategy for the single lost robot case, comparing the non cooperative environment of section 7.2 with the cooperative environment of subsection 7.3.1. The methodology has been as follows: the robot is placed in a given position and the particle filter run using only robot onboard sensors. For the presented run, the clustering step has generated a set H of $N_H = 19$ position hypotheses showed in figure 7.5. Afterwards, an exploration particle set, E, is generated for both cases, obtaining a common set of $N_E = 40$ exploration particles, $\{e_1^r \dots e_{40}^r\}$. The evaluation of each exploration particle is performed separately, that is, in the non cooperative case, only the onboard sensors are considered in equation 7.8, while in the cooperative case also the camera network has been considered to evaluate the actions.

Figure 7.6 shows the E set of $N_E = 40$ exploration particles in the robot frame. It can be noted how the exploration set has been adapted to the multi-hypotheses constraints, thanks to the multi-hypotheses path planning step. The figure also shows, marked with a blue cross, the six best actions for the non cooperative case, and, with a red square, the six best actions for the cooperative case. The reader can overlap the robot frame of this figure with the depicted robot frame on figure 7.5 in order to imagine where the robot will arrive if it goes to a given exploration particle. For the non cooperative case, the best actions are going to places which are distinctive from the laser scanner observation point of view. These six best actions are mainly related in going down of the corridor (see figure 7.5), since laser scanner is expected to take less ambiguous observations. On the other hand, for the cooperative case, the six best actions are going up of the corridor, since this is the area where more camera detections are expected taking into account all the hypotheses.

Figure 7.7 shows the value of each $\hat{N}(e_k^r)$, $\forall k = 1 \dots 40$, for both, non cooperative and cooperative cases. This figure shows how the expected number of hypotheses is always bounded to $[1, N_H]$. It can be observed that, for the cooperative case, the expected reduction of hypotheses is more significant that the obtained for the non cooperative environment. Therefore, some particles have the same $\hat{N}(e_k^r)$, indicating that no camera detection was expected even though considering all hypotheses, thus the onboard sensors remain the only

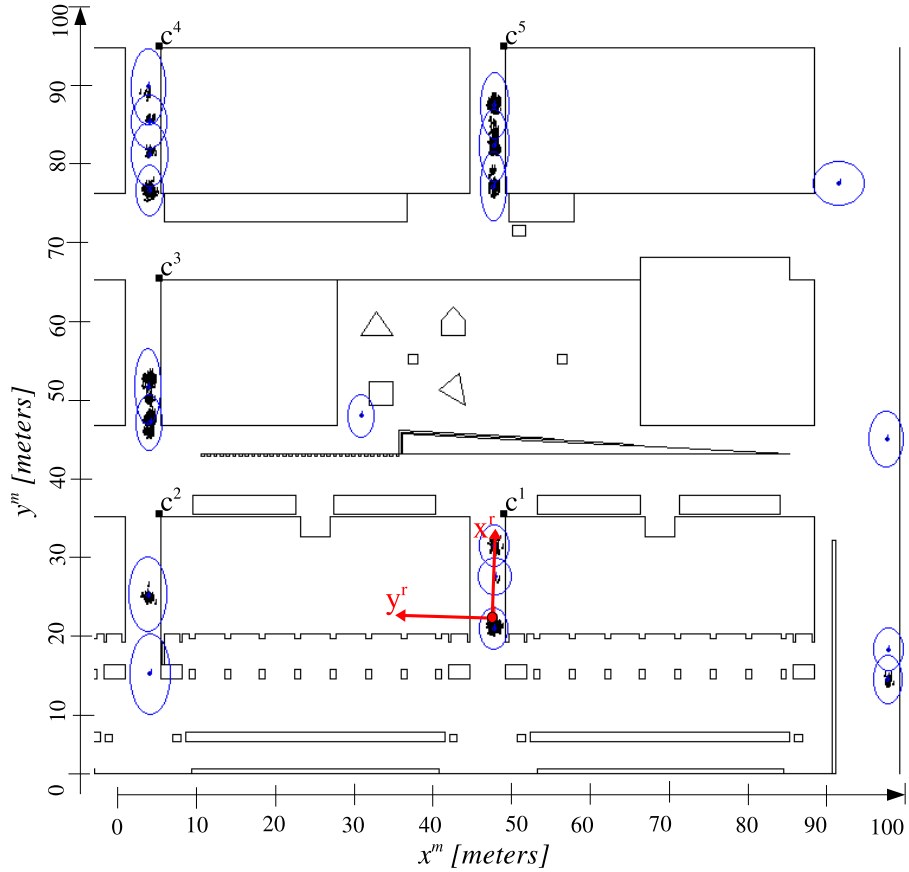


Figure 7.5: The set H of $N_H = 19$ position hypotheses on the map. Each hypothesis is marked as a blue ellipse. Please note also the red dot marking the true robot position X_r^m, the robot frame (x^r, y^r) and the position of the $N_C = 5$ cameras.

means to disambiguate the situation. However, some other actions take benefit from the potential remote observations of the camera network and reduce clearly their evaluation index.

7.6.3 Cooperative Environment. Two Lost Robots

This section presents the results for the two lost robots case studied in subsection 7.3.2. In order to better evaluate this case, only the laser scanners of the robots are considered in the active strategy, but there are no limitations on integrating remote observations of a camera network. We have proceeded placing two robots in the environment in positions within a line of sight. Robots first have performed a relative localization observation between them and, then, they have built the cooperative frame. Afterwards, the hypotheses generation step is performed with the particle filter, using the observations provided by the compasses and the laser scanners of each robot. For this case the particles s_i represent positions of the cooperative frame and this fact implies that each par-

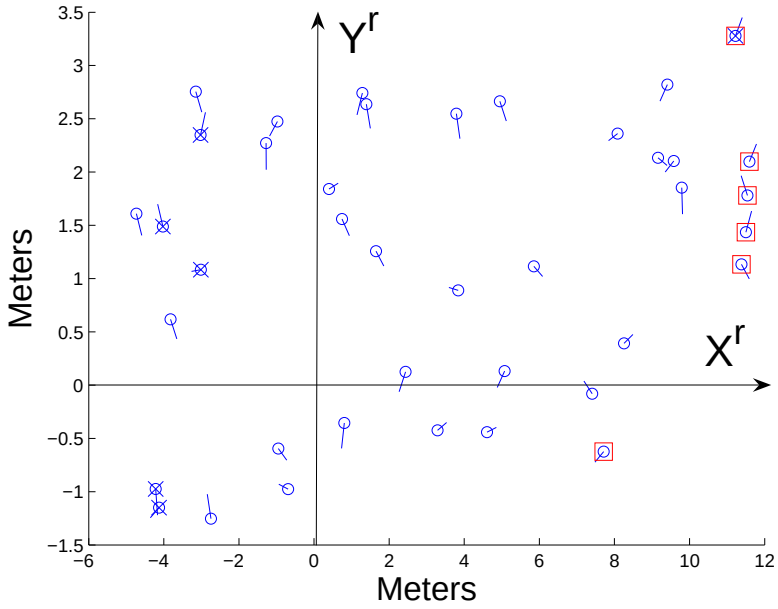


Figure 7.6: The set E of $N_E = 40$ exploration particles that has been evaluated for both cases. The six best actions for the non cooperative case are marked with a blue cross and the six best ones for the cooperative case with a red square.

ticle should satisfy a line of sight constraint imposed by the relative localization. Moreover, when position particles have to be corrected with real observations (see equation 7.21), they have to be translated to the relative positions of each observation process given the observed line of sight. Thanks to this line of sight constraint and to the integration of more observations taken from two diferent points of view, the hypotheses generation step is improved and a few number of hypotheses generated. In the experiment presented in figure 7.8, where the two robots are placed at the same corridor as the one in figure 7.5, the number of generated hypotheses has been $N_H = 11$. The two lost robots case entails, at this initial step, an improvement in the hypotheses generation which implies less computational efforts to perform the active strategy, since time complexity depends on N_H^2.

Once the set H has been cooperatively generated, the active strategy is executed and outputs two joint actions, one to be performed by each robot. Figure 7.9 shows the E set generated in the presented execution and marks the selected primary action with a green square, the five best support actions considering the quantitative criterion with red diamonds and the five best support actions considering the qualitative criterion with green crosses. The quantitative criterion would select redundant actions in most of the cases while the qualitative one clearly selects complementary actions.

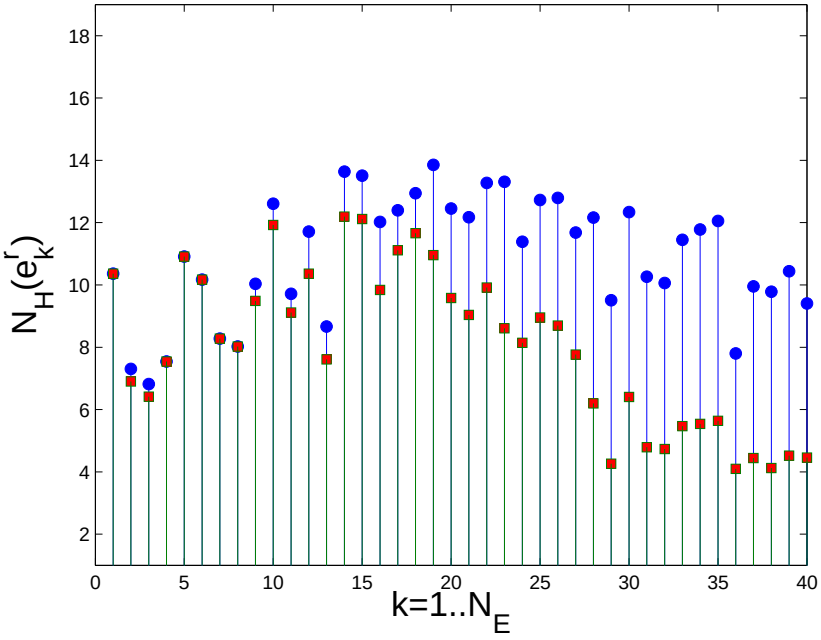


Figure 7.7: The $\hat{N}(e_k^r)$ values of the exploration set E for both the non cooperative (blue dots) and the cooperative (red squares) cases. Particles are sort by its x^r coordinate in order to be related with figure 7.6.

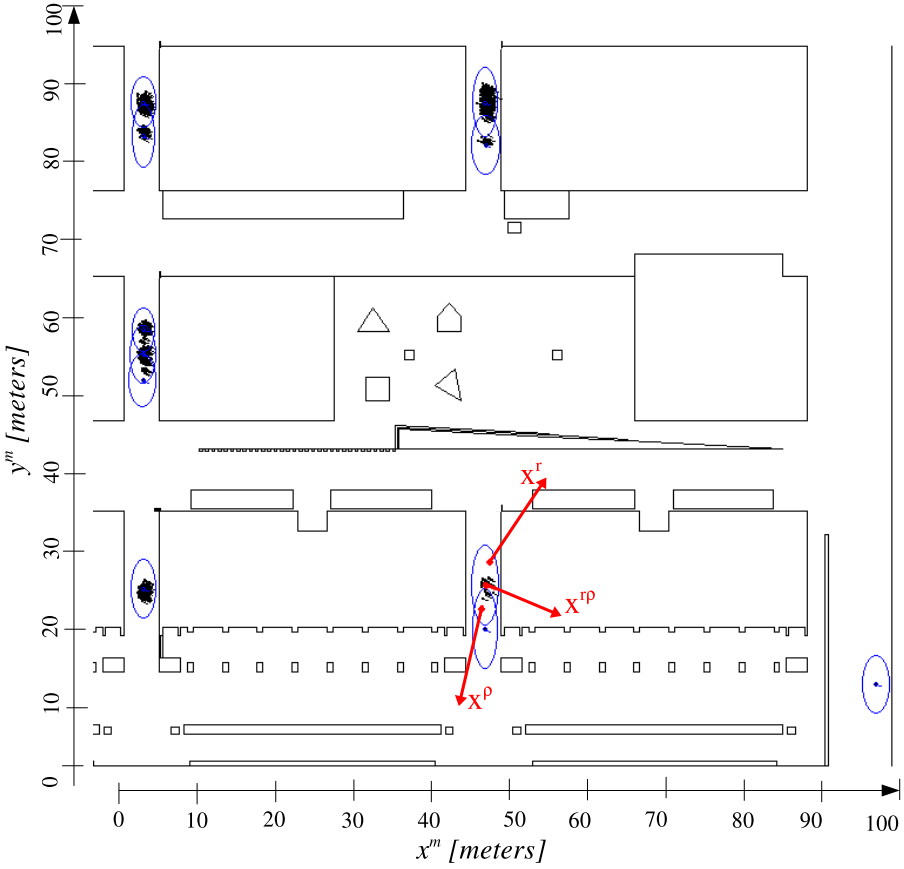


Figure 7.8: The set H of $N_H = 11$ position hypotheses of the cooperative frame. Each hypothesis is marked as a blue ellipse. The x axis of each robot and that of the cooperative frame are also showed.

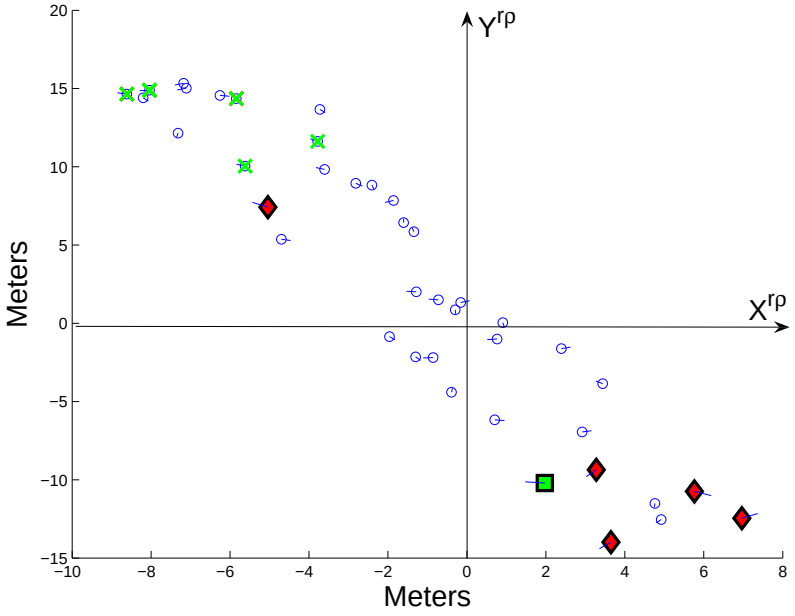


Figure 7.9: The set E of $N_E = 40$ exploration particles that has been evaluated, in the cooperative frame. The green square marks the primary action, $e^{r\rho}_{s1}$. Five best secondary actions following the quantitative criteria (red diamonds). Five best secondary actions following the qualitative criteria (green crosses).

Chapter 8

Software Integration

This chapter presents the software infrastructure used through this thesis to perform experimentation and obtain both real-world (on-line and off-line) and simulated results. Software infrastructure plays a key role in robotic research, specially in areas such as autonomous navigation where experimentaion is required and a large set of algorithms, sensors and devices are involved in a single real-time experiment.

8.1 Overview

The design principle is to decouple algorithm executions from their inputs and outputs. Processes comunicate between them through a TCP network, thus the overall application can be executed in a set of connected machines that run separately processes involved in the final application. This approach is not original of this thesis and it is also the main design principle of all robotic software available nowadays, such as CARMEN [16], openRTM [84], OPRoS [85] ORCA [86], OROCOS [87], Player [90], ROS [99] or YARP [116]. Given these large set of already existing approaches, the question that arises is, why another infrastructure? There are two main answers. Firstly, because the main research and engineering work of this thesis was done in the context of URUS European project [102, 112] and this project used the YARP library as the inter-process communication tool to integrate software pieces among different partners. Secondly, because when the project starts, at year 2006, some programming work was already done, so that the decision was to adapt the already done programming work to the project requirements. As it will be discussed in the conclusions of this thesis (chapter 9), nowadays, at year 2011, a more suitable solution should be to fully adopt an existing solution such as ROS [99] to take benefit of an easy-to-use, stable and open source software integration tool, with a large users community in robotics.

However this chapter is motivated with the aim to report all the programming work done during the thesis development. The chapter first describes how the software is organized through a hierarchy of classes. Afterwards a section details some aspects of the simulator used. Finally a section explains how off-line executions are performed, keeping the synchrony of all data, thanks to the logged timestamps. These off-line executions can be also run scaling the off-line time line by a factor, while keeping synchrony, with the aim of executing real-time processes in a relaxed time conditions. This has been used in section 6.5 to extract high accuracy localization data. A first version of the software architecture reported in this chapter can be found in [20].

8.2 Software architecture

The software is divided between processes and interfaces, so an execution can be viewed as a graph of processes that communicate between them through TCP messages. The design is done through a set of C++ classes organized in two hierarchies, one for the processes and another for the interfaces.

8.2.1 Process hierarchy

The process hierarchy is organized through three levels. The first layer defines the *basic* process class, a class that will be inherited by all the processes. The second layer defines *generic* processes such as 'localization' or 'laser scanner'. This layer mainly defines the input and outputs required for these generic modules. For instance, the generic laser scanner process defines that is a process with no input and providing an output with a laser scanner data message. The third layer is the *implementation* level and holds the final particular classes for each algorithm, where all details and particularities of each solution have to be

implemented. Figure 8.1 shows the process class hierarchy and unfolds, for illustrative purposes, the third layer of this hierarchy for the cases of the platform acquisition and laser scanner devices.

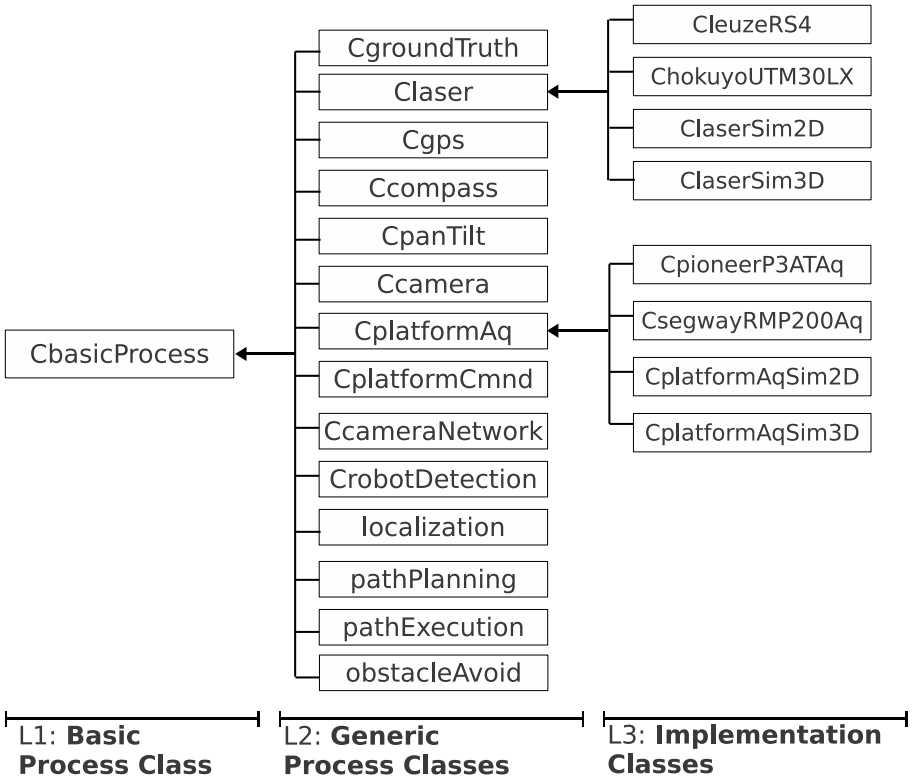


Figure 8.1: The process class hierarchy is divided in three layers.

Layer 1: Basic process class. This class defines the basic loop that all processes will execute as an independent thread, as well as some basic functionalities such as time logging data and events with an associated timestamp. The basic loop, outlined in algorithm 14, consists on three function calls and a time management to execute the loop in a fixed constant rate determined by the process period T_p. The three functions calls, process(), publish() and logData(), are pure virtual functions that will be implemented in layers 2 and 3 of the hierarchy. The function process() will implement the data transformation and computations, publish() will implement the publication of the output data through the process network and the function logData() will be in charge of defining how the relevant data of each process iteration is saved in a log file.

Algorithm 14 Basic process loop

INPUT: T_p //process period (user defined)

```
while run do
  T1 = getTime();
  process();
  publish();
  logData();
  T2 = getTime();
  Δp = Tp − (T2 − T1);
  if Δp > 0 then
    sleep(Δp);
  end if
end while
```

Layer 2: Generic process classes. This layer defines a set of classes, each one encapsulating a generic high-level functionality. However this level does not yet implement details of a given solution for that functionality, so that generic process classes only define inputs and outputs, and some other common utilities, such as the function publish() and logData(). Definition of these two functions in this layer allows to all processes that inherit from a given generic class to do not differentiate from the point of view of the process execution graph, since their inputs and outputs will be the same, as well as the way in which they save the data in a log file. For instance, for the localization generic class, all inherited localization implementations will behave exactly in a same way from the point of view of the integration with other processes. Figure 8.2 shows how the localization generic class defines inputs and outputs (grey part), but leaves the implementation of a particular solution to the third layer of the hierarchy (white box).

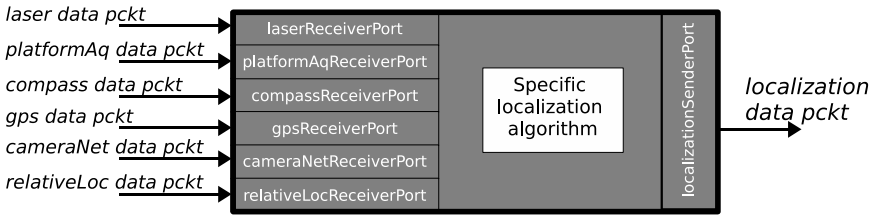


Figure 8.2: Generic class concept for localization. Grey box is defined in the generic localization class (layer 2). White box will be defined for each specific implementation (layer 3).

Layer 3: Implementation classes. This level of the hierarchy is in charge of implementing the process() virtual function and all the algorithms and functions that are particular for a given solution of a given functionality defined in layer 2. For instance, for this thesis four particular laser scanner software modules have been implemented: the leuzeRS4, the hokuyoURGX, a simulated laser scanner in 2D environments and a simulated laser scanner in 3D environments (see figure 8.1).

8.2.2 Communication interfaces

Interfaces implement TCP/IP ports used for interprocess communication. A two layered hierarchy is used to this end. The first layer is a basic interface class, while the second consists on the specific interfaces for each data packet.

Basic Interface. This class inherits the BufferedPort class provided by the YARP library, thus it implements a TCP/IP port. However, this layer does not define the content of the data to be sent or received, this will be implemented in the second layer of the interfaces. Therefore, this layer has the main role of implementing the naming protocol followed to label each comunication port [7]. This naming protocol was designed in the context of the URUS project [112] to label all TCP/IP ports available in a network robot system, involving several robots, while each robot provides several services. This labelling was set following the protocol rules, so that port names were deducible from its functionality and robot entity providing it.

Specific Interfaces. Each data packet required in the application requires a specific interface that mainly defines the data content that will be sent and received through the port, thus each port. i.e. each interface, only manages a given type of data packet. For illustrative purposes, figure 8.3 details the fields of the localization data packet defined in the localization interface.

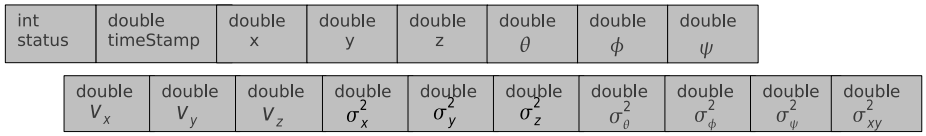


Figure 8.3: The localization data packet defined in the localization specific interface.

8.3 Mobile robot simulator

In the context of this thesis simulation has been used for three main purposes: (1) to execute and debug the software infrastructure before its utilization in real experiments, (2) to compare position tracking methods as it is done in section 6.3 and (3) to validate some theoretical aspects of global localization and obtain preliminary results presented in chapter 7. To this aim two simple simulators have been developed, both consisting on single point mobile platforms but one on 2D environments and the other in 3D worlds. In both cases only platform kinematics are considered and simulated sensors are mainly simple models where Gaussian noise is added to get more realistic noisy data.

The implemented simulator deals with a group of N_R robots. For the r^{th} robot, a platform process holds the position ground truth, X_r^t, updates it when velocity commands are received and publishes this ground truth position. Onboard simulated sensors receive the ground truth position of the associated robot and from this position computes the sensor models to output simulated observations, o_k^t. Moreover, other processes such as the camera network, use all ground truth positions to compute robot detections. Next paragraphs overview, for

the 2D simulator, the kinematic model used for the platform motion and the sensor models used to compute observations. The 3D simulator has been only partially implemented following the same design principles that the 2D one, but using the algorithm described in 4.3.5 for laser scanner simulation in 3D, instead of the one reported in 4.2.3 that it is used for ranging on 2D.

Please note that each process is an independent thread and has its own loop rate, so that iteration index t does not necessarily coincides, since it is a private variable of each process.

Kinematic Model. We simulate a generic holonomic wheeled mobile platform, that, at iteration t, receive translational and rotational velocities, v_ρ^t and v_θ^t, to propagate its position with the kinematic model $f(X_r^{t-1}, v_\rho^t, v_\theta^t)$:

$$X_r^t = \begin{bmatrix} x^r \\ y^r \\ \theta_r \end{bmatrix}^t = \begin{bmatrix} x^r \\ y^r \\ \theta_r \end{bmatrix}^{t-1} + \begin{bmatrix} \Delta T^t v_\rho^t cos(\theta_r^{t-1} + \Delta T^t v_\theta^t) \\ \Delta T^t v_\rho^t sin(\theta_r^{t-1} + \Delta T^t v_\theta^t) \\ \Delta T^t v_\theta^t \end{bmatrix} \tag{8.1}$$

where ΔT^t is the measured elapsed time between iteration $t-1$ and t.

Wheel Odometry. Each platform is equipped with **wheel encoders** that provide, at iteration t, the odometry observation o_U^t. This observation is composed by the increment in translational motion, $\Delta\rho^t$, and the increment in rotational motion, $\Delta\theta^t$. To these increments, simulated Gaussian noise with standard deviation of 5% of $\Delta\rho^t$ in translation and 10% of $\Delta\theta^t$ in rotation is added:

$$o_U^t = (\Delta\rho^t + \mathcal{N}(0, 0.05 * \Delta\rho^t), \Delta\theta^t + \mathcal{N}(0, 0.1 * \Delta\theta^t)) \tag{8.2}$$

where,

$$\Delta\rho^t = \sqrt{(x_r^t - x_r^{t-1})^2 + (y_r^t - y_r^{t-1})^2}; \quad \Delta\theta^t = \theta_r^t - \theta_r^{t-1} \tag{8.3}$$

Laser Scanner. Laser scanners onboard each robot provide, at iteration t, the real observation o_L^t. Laser scanner observations in 2D environments are computed with the algorithm presented in section 4.2.3, providing as inputs the ground truth robot position and the environment model used by the simulator, $\mathcal{M}$. Moreover, to simulate a real observation, a normal noise with $5cm$ of standard deviation is added, so that an entire scan of N_L rays will be:

$$o_1^t = (o_{1,1}^t, .., o_{1,N_{LS}}^t); \quad o_{1,i}^t = rayMapInterference(X_r^{m,t}, \mathcal{M}, i) + \mathcal{N}(0, 0.05) \tag{8.4}$$

To simulate the LeuzeRS4 device, 133 points are used with a loop rate of $6Hz$.

Electronic Compass. Onboard electronic compass is also simulated to provide, at iteration t, the heading observation o_C^t. This observation is directly the heading of the simulated ground truth position with an added Gaussian noise of $0.05rad$ of standard deviation, thus for the r^{th} robot:

$$o_C^t = \theta_r^t + \mathcal{N}(0, 0.05) \tag{8.5}$$

In the simulator, compass observations are always available, even if in real devices compass measurements are sometimes corrupted by magnetic distortions.

However state of the art devices, such as TCM3 from PNI corporation, provide autocalibration procedures to detect external distortions of the magnetic field and signal corrupted readings with an alarm (see section 3.2).

GPS. GPS receiver is also simulated to be onboard the robots. The implemented simulation model addresses, in a simple way, two main characteristics of the GPS observations: partial availability and short-term biased noise (not centered to the true position during short time periods). To follow this requirement, the model is also computed from the simulated ground truth position and adds two noise components: a zero-mean Gaussian noise updated at each iteration and a biased Gaussian noise updated after a period of 20 seconds. Moreover a set of coverage circles are defined on the map representing the GPS coverage area on the environment. When the robot is on the coverage area for at least three consecutive iterations, the GPS observation is available and is computed as:

$$o_G^t = (x_r^t, y_r^t) + (\Delta_{GPSx}, \Delta_{GPSy}) + (\mathcal{N}(0, 0.5)) \tag{8.6}$$

where Δ_{GPSx} and Δ_{GPSy} are the bias noise component and are updated each 20 seconds as:

$$\Delta_{GPSx} = \mathcal{N}(0, 2); \quad \Delta_{GPSy} = \mathcal{N}(0, 2); \tag{8.7}$$

Sensor Network. A network of cameras deployed on the environment is modelled as a set of N_C **omnidirectional cameras** fixed at known positions. This camera network provides, at iteration t, the observation o_N^t. The model to compute real observations of the camera network is the same than the one exposed in section 7.4, but adding normal noise to the measurements. If at iteration t, the r^{th} robot is in the coverage area of the c^{th} camera, $X_r^t \in C_c$, a real observation of the camera network is available as:

$$o_N^t = \begin{bmatrix} \delta_{c,r}^t \\ \alpha_{c,r}^t \\ \phi_{c,r}^t \end{bmatrix} = \begin{bmatrix} \sqrt{(x_r^{m,t} - x_c)^2 + (y_r^{m,t} - y_c)^2} + \mathcal{N}(0, 0.5)) \\ atan(\frac{y_r^{m,t} - y_c}{x_r^{m,t} - x_c}) + \mathcal{N}(0, 0.02 * \delta_{c,r}^t) \\ \theta_r^{m,t} + \mathcal{N}(0, 0.2) \end{bmatrix} \tag{8.8}$$

Robot to Robot Detection. When the ρ^{th} robot sees the r^{th} one within a line of sight of length less than $R_R = 10m$, the observation made by the ρ^{th} robot at iteration t is a relative localization, so it is expressed in terms of the ρ^{th} frame. The detection model is the same described in subsection 7.3.2 but adding some Gaussian noise to the data.

$$o_D^t = \begin{bmatrix} \delta_r^{\rho,t} \\ \alpha_r^{\rho,t} \\ \phi_r^{\rho,t} \end{bmatrix} = \begin{bmatrix} \sqrt{(x_r^t - x_\rho^t)^2 + (y_r^t - y_\rho^t)^2} + \mathcal{N}(0, 0.05)) \\ atan(\frac{y_r^t - y_\rho^t}{x_r^t - x_\rho^t}) + \mathcal{N}(0, 0.01 * \delta_r^{\rho,t}) \\ \theta_r^t - \theta_\rho^t + \mathcal{N}(0, 0.1) \end{bmatrix} \tag{8.9}$$

8.4 Off-line executions

Off-line executions are useful to compare different localization approaches (or other estimation processes) using exactly the same set of real sensory data. The proposed software infrastructure has implemented a 'logReader' process

that gets data from a set of log files and publishes this data through diferent interfaces in the same way as sensor acquisition processes do. Publication of data keeps the synchrony given by the logged timestamps so that off-line executions provides a data flow very similar as that obtained during the on-line acquisition. Moreover, the logReader process can slow down the off-line time by a factor. Slowing down the off-line time is equivalent to multiply the off-line time line by a factor, $\alpha > 1$, thus the data flow will be published more slowly but keeping synchrony. This procedure allows to execute localization with real data but relaxing the real-time constraints, as it has been done in section 6.5 to extract high accuracy localization data with a particle filter using a large particle set.

Chapter 9

Contributions, Conclusions and Future Works

Ending a thesis gives the opportunity to look backwards and write a valuable set of conclusions about the work, with the aim of sharing them with the community. Moreover, a thesis work also opens even more questions that it answers, and it is also important to clearly define which are these new questions to be adressed by future works.

9.1 Main Contributions.

Next paragraphs discuss the main contributions that have been resulted from this thesis.

Fast and Online Computation of Expected Observations. To extend the particle filter to the use of 3D models while keeping real-time performance, it has been necessary to develop and implement new observation models accomplishing these real-time requirements. With this aim, a new range observation model has been developed [24]. It uses the openGL computer graphics library to renderize the 3D environment model in a proper way that minimize the computational time while it keeps the required angular and ranging accuracy of laser scanner devices. This range observation model has been successfully and massively used in the long-term experiments carried out in two scenarios: the campus of the Universitat Politècnica de Catalunya and in an urban scenario in the city of Barcelona.

Integration of Delayed Observations in a Particle Filter. In cooperative environments, observations made by remote observation processes arrive to the computer that executes the localization filter with a non negligible delay relative to the platform speeds. This fact has forced to redesign how such delayed observations are integrated in the particle filter framework, to keep real-time requirements in the sense of outputing an accurate estimation with minimal latency, ready to be used for other navigation control processes [22]. The thesis presents simulation results comparing a basic particle filter with the modified one that shows improvements, in terms of localization accuracy, when the time stamp of each observation is properly considered, so that each observation is integrated in the correct time moment.

Long-term localization in autonomous navigation sessions. Several autonomous navigation sessions have been performed in real outdoor pedestrian environments to test a whole navigation system, but in particular to test also the proposed localization approach. About a $3Km$ execution using the 2D approach [25] and more than $6Km$ using the 3D one [111] have been navigated by an experimental two-wheel self-balancing Segway platform. Localization results shown the robustness of the approach while they point out weakness and future reserach directions to cope with them.

Active and Cooperative Global Localization. This thesis presents theoretically a general probabilistic approach to solve the map-based global localization problem in large environments. This has been done by proposing an active strategy based on estimating the reduction of position hypotheses [21, 23]. The method is general since it is neither sensor dependent nor spatial representation dependent and it uses the same observation models used by the position tracking. The presented algorithm is computationally efficient, both in time and memory, when compared to other existing approaches. Moreoever, the most important contribution of this new active approach is the possibility to use it in cooperative environments, both for a single robot using a sensor network and for multiple robots within a sensor network context. In the former case the

cooperation arises in terms of information sharing since the robot uses observations coming from remote sensors, while in the later case cooperation is further developed as an strategy that plans coordinated joint actions where two lost robots are in a line of sight.

9.2 Secondary Conclusions.

Next paragraphes outline some secondary conclusions derived from the gained author's experience through the thesis work.

Map-based Particle Filter Localization. From the study of the state of the art as well as from the practical experience gained through the work in this thesis, the particle filter approach have shown adequate properties to address the data fusion issue required to solve the map-based localization problem. Particle filters are flexible and robust to deal with the stochasticity of map-based localization, that is often more critical in terms of model innacuracies than in terms of gaussian noise of sensor readings. In this thesis, the filters have been adapted to cope different situations such as the model incompleteness and the integration of asynchronous and delayed data.

2D vs 3D Environment Models. Map-based position tracking in urban pedestrian areas have been addressed using both 2D and 3D geometric environment models. In both cases, experimental results are obtained from long-term autonomous navigation sessions in an outdoor campus area. The filter using the 3D model, that extends the position state to a 6DOF space, outperformed the 2D approach, specially in situations with high presence of 3D environment elements, commonly found in urban pedestrian scenarios, such as ramps, steps or holes.

Segway Platforms in Urban Environments. Two-wheel, self-balancing platforms such as that used for experimentation in this thesis offer practical benefits, specially in terms of mobility in urban pedestrian settings, and also in terms of payload to carry the sensors, computers and batteries required for long-term autonomous navigation in such environments. However, these platforms present also some drawbacks, specially in terms of perception since modelling the self-balancing behavior is not evident when the platform has variable payloads and runs on tilted surfaces. The extension of the localization to 3D maps with a 6DOF state space seems to be mandatory to cope with the self-balancing of the platform. This extension to 3D has dramatically improved the likelihood scores between real observations and the expected ones during the filter execution.

Software Infrastructure. In robotics, specially in that fields where long-term experiments have a key role in the research loop, having a solid software infrastructure is a key factor for a successful research. In this sense, during the thesis development a software infrastructure has been developed based on the YARP middleware and it has allowed to perform experimental sessions involving several processes and machines. Besides many technical limitations, this software infrastructure had only a local community of non-full-time developers,

so the software maintenance and improvement of this infrastructure has been collapsed, leading to a logical (and ecological) situation of abandon. The conclusion is that spending time in software design and development is one of the best investments that a reserach center can do in order to succeed in future projects and drastically improve its productivity. In that line, it is strategic to develop low level software independently from a given framework but to encapsulate it with some of the currently existing robot software frameworks. This leads to participate to international software projects and to benefit of code reusability and user debugging. In this line, the robotics laboratory services at the Institut de Robòtica i Informàtica Industrial (IRI) has started, from the mids of 2010, to build a solid software infrastructure that is encapsulated with ROS for multi-process executions [42].

9.3 Future Works.

From the conclusions reported in the previous section, a set of future research lines and development works are outlined in the following paragraphes. This helps to arrange ideas and organize future tasks.

Extensive Evaluation of Position Tracking Robustness. Developing methodologies to evaluate the robustness of the position tracking approach, beyond the particular conditions encountered in the experimental sessions, is a major issue. Robustness should be mainly evaluated in terms of position recovery, showing how the filter is capable to deal with the main causes of localization fails: the environment dynamics and model innacuracies encountered in practical implementations.

Dynamic model of the Segway platform. Using only kinematics for state propagation of a mobile platforms has limitations in terms of accuracy, specially for long-term predictions or high-speed platforms. In the case of two-wheel self-balancing platforms, integration of inclinometer and torque data in a dynamical model would lead to obtain a more accurate probabilistic transition model to compute better proposal densities in the particle filter, and so improve the estimation performance.

Visual Odometry and Multi-loop Localization. Visual odometry has received great attention in recent years, and the state of the art report impressive results. A visual odometry module provides valuable data to a map-based particle filter localization process, in the sense that the proposal density can be more accurately estimated, thus reducing the particle set required to represent it, so computational efforts could decrease, or could be focused on growing the accuracy and robustness. Multi-loop approaches stands for localization systems that combine multiple techniques at different rates, in order to estimate a robust position of the platform dealing with particularities of each layer. A first proposal for map-based multi-layer localization could consist on having at the lower layer a visual odometry module, fusing data from camera, wheel encoders, inclinometers, torque sensors and, if it was present, an inertial measurement unit. A second layer would execute a local SLAM, so that robust localization would be achieved thanks to the correction with the concurrently created map, but not

with the static one. Finally, a third layer, running at slower rate, would match this locally created maps with the a piori provided model of the environment, so poducing global correction of the estimated position.

Optimal Particle Filter. Results of real-time particle filter for position tracking has demonstrate the possible presence of an optimal number of particles (N_P) in terms of localization accuracy. The presence of an optimal number of particles is derived by the fact that having a filter with very few particles will lead clearly to a big accuracy error, but having larger number of particles implies larger latency on the estimation output, that could be only overcome by outputing a prediction (a prior), so the error increases as the prediction covers larger latencies. In the experiments reported in the thesis it has been observed that the most accurate filter has resulted as that iterating at a rate close to the horizontal laser scanners acquisition rate, the most important sensors. This leads to investigate theoretical evidences from information theory, specially on quantifying the information gain rate of the filter as a function of N_P, to check if the most informative filter coincides with the most accurate one.

Relative Localization. A key module to continue the research on multi-robot localization is the relative localization among robots. For a robust solution, this could be implemented with a fusion of visual and laser data to detect and track other robots within the local surroundings of the given robot. Having such a module, cooperative position tracking or active global localization could be implemented and tested in real platforms.

Camera Network. In order to integrate absolute position information provided by a camera network deployed on the environment, the camera network has to be metrically calibrated with the map, so the fusion of information would be metrically consistent. This implies to develop calibration procedures that produce the set of 6DOF positions of the cameras with respect to the map frame. Such techniques are not evident to implement and they have to be executed periodically since cameras placed in outdoor environments move due to environmental conditions such as wind and others. Due to these issues, localization methods that avoid this global calibration should be investigated, exploiting relative localization data that can be provided by fixed cameras not necessarily calibrated with the map.

Active Global Localization from Information Theory. The approach presented in chapter 7 to solve actively the global localization problem could be formalized by means of some information theory tool such as information gain. A novel formulation of such approach should allow to investigate information gain of a given trajectory candidate instead of only consider the information gain of the final target point.

Appendix A

Tibi and Dabo Mobile Robots

Two mobile service robots, designed to operate in urban, pedestrian areas, were bought and properly modified and equipped for the URUS project. These are Tibi and Dabo, pictured in figure A.1. They are based on two-wheeled, self-balancing Segway RMP200 platforms, and as such are highly mobile, with a small footprint, a nominal speed up to $4.4m/s$, and the ability to rotate on the spot (while stationary).

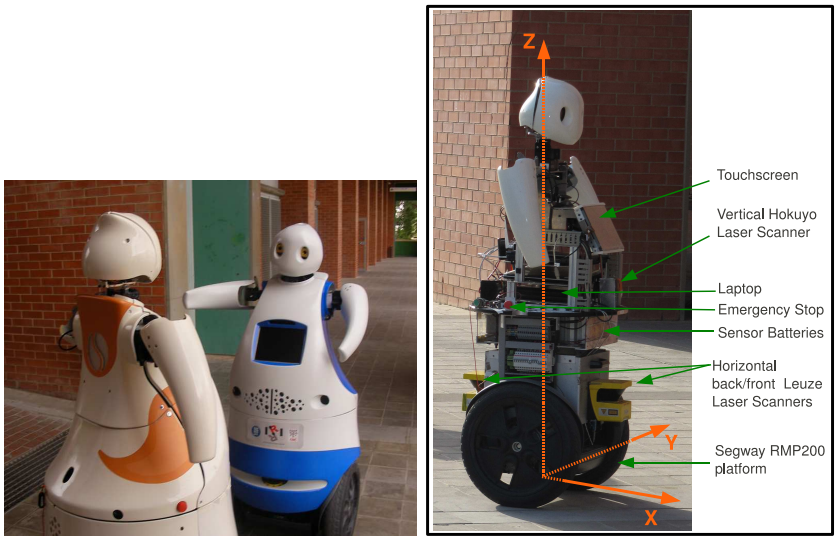


Figure A.1: On the left, Tibi (left) facing Dabo. On the right, on-board devices used in this work, and the robot coordinate frame.

They are equipped with the following sensors:

- Two Leuze RS4 2D laser range finders, scanning over the XY plane, pointing forward and backward respectively, at a height of $40cm$ from the ground. These scanners provide 133 points over 190° at the fastest

setting, running at approximately $6Hz$. This device has a range of $64m$, but in practice we limit it to $15m$. Front and back laser observations are notated as $o^t_{L_F}$ and $o^t_{L_B}$ respectively.

- A third 2D laser scanner, a Hokuyo UTM-30LX, mounted at a height of $90cm.$, pointing forward and rotated 90° over its side, scanning over the XZ plane. This scanner provides 1081 points over 270° at $40Hz$, and has a range of $30m$, again capped to $15m$. Aperture is limited to 60° to ignore points interfering with the robot's frame or aiming too high for our needs. This observation is notated as $o^t_{L_V}$.

- Wheel encoders, providing odometry readings o^t_U, from the Segway platform.

- Inclinometers from the Segway platform, providing pitch and roll data, o^t_I.

The robot also features two stereo camera pairs and a GPS receiver, which are not used in this work. The user can interact with the robot through a touchscreen, entering *go-to* requests manually. Two off-the-shelf laptop computers running Ubuntu Linux are on-board the robot, one for navigation and the other for communications and human-robot interaction. Experiments were performed using only one robot at time, Tibi or Dabo.

The Segway RMP200 is in many ways an ideal platform to build an urban robot. Humanoid robots are not yet ready for outdoor environments, and four-wheeled vehicles have a much larger footprint and are more restricted in their mobility. Moreover, Segway robots can carry heavy payloads, up to 45 kg for this model. On the downside, two-wheeled platforms are statically (and dynamically) unstable, keeping their balance using gyroscopic sensors to track and correct their tilt. The robot will pitch forward or backward to accelerate or decelerate, or simply to keep its balance while stationary. This behavior presents two issues for their use in robotics.

On one hand, it creates a perception issue for on-board 2D laser scanners. A 2D laser range finder scanning over the XY plane, a very common solution in robotics for navigation or SLAM, may point higher towards the sky/roof or, more critically, lower towards the ground. Using this configuration may result in spurious features or obstacles, unless some kind of filtering is used. Figure A.2 displays a sequence of 2D range data over time, starting with the robot in a stationary, upright position, which is then instructed to move forward and later to stop. The front laser visibility is reduced significantly due to the platform's tilt, up to $2m$ on a flat surface and less on a ramp. The figure also shows velocity commands and the estimation for velocity and pitch from the Segway platform, for the same sequence. This data was taken under laboratory conditions, on a flat, regular surface. On outdoor environments this behavior is much more pronounced, specially in slopes and changes in slope.

The second issue in using Segway platforms is control: the platform's own control algorithm takes precedence over the user's instructions, as its first priority is to stay upright. This problem, present in all Segway platforms, is compounded by the fact that our robots weigh about $120Kg$, which slows them down. In practice, the platform typically takes one to two seconds to react to the user's commands, or even more in extreme situations such as when moving

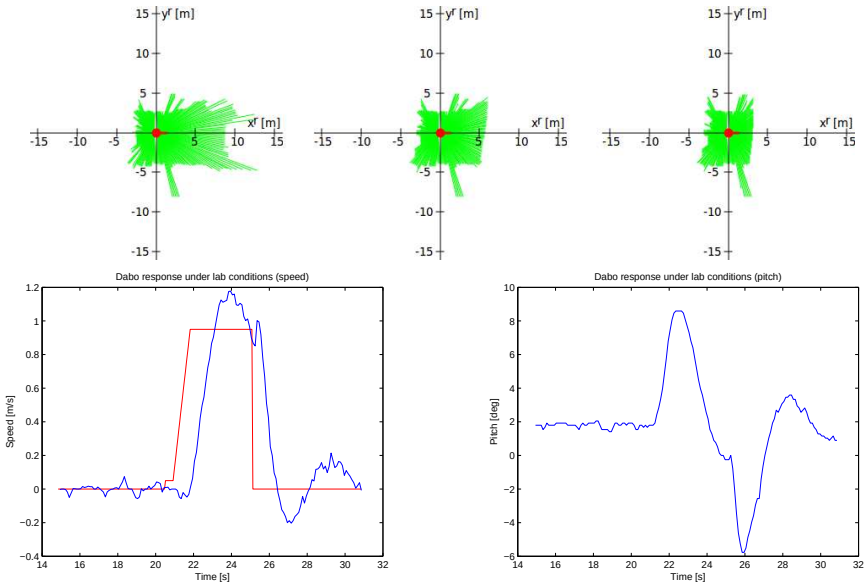


Figure A.2: On top, left to right, sequence of horizontal laser scans with the robot accelerating forward on a flat surface. Time between scans is about $0.6s$. On the bottom left, commands for translational velocity (v) in red, and its estimation from the Segway platform in blue. On the bottom right, pitch estimation from the Segway platform.

from a flat surface to a slope or vice-versa. This ultimately means it is not possible to execute previously planned trajectories with a high degree of accuracy. An example can be seen in figure A.2.

Appendix B

Cposition3d class

Purpose

This appendix details the mathemathics used to code a C++ class that implements a position in the three dimensional space, that is a vector representing a location and an orientation in the space. Such a position can be also interpreted as a coordinate frame and therefore a set of functions allowing rotations around arbitrary axis and displacements along arbitrary directions are programmed. Therefore, this appendix acts as a mathematical guide of the Cposition3d class but not as a programming guide. For programming issues the reader can directly explore the code, its comments and the associated example programs.

Mathematical Definitions

Given a reference coordinate frame, $\mathcal{F}$, in the three dimensional space, a position p, or frame $\mathcal{P}$, is defined by means of three *location* coordinates $(x_p^{\mathcal{F}}, y_p^{\mathcal{F}}, z_p^{\mathcal{F}})$ and some mathematical representation of the *orientation* as a set of three Euler angles, a direct cosine matrix (rotation matrix) or a quaternion. In this implementation the Euler angles representation is used while keeping a rotation matrix updated when necessary, since it provides advantages in some computations. The three Euler angles, $(\theta_p^{\mathcal{F}}, \phi_p^{\mathcal{F}}, \psi_p^{\mathcal{F}})$, are defined following the convention $z - y - x$. For notation simplicity, no frame superindex will be indicated, meaning that the formulation is general for any given frame or that the position is expressed in the world fixed frame, $\mathcal{W}$. When multiple frames are involved in the scene, the superindex will be indicated when necessary to avoid confusion. Summarizing, a position in the 3D space is defined as:

- Location, (x_p, y_p, z_p), as a 3D point indicating the location of p in terms of the reference frame.
- Heading, θ_p, is the first angle, a rotation around the Z axis of the reference frame, $\theta_p \in (-\pi, \pi]$.
- Pitch, ϕ_p, is the second angle, a rotation around the current (once rotated) y axis, $\phi_p \in (-\pi/2, \pi/2]$.

- Roll, ψ_p, is the third angle, a rotation around the current (twice rotated) x axis, $\psi_p \in (-\pi, \pi]$.

Figure B.1 shows the order of the rotations of each euler angle to get the final orientation of the position.

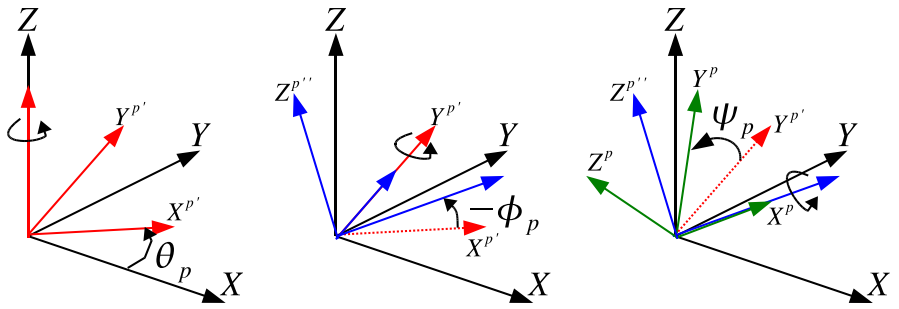


Figure B.1: Definition of the three euler angles heading (θ_p), pitch (ϕ_p) and roll (ψ_p). Reference frame drawn in black. Intermediate frames drawn in red and blue. Final orientation frame shown in green.

updateRmatrix(): Rotation Matrix from Euler Angles

Since rotations are performed always around the current axis instead of around the fixed world axis, the whole rotation matrix is computed as:

$$R(\theta_p, \phi_p, \psi_p) = R_p = R_z(\theta_p) R_y(\phi_p) R_x(\psi_p) \tag{B.1}$$

where,

$$\begin{aligned} R_z(\theta_p) &= \begin{pmatrix} \cos\theta_p & -\sin\theta_p & 0 \\ \sin\theta_p & \cos\theta_p & 0 \\ 0 & 0 & 1 \end{pmatrix}; \\ R_y(\phi_p) &= \begin{pmatrix} \cos\phi_p & 0 & -\sin\phi_p \\ 0 & 1 & 0 \\ \sin\phi_p & 0 & \cos\phi_p \end{pmatrix}; \\ R_x(\psi_p) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi_p & -\sin\psi_p \\ 0 & \sin\psi_p & \cos\psi_p \end{pmatrix}; \end{aligned} \tag{B.2}$$

Rotation Matrix Interpretation

The rotation matrix R_p has an important meaning, since its columns hold the three vectors building the orthogonal basis oriented as the positon p, also called the frame p. Let (X_p, Y_p, Z_p) be the three vectors of the frame p expressed in

terms of the reference frame, thus their coordinates will be:

$$X_p = \begin{pmatrix} R_{p_{00}} \\ R_{p_{10}} \\ R_{p_{20}} \end{pmatrix}; Y_p = \begin{pmatrix} R_{p_{01}} \\ R_{p_{11}} \\ R_{p_{21}} \end{pmatrix}; Z_p = \begin{pmatrix} R_{p_{02}} \\ R_{p_{12}} \\ R_{p_{22}} \end{pmatrix}; \tag{B.3}$$

where the subindex after p indicates the row and column index of the matrix. This indicates that the orientation part of a position can be also uniquely defined by its rotation matrix instead of the three Euler angles.

updateEulerAngles(): Euler Angles from Rotation Matrix

Since some functions as rotateUaxis() update the rotation matrix without setting the Euler angles explicitly, sometimes we need to compute those angles from the R_p matrix in order to keep consistency between the matrix and the angles.

The heading is directly computed as the angle from the X reference axis to the projection of the X_p axis with the plane formed by X and Y:

$$\theta_p = atan2(R_{p_{10}}, R_{p_{00}}) \tag{B.4}$$

To compute the pitch we have to compute first the axis $X_p^{'}$ (see figure B.1). $X_p^{'}$ can be seen as the normalized projection of the axis X_P to the plane formed by the vectors X and Y or as the axis X after a single rotation of θ_p. For both cases, the expression is as follows:

$$X_p^{'} = \frac{1}{\sqrt{R_{p_{00}}^2 + R_{p_{10}}^2}} \begin{pmatrix} R_{p_{00}} \\ R_{p_{10}} \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\theta_p \\ \sin\theta_p \\ 0 \end{pmatrix} \tag{B.5}$$

The pitch angle is defined as the angle of the vector X_p with the vector $X_p{}'$ in the plane built by X' and Z:

$$\phi_p = -atan2(X_P \cdot Z \ , \ X_P \cdot X_p{}') = -atan2(R_{p_{20}} \ , \ \frac{(R_{p_{00}}^2 + R_{p_{10}}^2)}{\sqrt{R_{p_{00}}^2 + R_{p_{10}}^2}}) \tag{B.6}$$

The roll angle is computed in a similar way as that used to find the pitch, but the roll angle is defined as that from the $Y_p{}'$ axis to the Y_P axis (see figure B.1). First we find the $Y_p{}'$ and the $Z_p{}''$ axis as:

$$\begin{aligned} Y_p{}' &= Z \times X_p^{'} = \frac{1}{\sqrt{R_{p_{00}}^2 + R_{p_{10}}^2}} \begin{pmatrix} -R_{p_{10}} \\ R_{p_{00}} \\ 0 \end{pmatrix} = \begin{pmatrix} -\sin\theta_p \\ \cos\theta_p \\ 0 \end{pmatrix}; \\ Z_p{}'' &= X_P \times Y_p^{'} = \frac{1}{\sqrt{R_{p_{00}}^2 + R_{p_{10}}^2}} \begin{pmatrix} -R_{p_{00}} R_{p_{20}} \\ -R_{p_{10}} R_{p_{20}} \\ R_{p_{00}}^2 + R_{p_{10}}^2 \end{pmatrix} = \begin{pmatrix} \cos\theta_p \sin\phi_p \\ \sin\theta_p \sin\phi_p \\ \cos\phi_p \end{pmatrix} \end{aligned} \tag{B.7}$$

The roll angle will be:

$$\psi_p = atan2(Y_p \cdot Z_p{}'' \ , \ Y_p \cdot Y_p{}') = \\ = atan2(R_{p_{21}}(R_{p_{00}}^2 + R_{p_{10}}^2) - R_{p_{20}}(R_{p_{00}}R_{p_{01}} + R_{p_{10}}R_{p_{11}}), R_{p_{00}}R_{p_{11}} - R_{p_{10}}R_{p_{01}}) \tag{B.8}$$

Equations B.4, B.6 and B.8 explicit how to compute the Euler angles given the rotation matrix. This will be useful to keep consistency between angles and matrix when some functions manipulate the matrix.

moveForward(): Forward displacement

This function displaces the position a distance Δ along its X_p axis. This is computed as:

$$\begin{aligned} x_p^+ &= x_p + \Delta R_{p_{00}} \\ y_p^+ &= y_p + \Delta R_{p_{10}} \\ z_p^+ &= z_p + \Delta R_{p_{20}} \end{aligned} \tag{B.9}$$

rotateUaxis(): Rotation around an arbitrary axis

Sometimes we want to rotate the position around an arbitrary axis defined in reference coordinates as the vector $u = (x_u, y_u, z_u)^T$. To perform this operation the class implements the Rodrigues formula [73]. Given the matix U and Q defined as:

$$U = u \cdot u^T; \quad Q = \begin{pmatrix} 0 & -z_u & y_u \\ z_u & 0 & -x_u \\ -y_u & x_u & 0 \end{pmatrix} \tag{B.10}$$

The Rodrigues Formula computes the rotation matrix associated to a rotation of α around the u axis as:

$$R(\alpha, u) = U + (I - U)\cos\alpha + Q\sin\alpha \tag{B.11}$$

And finally we have to update the whole rotation matrix as:

$$R_p^+ = R(\alpha, u)R_p \tag{B.12}$$

Where superindex $^+$ indicates the updated version of the matrix.

turnZaxis(): vehicle turn

This function implements the turning of a ground vehicle oriented in some position. It is summarized as a call to the previously defined rotateUaxis() function, passing as the axis parameter the Z_p vector since the turning is around the vehicle vertical axis.

coordTr(): Coordinate transformation

Point Transformation

A vector in the 3D space is always referred to some frame. Let be a^w the point a in terms of some reference frame as the world one, $\mathcal{W}$. Let be $\mathcal{P}$ another frame expressed also in terms of the $\mathcal{W}$, with its location coordinates $l_p^w = (x_p^w, y_p^w, z_p^w)$ and the rotation matrix R_p^w. Then the transfromation between a^w and a^p is as follows:

$$a^w = R_p^w a^p + l_p^w \tag{B.13}$$

The above equation can be reformulated using homogeneeous coordinates as:

$$\hat{a}^w = H_p^w \hat{a}^p \tag{B.14}$$

where $\hat{a}^i = (x_a^i, y_a^i, z_a^i, 1)^T$ and the matrix H_p^w encapsulates the whole transformation (rotation and translation) as follows:

$$H_p^w = \left(\begin{array}{ccc|c} & R_p^w & & l_p^w \\ & & & \\ 0 & 0 & 0 & 1 \end{array}\right) \tag{B.15}$$

Alternatively, given the point a expressed in terms of $\mathcal{W}$, a^w, we can find its expression in terms of the frame $\mathcal{P}$ as:

$$\hat{a}^p = (H_p^w)^{-1} \hat{a}^w = H_w^p \hat{a}^w \tag{B.16}$$

Position or Frame Transformation

Given a frame $\mathcal{P}$ expressed in terms of the world frame $\mathcal{W}$ by the homogeneous matrix H_p^w and another frame $\mathcal{Q}$ expressed in terms of the $\mathcal{P}$-frame by the homogeneous matrix H_q^p, the expression of the $\mathcal{Q}$-frame in terms of $\mathcal{W}$ is:

$$H_q^w = H_p^w \; H_q^p \tag{B.17}$$

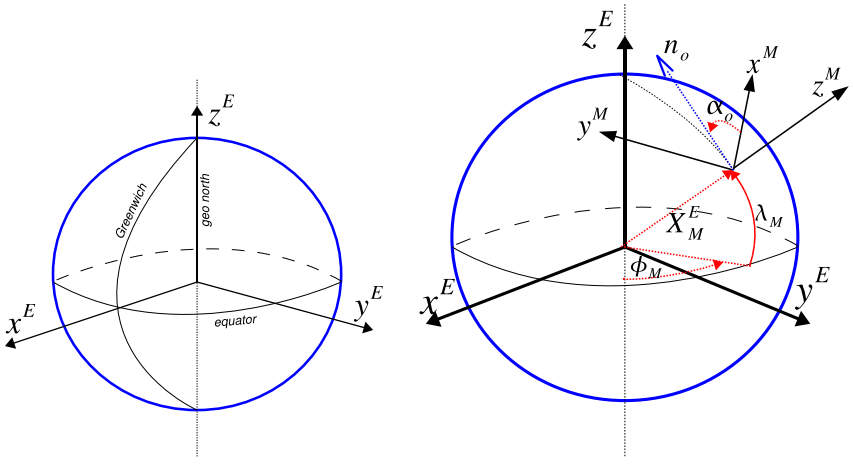
This last equation can be used when a concatenation of frames are involved as a robotic arm or a sensor-platform-world system.

Appendix C

GPS Coordinate Transformation

Purpose

This appendix deals with the GPS coordinates issue. First, it summarizes the equations to convert geographic coordinates latitude, longitude and altitude, (λ, ϕ, h) to the Earth-Fixed Earth-Centered (ECEF) coordinates. Second, it details the computation of the homogeneous transformtion matrix that converts a point expressed in ECEF coordinates to a local coordinate frame, as the map coordinate frame used in this thesis to fuse all observations. Figure C.1 shows the three coordinate frames involved.



(a) Geo and ECEF frames (b) ECEF and Map frames

Figure C.1: Geographic, ECEF and map coordinates.

From Geodetic to ECEF

Even if nowadays most of the GPS receivers provide directly ECEF data in their output, for the completeness of the appendix, the equations to convert the geographic coordinates of a point, (λ_p, ϕ_p, h_p) to its ECEF coordinates (x_p^E, y_p^E, z_p^E) are summarized below.

Given, the constants R_e and E_e, that are respectively the Earth radius and the Earth excentricity, and have the values $R_e = 6378137m$ and $E_e = 0.081819190842622$, the conversion from geographic coordinates to ECEF for the point (λ_p, ϕ_p, h_p) is:

$$\begin{aligned} K_e &= \frac{R_e}{\sqrt{1-(E_e sin(\lambda_p))^2}} \\ x_p^E &= (K_e + h_p)cos(\lambda_p)cos(\phi_p) \\ y_p^E &= (K_e + h_p)cos(\lambda_p)sin(\phi_p) \\ z_p^E &= ((1-E_e^2)K_e + h_p)sin(\lambda_p) \end{aligned} \tag{C.1}$$

From ECEF to Map Coordinates

This coordinate transformation can be expressed with an homogeneous matrix integrating three rotations, a coordinate change and a translation from the center of the earth, (x_o^E, y_o^E, z_o^E) , to the origin of the map, (x_M^E, y_M^E, z_M^E).

To find this transformation it is required to know accurately the geographic coordinates of the map origin, (λ_M, ϕ_M, h_M). This step can be resolved using Google Maps as shown in figure C.2, to find the origin of the map frame in terms of geographic coordinates λ_M and ϕ_M. To find altitude, h_M, we use a topographic map. The image from Google Maps is also used to find the orientation of the x_E^M axis with respect to n_o, the local geographic north vector. This orientation is the value of α_o and is computed as:

$$\alpha_o = arctg(\frac{p_h}{p_w}) \tag{C.2}$$

where p_h and p_w are the number of pixels of each segment of the Google Maps image, shown in figure C.2. Figure C.2 depicts the Google Maps snapshot with the two segments used to compute α_o, the origin of the map frame and the x^M and y^M axis.

The first rotation matrix, R_1, is a rotation on the z^E axis of ϕ_M:

$$R_1 = \begin{pmatrix} cos\phi_M & sin\phi_M & 0 \\ -sin\phi_M & cos\phi_M & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{C.3}$$

A second matrix, R_2, describes a rotation on the current y axis of $-\lambda_M$:

$$R_2 = \begin{pmatrix} cos\lambda_M & 0 & sin\lambda_M \\ 0 & 1 & 0 \\ -sin\lambda_M & 0 & cos\lambda_M \end{pmatrix} \tag{C.4}$$

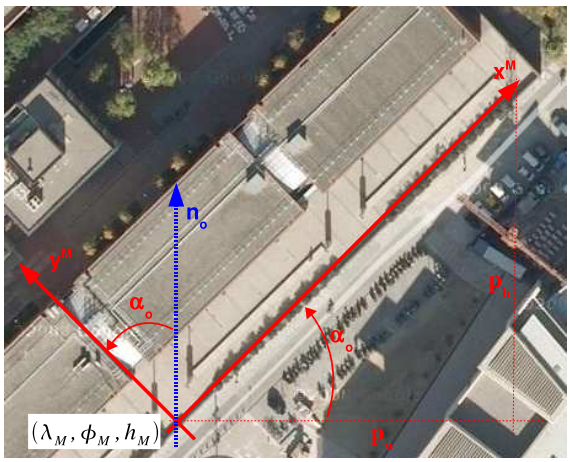


Figure C.2: Google Maps image with the map coordinate frame (x^M, y^M, z^M), the geographic north vector n_o, and the segments to compute the orientation angle α_o.

A third matrix, R_3, is a change of the coordinates to force the new z axis to point up to the sky, instead of x:

$$R_3 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \tag{C.5}$$

Applying R_1, R_2 and R_3 the so called ENU (east, north, up) coordinates are obtained. However, the map frame of this work needs an additional rotation since the map frame is not aligned with respect to the north vector n_o. Therefore, a last transformation, R_4, is required on the current z axis, rotating the frame α_o.

$$R_4 = \begin{pmatrix} cos\alpha_o & sin\alpha_o & 0 \\ -sin\alpha_o & cos\alpha_o & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{C.6}$$

o The complete transformation matrix is:

$$R_E^M = R_4 R_3 R_2 R_1 \tag{C.7}$$

Summarizing, a point $X_p^E = (x_p^E, y_p^E, z_p^E,)^T$ described in terms of the ECEF coordinates can be expressed in terms of the map coordinates, X_p^M, applying the following:

$$X_p^M = R_E^M \cdot (X_p^E - X_M^E) \tag{C.8}$$

In order to express the whole transformation in a more compact way, an homogeneous matrix can be found as:

$$H_E^M = \begin{pmatrix} & R_M^E & & \vert & - R_E^M \cdot X_E^M \\ & & & \vert & \\ 0 & 0 & 0 & \vert & 1 \end{pmatrix} \tag{C.9}$$

Therefore a point expressed in terms of the ECEF coordinates, X_p^E can be expressed in terms of map coordinates applying the whole homogeneous transformation:

$$X_p^M = H_E^M \cdot X_p^E \tag{C.10}$$

To convert a velocity vector expressed in terms of ECEF coordinates, v_p^E, the transformation consists in just applying the rotation part of the homogeneous matrix, that is:

$$v_p^M = R_E^M \cdot v_p^E \tag{C.11}$$

Finnally, the numeric values used in this thesis are provided:

$$\begin{aligned} \lambda_M &= 41.388595^\circ \\ \phi_M &= 2.113133^\circ \\ h_M &= 90m \\ \alpha_o &= 44.2^\circ \end{aligned} \tag{C.12}$$

that lead to the following homogeneous transformation:

$$H_E^M = \begin{pmatrix} -0.4870605 & 0.6994270 & 0.5230430 & 14787.21 \\ -0.4479657 & -0.7141685 & 0.5378569 & 15206.022 \\ 0.7497325 & 0.0276635 & 0.6611625 & -6368887.8 \\ 0. & 0. & 0. & 1 \end{pmatrix} \tag{C.13}$$

Appendix D

Leuze RS4 and Hokuyo UTM30-LX Observation Models

Two kind of laser scanner models have been used, using the same software. Table D.1 summarizes the input parameters of these laser scanner models. Our implementation sets angular accuracies equal to angular resolutions. Please note also that, due to application requirements, we only model part of the scan provided by the Hokuyo laser.

Table D.1: Input device parameters for laser scanner models

Input Parameter	Leuze RS4	Hokuyo UTM 30-LX (partial)
$\Delta_\alpha,\ \Delta_\beta$	$190°,\ 1°$	$60°,\ 1°$
$n_\alpha,\ n_\beta$	$133,\ 5$ points	$241,\ 5$ points
$\delta_\alpha = \Delta_\alpha / n_\alpha,\ \delta_\beta = \Delta_\beta / n_\beta$	$1.43°,\ 0.2°$	$0.25°,\ 0.2°$
$r_{min},\ r_{max}$	$0.3,\ 20m$	$0.3,\ 20m$

Table D.2 outlines the derived parameters of the models. Leuze device has an horizontal aperture greater than 180° and that poses numerical problems on computing equation 4.8. This issue is overcome by dividing the computation in two scanning sectors, each one with the half of sensor's aperture, so the parameters given in table D.2 in the Leuze column are for a single scanning sector.

Table D.2: Derived model parameters for laser scanner models

Derived Parameter	Leuze RS4 (per scanning sector)	Hokuyo UTM 30-LX (partial)
$w\ [m]$	0.655	0.346
$h\ [m]$	0.005	0.005
$\rho\ [.]$	125	66
$p_\alpha\ [pixels]$	88	265
$p_\beta\ [pixels]$	5	5

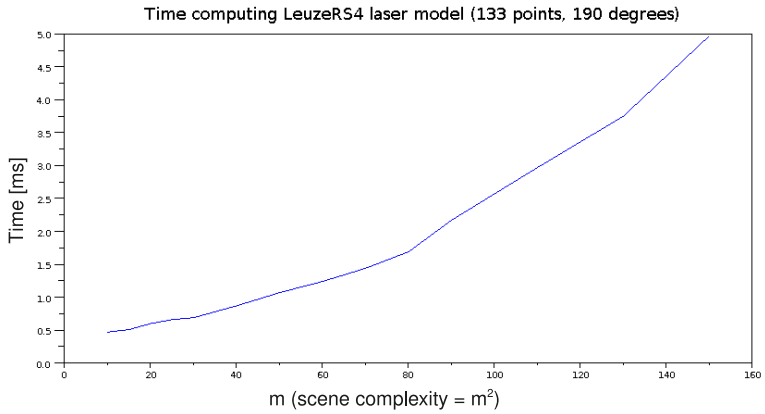


Figure D.1: Time performance versus scene complexity for the Leuze RS4 laser scanner. The sphere object has m^2 elements.

To evaluate the computational perfomance of the proposed implementation while increasing the scene complexity, we have done a set of experiments consisting on computing 100 times the Leuze model against a testbench environment composed of a single sphere, while increasing the number of sectors and slices of that shape. The results are shown in figure D.1. For a given m, the sphere is formed by m sectors and m slices, and thus the scene has m^2 elements.

Please note that using the same software implementation, other range models of devices providing point clouds such as time-of-flight cameras or 3D laser scanners can be easily configured and computed.

Appendix E

Videos

The content of the videos referenced through the thesis is detailed in this appendix. All videos can be found on internet, at the gallery page of the author's website: `www.iri.upc.edu/people/acoromin/gallery`.

Video 1. Comparison of Basic/Asynchronous Particle Filters Section 6.3 of this thesis proposes an improved version of a basic particle filter to deal with integration of delayed data. This video compares the output position estimate of the proposed approach (blue) with the basic one (red). All sensor data is simulated and integrated by the filters in real-time. Provided sensor data is: wheel odometry (green path on the map), front and back laser scanners (up-right window), compass, gps (small green points on the map), and detections from a fixed camera network (green segments on the map).

Video 2. Localization with odometry and delayed camera network detections This video shows how the asynchronous particle filter successfully tracks the position of a robot fusing only wheel odometry and delayed detections coming from a camera network (see section 6.3). The video shows a zoom of the environment centered on the robot position estimate and plots all the particle set as small red points with a heading mark. Camera network detections are plotted as green lines from the camera point to the observation point.

Video 3. Range observation model This video demonstrates in a qualitative way the resulting data from computing range observation models by using the method proposed in chapter 4, section 4.3. On the top, a visualization of a 3D geometric world model from senor's viewpoint. At bottom-left, ranges computed by a model of the LeuzeRS4 laser scanner. At the bottom-right, depth image computed with a model of the time-of-flight camera SR4000.

Video 4. Autonomous navigation with 2D localization in the UPC Campus A full autonomous navigation experiment is reported in this video. Map-based position tracking used a 2D representation of the environment and the robot state was a pose in 2D, i.e. (x, y, θ). The particle filter was fusing wheel odometry (green path on the map) and front and back laser scanner data (up-right window). A zoom on the map shows the evolution of the particle set.

Other windows are related to navigation processes such as local planner and motion control.

Video 5. Autonomous navigation with 3D localization in the UPC Campus This video also reports a full autonomous navigation experiment, but in this case the map-based position tracking used a 3D representation of the environment and the state space was that of poses in 3D, i.e. $(x, y, z, \theta, \phi, \psi)$. From the up-left, and clockwise: 3D map view from the estimated position of the robot, hand-held camera view of the session, front and back laser ranges on the robot local coordinate plane, institutional logos, local planner (RRT) on the robot plane where red dots are front laser points and grey ones are back laser hits, particle set on the zoomed area of interest of the global map, and finally the whole global map with the odometry path in green and the estimated localization in blue.

Appendix F

About Pictures

This appendix provides details about the pictures placed at the cover page of each chapter, and discuss in a non-rigorous way their relation with each related chapter content.

Chapter 1. This picture is taken in a small village of the Baix Empordà region, called Sant Julià de Boada. The image is while opening the wooden door of the church, which is usually closed. However, a neighbor of the village holds the big old key that opens that door, so that interested travellers can ask him for it. The scene of opening a door suggested me the concept of introduction as an invitation to enter to a place.

Chapter 2. Each saturday morning, in the Place Saint Etienne in the city of Toulouse, old book sellers met to put their stands in a relaxed and familiar way. The image of a stack of old books, that however still have a great value leads to a good visual representation for the state of the art.

Chapter 3. This chapter talks about sensors, and these hands placed on the water surface of the little river Riera de Fontscalentes, near the village of Castellterçol, were providing a lot of sensory information about the temperature, velocity or density of that clean water going down after some nice spring rainings.

Chapter 4. May be, the most certain observation model that humans have is that of seeing how each morning the day wakes up, and if there are no clouds, the sun is expected to be seen at a given point. If some day in the future we don't observe the sunlight raising in the morning, either we are in a polar region or we must revise our observation model! The picture was taken in the top of a peak called Puigsacalm, after passing a cold night there, so that the sunrising was widely expected.

Chapter 5. The city of Toulouse celebrated the first flight of the A380 airplane in 2005. Thousdands of people met in the city downtown to see the big engine flying over the city. A set of colorful balloons were raised, and they evoluted in the air shaping a distribution that some physical model could explain. This

set of balloons forming a draw of some probabilistic distribution had a lot of relation with a set of particles in the particle filter algorithm.

Chapter 6. On a snowed surface, the footprints after a passage of a person leave a visualization of the ground truth position tracking data. The picture was taken in March, on the surroundings of the peak Tossa Plana de Lles, located in the Pirineus.

Chapter 7. Hikers climbing mountains need global localization, even in terrains where local planning is also highly required. Global localization helps to solve the lost situation, seeing to the map and accumulating as most evidences as possible to converge to a single hypothesis. The picture was taken close to the mountain refuge of the Mulleres peak, also located in the Pirineus.

Chapter 8. This enormous and flowering tree is a nice exemplification on how thousands of small parts can build an upper level entity with such wonderful appeal. The marvel was at the begining of May in a park of the city of Copenhaguen.

Chapter 9. When old people are sit down regarding the horizon, they can have a moment to look to the past and think some conclusions about their life and their works. Probably, they have also future plans, issued from their conclusions, plans that will be performed by themselves, passed to next generations or simply kept in their minds. The picture was taken in an autumn day, close to the village of L'Estany (El Bages), and the old man was relaxing after having picked from the forest a full basket of mushrooms.

Last Picture. The author of the thesis was catched working hard during a demonstration where the robot didn't behave as expected ...

Bibliography

[1] H. Andreasson, T. Duckett, and A.J. Lilienthal. A minimalistic approach to appearance-based visual slam. *Robotics, IEEE Transactions on*, 24(5):991 –1001, October 2008.

[2] T. Arai, E. Pagello, and L.E. Parker. Editorial: Advances in Multi-Robot Systems. *Transactions on Robotics and Automation*, 18(5):655–661, October 2002.

[3] K.O. Arras, J.A. Castellanos, M. Schilt, and R. Siegwart. Feature-based multi-hypothesis localization and tracking using geometric constraints. *Journal of Robotics and Autonomous Systems*, 44:41–53, 2003.

[4] K.O. Arras, R. Philippsen, N. Tomatis, M. de Battista, M. Schilt, and R. Siegwart. A Navigation Framework for Multiple Mobile Robots and its Application at the Expo.02 Exhibition. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan. September, 2003.

[5] S.M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *Transactions on Signal Processing*, 50(2):174–188, February 2002.

[6] T. Bailey, E.M. Nebot, J.K. Rosenblatt, and H.F. Durrant-White. Data Association for Mobile Robot Navigation: A Graph Theoretic Approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, USA. April, 2000.

[7] M. Barbosa and M. Ransan. URUS Communication Protocol. Technical report, April 2009.

[8] P. Beeson, A. Muraka, and B. Kuipers. Adapting Proposal Distributions for Accurate, Efficient Mobile Robot Localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, USA. May, 2006.

[9] S. Bhuvanagiri and K.M. Krishna. Intelligent robots and systems, 2008. iros 2008. ieee/rsj international conference on. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nice, France. October, 2008.

[10] S. Bhuvanagiri and K.M. Krishna. Motion in ambiguity: Coordinated active global localization for multiple robots. *Robotics and Autonomous Systems*, 58(4):399–424, 2010.

[11] J-L. Blanco, J. González, and J-A. Fernández-Madrigal. Optimal Filtering for Non-parametric Observation Models: Applications to Localization and SLAM. *The International Journal of Robotics Research*, 29(14):1726–1742, 2010.

[12] M. Boccadoro, F. Martinelli, and S. Pagnottelli. Constrained and quantized kalman filtering for an rfid robot localization problem. *Autonomous Robots*, 29:235–251, 2010. 10.1007/s10514-010-9194-z.

[13] J. Buhmann, W. Burgard, A.B. Cremers, D. Fox, T. Hofmann, F.E. Schneider, J. Strikos, and S. Thrun. The Mobile Robot RHINO. *AI Magazine*, 16(2):31–38, 1995.

[14] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulza, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot . *Artificial Intelligence*, 114(1-2):3–55, 1999.

[15] V. Caglioti, A. Citterio, and A. Fossati. Cooperative, distributed localization in multi-robot systems: a minimum-entropy approach. In *Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS)*, Prague, Czech Republic. June, 2006.

[16] CARMEN. Carmen (the carnegie mellon robot navigation toolkit) website. `carmen.sourceforge.net`.

[17] A.R. Cassandra, L.P. Kaelbling, and J.A Kurien. Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Osaka, Japan. November, 1996.

[18] M. Chen, T. Sohn, D. Chmelev, D. Haehnel, J. Hightower, J. Hughes, A. LaMarca, F. Potter, I. Smith, and A. Varshavsky. Practical Metropolitan-Scale Positioning for GSM Phones. In *Proceedings of the International Conference of Ubiquitous Computing (UbiComp)*, California, USA. September, 2006.

[19] A. Corominas Murtra and J.M. Mirats Tur. Map Format for Mobile Robot Map-based Autonomous Navigation. Technical Report IRI-DT 2007/01, Institut de Robòtica i Informàtica Industrial, Barcelona, Spain, February 2007.

[20] A. Corominas Murtra, J.M. Mirats Tur, O. Sandoval, and A. Sanfeliu. Real-time Software for Mobile Robot Simulation and Experimentation in Cooperative Environments. In *Proceedings of the Simulation, Modelling and Programming for Autonomous Robots (SIMPAR).*, pages 135–146, Lecture Notes on Artificial Intelligence, Num. 5325. Springer ed. ISBN 978-3-540-89075-1. Venice, Italy. November, 2008.

[21] A. Corominas Murtra, J.M. Mirats Tur, and A. Sanfeliu. Efficient Active Global Localization for Mobile Robots Operating in Large and Cooperative Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, USA. May, 2008.

[22] A. Corominas Murtra, J.M. Mirats Tur, and A. Sanfeliu. Integrating Asynchronous Observations for Mobile Robot Position Tracking in Cooperative Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Saint Louis, USA. October, 2009.

[23] A. Corominas Murtra, J.M. Mirats Tur, and A. Sanfeliu. Action Evaluation for Mobile Robot Global Localization in Cooperative Environments. *Journal of Robotics and Autonomous Systems, Special Issue on Network Robot Systems*, 56:807–818, 2008.

[24] A. Corominas Murtra, E. Trulls, J.M. Mirats Tur, and A. Sanfeliu. Efficient Use of 3D Environment Models for Mobile Robot Simulation and Localization. In *Proceedings of the Simulation, Modelling and Programming for Autonomous Robots (SIMPAR).*, pages 461–472, Lecture Notes on Artificial Intelligence, Num. 6472. Springer ed. ISBN 978-3-642-17318-9. Darmstadt, Germany. November, 2010.

[25] A. Corominas Murtra, E. Trulls, O. Sandoval, J. Pérez-Ibarz, D. Vasquez, J.M. Mirats Tur, M. Ferrer, and A. Sanfeliu. Autonomous Navigation for Urban Service Mobile Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan. October, 2010.

[26] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Loalization for Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Detroit, USA. May, 1999.

[27] G.N. DeSouza and A.C. Kak. Vision for Mobile Robot Navigation: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2), February 2002.

[28] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Science, 2001.

[29] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99 –110, 2006.

[30] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer, IEEE Computer Society Press*, 22(6), 1989.

[31] A. Farinelli, R. Farinelli, L. Iocchi, and D. Nardi. Multi-robot systems: A classification focused on coordination. *IEEE Transactions on Systems, Man, Cybernetics. B*, 34:2015–2028, 2004.

[32] D. Fox. Adapting the Sample Size in Particle Filters through KLD-Sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003.

[33] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A Probabilistic Approach to Collaborative Multi-Robot Localization. *Autonomous Robots. Special Issue on Heterogeneous Multi-Robot Systems*, 8(3):325–344, June 2000.

[34] D. Fox, W. Burgard, and S. Thrun. Active Markov Localization for Mobile Robots. *Robotics and Automous Systems*, 25(3-4):195–207, 1998.

[35] D. Fox, W. Burgard, and S. Thrun. Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[36] B. Gärtner. Fast and robust smallest enclosing balls. In *Proceedings of the European Symposium on Algorithms (ESA)*, Prague, Czech Republic. July, 1999.

[37] A. Gasparri, S. Panzieri, F. Pascucci, and G. Ulivi. A Hybrid Active Global Localisation Algorithm for Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy. April, 2007.

[38] A. Georgiev and P.K. Allen. Localization methods for a mobile robot in urban environments. *IEEE Transactions on Robotics and Automation*, 20(5), October 2004.

[39] M. Göller, F. Steinhardt, T Kerscher, J.M. Zllner, and R. Dillmann. Robust navigation system based on RFID transponder barriers for the interactive behavior-operated shopping trolley (InBOT). *Industrial Robot: An International Journal*, 36(4):377 – 388, 2009.

[40] A. Harrison and P. Newman. High quality 3d laser ranging under general vehicle motion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, USA. May, 2008.

[41] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3946 –3952, Nice, France. October, 2008.

[42] IRI Robotics Laboratory. Website of the IRI robotics laboratory software infrastructure. `wikiri.upc.es/index.php/Robotics_Lab`.

[43] P. Jensfelt and S. Kristensen. Active Global Localisation for a Mobile Robot Using Multiple Hypothesis Tracking. *IEEE Transactions on Robotics and Automation*, 17(5), October 2001.

[44] E. Kaplan and C. Hegarty. *Understanding GPS: Principles and Applications, Second Edition*. Artech House, 2006.

[45] A. Kelly. Linearized error propagation in odometry. *The International Journal of Robotics Research*, 23(2):179, 2004.

[46] J. Kelly and G.S. Sukhatme. Visual-Inertial Sensor Fusion: Localization, Mapping and Sensor-to-Sensor Self-Calibration. *The International Journal of Robotics Research*, 2010.

[47] B. Kuipers and Yung-TAi Byun. A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. *Journal of Robotics and Autonomous Systems*, (8):47–63, 1991.

[48] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, USA. April, 2004.

[49] C. Kwok, D. Fox, and M. Meila. Adaptive Real-Time Particle Filters for Robot Localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan. September, 2003.

[50] T-B. Kwon, J-B. Song, and S-H Joo. Elevation Moment of Inertia: A New Feature for Monte Carlo Localization in Outdoor Environment with Elevation Map. *Journal of Field Robotics*, 3(27):371–386, 2010.

[51] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous Rover Navigation on Unknown Terrains: Functions and Integration. *The International Journal of Robotics Research*, 21(10-11):917–942, 2002.

[52] A. Lankenau, T. Röfer, and B. Krieg-Brückner. Self-Localization in Large-Scale Environments for the Bremen Autonomous Wheelchair. In C Freksa, W Brauer, C. Habel, and Wender K.F., editors, *Spatial Cognition III. Lecture Notes in Artificial Intelligence 2685*, pages 34–61. Heidelberg, Springer, 2003.

[53] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[54] J-S. Lee and W. K. Chung. Robust mobile robot localization in highly non-static environments. *Autonomous Robots*, (29):1–16, 2010.

[55] J. Levinson, M. Montemerlo, and S. Thrun. Map-Based Precision Vehicle Localization in Urban Environments. In *Proceedings of the Robotics: Science and Systems Conference*, Atlanta, USA. June, 2007.

[56] K. Lingemann, A. Nüchter, J. Hertzberg, and H. Surmann. High-speed laser localization for mobile robots. *Robotics and Autonomous Systems*, (51), 2005.

[57] E. Lu, F. anf Milios. Robot pose estimation in unknown environments by Matching 2D Range Scans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA. 1994.

[58] T. Luettel, M. Himmelsbach, F.V. Hundelshausen, M. Manz, A. Mueller, and H.-J. Wuensche. Autonomous offroad navigation under poor gps conditions. In *Proceedings of the IEEE/RSJ IROS Workshop Planning, Perception and Navigation for Intelligent Vehicles.*, Saint Louis, USA. October, 2009.

[59] J. Maantay and J. Ziegler. *GIS for the urban environment*. Environmental Systems Research Institute, 2005.

[60] K. Macek, D.A. Vasquez Govea, T. Fraichard, and R. Siegwart. Towards safe vehicle navigation in dynamic urban scenarios. *Automatika*, November 2009.

[61] R. Madhavan, K. Fregene, and L.E. Parker. Distributed Heterogeneous Outdoor Multi-robot Localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC. May, 2002.

[62] M. Maimone, Y. Cheng, and L. Matthies. Two Years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 3(24):169–186, 2007.

[63] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, USA. May, 2010.

[64] B. Mazzolai, V. Mattoli, C. Laschi, P. Salvini, G Ferri, G. Ciaravella, and P. Dario. Networking and Cooperating Robots for Urban Hygiene: the EU funded DustBot project. In *Proceedings of the Inetrnational Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Seoul, Korea. November20-22, 2008.

[65] E. Menegatti, A. Pretto, A. Scarpa, and E. Pagello. Omnidirectional vision scan matching for robot localization in dynamic environments. *Robotics, IEEE Transactions on*, 22(3):523 – 535, 2006.

[66] E. Menegatti, M. Zoccarato, E. Pagello, and H. Ishiguro. Image-based Monte Carlo localisation with omnidirectional images. *Journal of Robotics and Autonomous Systems*, (48):17–30, 2004.

[67] A. Milella, B. Nardelli, D. Di Paola, and G. Cicirelli. Robust Feature Detection and Matching for Vehicle Localization in Uncharted Environments. In *Proceedings of the IEEE/RSJ IROS Workshop Planning, Perception and Navigation for Intelligent Vehicles.*, Saint Louis, USA. October, 2009.

[68] A. Milella, F. Pont, and R. Siegwart. Model-Based Relative Localization for Cooperative Robots Using Stereo Vision. In *Proceedings of the IEEE International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Manila, Philippines. November - December, 2005.

[69] J.M. Mirats Tur, J.L. Gordillo, and C.A. Borja. A closed-form expression for the uncertainty in odometry position estimate of an autonomous vehicle. *IEEE Transactions on Robotics*, 21(5):1017–1022, 2005.

[70] J.M. Mirats Tur, C. Zinggerling, and A. Corominas Murtra. Geographical information systems for map based navigation in urban environments. *Robotics and Autonomous Systems*, 57(9):922–930, 2009.

[71] Y. Morales, A. Carballo, E. Takeuchi, A. Aburadani, and T. Tsubouchi. Autonomous robot navigation in outdoor cluttered pedestrian walkways. *Journal of Field Robotics*, 26(8):609–635, 2009.

[72] A.C. Murillo, C. Sagüés, J.J. Guerrero, T. Goedemé, T. Tuytelaars, and L. Van Gool. From omnidirectional images to hierarchical localization. *Robotics and Autonomous Systems*, 55:372–382, 2007.

[73] R. M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press, 1994.

[74] U. Nagarajan, G. Kantor, and R.L. Hollis. Trajectory Planning and Control of an Underactuated Dynamically Stable Single Spherical Wheeled Mobile Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan. May, 2009.

[75] P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schrter, L. Murphy, W. Churchill, D Cole, and I. Reid. Navigating, recognising and describing urban spaces with vision and laser. *The International Journal of Robotics Research*, 28, October 2009.

[76] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1):3–20, 2006.

[77] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH: An office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

[78] S. Nuske, J. Roberts, and G. Wyeth. Robust Outdoor Visual Localization Using a Three-Dimensional-Edge Map. *Journal of Field Robotics*, 9(26):728–756, 2009.

[79] OBJfile. *OBJ* file format specification. `local.wasp.uwa.edu.au/~pbourke/dataformats/obj/`.

[80] J.M. O'Kane. Global localization using odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, USA. May, 2006.

[81] C.F. Olson. Probabilistic Self-Localization for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 16(1), February 2000.

[82] S. Oluf and M. Vincze. . In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Saint Louis, USA. October, 2009.

[83] openGL. Opengl website. `www.opengl.org`.

[84] openRTM. Openrtm website. `www.openrtm.org`.

[85] OpROS. Opros website. `www.opros.or.kr`.

[86] ORCA. Orca-robotics project. `orca-robotics.sourceforge.net/`.

[87] OROCOS. Orocos-open robot control software. `www.orocos.org`.

[88] E. Pagello, A. D'Angelo, and E. Menegatti. Cooperation issues and distributed sensing for multi-robot systems. In *IEEE Proceedings of IEEE*, volume 94, pages 1370–1383, July 2006.

[89] I. Parra, M.A. Sotelo, and L. Vlacic. Robust visual odometry for complex urban environments. In *IEEE Intelligent Vehicles Symposium*, pages 440–445, June 2008.

[90] PLAYER/STAGE. Player-stage project. `playerstage.sourceforge.net/`.

[91] J.M. Porta, J.J. Verbeek, and B.J.A. Kröse. Active Appearance-Based Robot Localization Using Stereo Vision. *Autonomous Robots*, 18(1):59–80, 2005.

[92] S. Premvuti and J. Wang. Relative position localizing system for multiple autonomous mobile robots in distributed robotic system : System design and simulation. *Journal of Robotics and Autonomous Systems*, 18(3), August 1996.

[93] M. Raibert, K. Blankespoor, G. Nelson, R. Playter, and BigDog Team. BigDog, the Rough-Terrain Quadruped Robot. In *Proceedings of the World Congress of the International Federation of Automatic Control*, Seoul, Korea, July 6-11, 2008.

[94] M. Rao, G. Dudek, and S. Whitesides. Randomized Algorithms for Minimum Distance Localization. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Utrecht/Zeist, The Netherlands. July, 2004.

[95] F. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, J-M. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier, J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Gülke, F. Klose, and B. Rumpe. Caroline: An autonomously driving vehicle for urban environments. *Journal of Field Robotics*, 25(9):674–724, September 2008.

[96] I.M. Rekleitis, G. Dudek, and E.E. Milios. Graph-Based Exploration using Multiple Robots. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Knoxville, Tennessee, USA. October, 2000.

[97] I.M. Rekleitis, G. Dudek, and E.E. Milios. Multi-Robot Cooperative Localization: A Study of Trade-offs Between Efficiency and Accuracy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland. September, 2002.

[98] C. Röhrig and F. Künemund. Mobile Robot Localization using WLAN Signal Strengths. *International Journal of Computing*, 7(2):78–83, 2008.

[99] ROS. Ros (robotic operation system) website. `www.ros.org`.

[100] S.I. Roumeliotis and G.A. Bekey. Distributed Multi-Robot Localization. *Transactions on Robotics and Automation*, 18(5):781–795, October 2002.

[101] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland. September, 2002.

[102] A. Sanfeliu and J. Andrade-Cetto. Ubiquitous networking robotics in urban settings. In *Proceedings of the IEEE/RSJ IROS Workshop on Network Robot Systems*, Beijing, China. October, 2006.

[103] D. Scaramuzza, A. Martinelli, and R. Siegwart. Precise Bearing Angle Measurement Based on Omnidirectional Conic Sensor and Defocusing. In *Proceedings of the European Conference on Mobile Robots*, Ancona, Italy. September, 2005.

[104] R. Simmons and S. Koenig. Probabilistic Robot Navigation in Partially Observable Environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Canada. August, 1995.

[105] D. Simon. *Optimal State Estimation*. John Wiley & Sons, Inc., 2006.

[106] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet Romea, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and J.M. Vandeweghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, January 2010.

[107] R. Thrapp, C. Westbrook, and D. Subramanian. Robust localization algorithms for an autonomous campus tour guide. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Korea. May, 2001.

[108] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: a second-generation museum tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Detroit, USA. May, 1999.

[109] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[110] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141, 2001.

[111] E. Trulls, A. Corominas Murtra, J. Pérez-Ibarz, G. Ferrer, D. Vasquez, Josep M. Mirats-Tur, and A. Sanfeliu. Autonomous navigation for mobile service robots in urban pedestrian environments. *Journal of Field Robotics*, 28(3):329–354, 2011.

[112] URUS. Urus website. `urus.upc.edu`.

[113] R. Valencia, E.H. Teniente, E. Trulls Fortuny, and J. Andrade-Cetto. 3D mapping for urban service robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Saint Louis, USA. October, 2009.

[114] N. Vlassis, B. Terwijn, and B. Kröse. Auxiliary Particle Filter Robot Localization from High-Dimensional Sensor Observations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Washington, DC. May, 2002.

[115] J. Wolf, W. Burgard, and H. Burkhardt. Robust Vision-based Localization by Combining an Image Retrieval System with Monte Carlo Localization. *IEEE Transactions on Robotics*, 21(2), 2005.

[116] YARP. Yarp (yet another robot platform) website. `eris.liralab.it/yarp/`.

[117] J. Yun and J. Miura. Multi-Hypothesis Outdoor Localization using Multiple Visual Features with a Rough Map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy. April, 2007.

[118] H. Zhou and Sakane S. Mobile robot localization using active sensing based on Bayesian network inference. *Journal of Robotics and Autonomous Systems*, (55):292–305, 2007.

[119] H. Zhou and Sakane S. Sensor Planning for Mobile Robot Localization - A Hierarchical Approach Using a Bayesian Network and a Particle Filter. *IEEE Transactions on Robotics and Automation*, 24(2):481–487, April 2008.