Center for Research and Advanced Studies of the
National Polytechnic Institute
Department of Automatic Control

Map-building and planning for autonomous navigation of a mobile robot

Thesis submitted by:

Erik Zamora Gómez

A thesis submitted in the fulfillment for the degree of:

Doctor of Philosophy

In the specialization of

Automatic Control

Thesis supervisor:

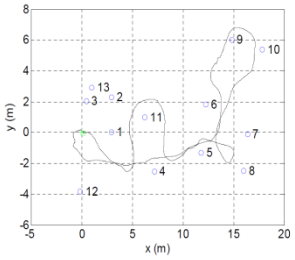
Dr. Wen Yu Liu

México, D.F., January 2015

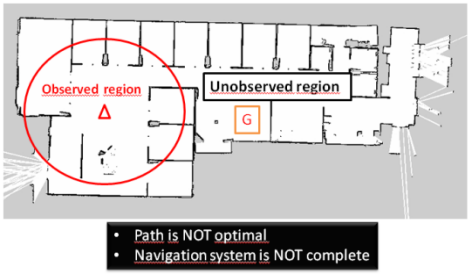
ABSTRACT

The autonomous navigation of robots in uncontrolled environments is a challenge because it requires a set of subsystems to work together. It requires building a map of the environment, localizing the robot in that map, making a motion plan according to the map, executing that plan with a controller, and other tasks; all at the same time. On the other hand, the autonomous navigation problem is very important for the future of robotics. There are many applications that require this problem to be solved, such as package delivery, cleaning, agriculture, surveillance, search & rescue, construction and transportation. Note that all these applications occur in uncontrolled environments. In this thesis, we try to solve some sub-problems related to autonomous navigation in uncontrolled environments.

For map-building, we propose the first method based on ellipsoidal sets that can solve large-scale problems, unlike other methods based on sets. The errors are modeled by ellipsoid sets without assuming Gaussianity, so we have more realistic error modeling. Its performance is similar to, and in some cases better than, the method based on a Kalman filter. Some real experiments show it is capable of building maps indoors and outdoors.



For planning, we introduce and study the fundamental concepts of Known Space, Free Space and Unknown Space Assumptions to deal with the partial observability of environment. These assumptions are needed to navigate in dynamic and unknown environments. **For dynamic environments,** we propose the first two methods to recover from false obstacles in the map and make the robot continue searching for its goal. **For unknown environments,** we propose a method to navigate more efficiently. We ran Monte Carlo simulations to evaluate the performance of our algorithms. The results show in which conditions our algorithms perform better and worse.



Contents

List of Figures

Chapter 1

Introduction

"Ask not what your country can do for you; ask what you can do for your country." —John F Kennedy

Why is it important to develop autonomous navigation robots? Basically, there are three kinds of robots: operated, automatic and autonomous. Operated robots are those that require control by a human, e.g. teleoperated robots for surgery or army exploration. Automatic robots do preprogrammed and repeated activities in controlled environments, e.g. robotic arms in car production lines or line follower robots. In contrast, autonomous robots do tasks in unstructured environments and make their own decisions as a function of the given goal, e.g. courier robots in hospitals and driverless cars in cities. **The tendency is to give robots more autonomy, which means robots do tasks with as little human assistance as possible.**

Autonomous robots could catapult the productivity and quality of various human activities. Some applications are: package delivery, cleaning, agriculture, surveillance, search & rescue, building and transportation. However, a basic task that robots must do is to navigate in natural and human environments in order to achieve these applications. If someday we wish to have robots that build our highways, clean our streets, and grow and harvest our food, it is crucial they be able to navigate in unstructured environments. The mission of this thesis is to contribute to this goal: **solving some problems of autonomous navigation**

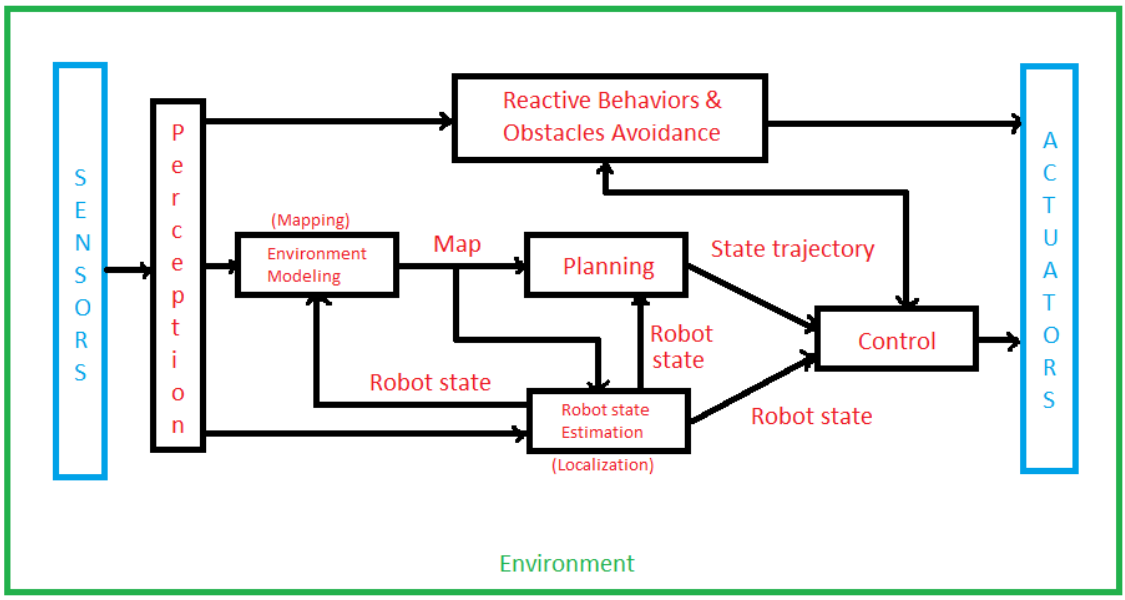


Figure 1.1: Subsystems needed to solve the whole autonomous navigation problem.

in unstructured environments.

Autonomous navigation is the capacity of a robot to navigate with as little human assistance as possible. *Unstructured environment* is an environment that is NOT adapted to facilitate robot navigation, but is given as it is; e.g. offices, parks, streets, forests, desert, etc. Navigation in unstructured environments is a problem with many variants. The technical difficulties to be overcome to navigate depend on the type of environment: in air, on land, at sea or under the sea. The type of locomotion also ims restrictions: wheels, legs, propellers, etc. In this thesis, we limit our work to robots in terrestrial environments with flat surfaces and we use only wheeled robots.

Although navigation is easy for humans, it is a difficult task for robots. Because they must navigate in unstructured environments without damage and should use several subsystems to solve the problem. Figure 1.1 shows an autonomous navigation system consisting of:

- **Perception**: Interprets the numbers sent by sensors to recognize objects, places, and

events that occur in the environment, or in the robot. In this way, the robot can prevent damage, know where it is, or know how the environment is.

- **Map building**: Creates a numerical model of the environment around the robot. The map allows the robot to make appropriate decisions and avoid damage.
- **Location**: Estimates the robot's position relative to the map. This helps it plan and execute movements, and build a correct map of the environment.
- **Planning**: Decides the movements necessary to reach the goal without colliding and in minimum time or distance.
- **Control**: Ensures the planned movements are executed, despite unexpected disturbances.
- **Obstacle avoidance**: Avoids crashing into moving objects, such as people, animals, doors, furniture or other robots that are not on the map.

The contributions of this thesis are mainly related to map building and planning because we identified relevant research opportunities, as we explain in the next section.

1.1 Motivation of research

Autonomous navigation has been developing for almost fifty years and just in the last decade has some useful results (see **section 2.1**). It is too large a problem to be solved with only one study. Therefore, we decided to focus on two subproblems:

- **Map-building problem.** If we want to navigate through an environment, we need a map for two reasons: (1) to localize the robot and the goal, and (2) to plan a path between the robot's position and the goal position. But sometimes there are no maps of buildings, public spaces or cities that can be useful for navigation. Consequently, the robot must create its own map. In unstructured environments, the map building

process requires knowing the robot's position, and the robot's position estimation requires a map. So we need to locate the robot and make the map at the same time. This is why the map-building problem is so interesting and also problematic: errors in position and map estimation affect each other, producing inconsistent maps. The effectiveness and efficiency of navigation depends on having a map without large errors. Most map building methods are based on the Gaussian assumption (i.e. estimation errors are described by a Gaussian probability distribution). But this assumption is incorrect because the real error distributions are not Gaussian. So we propose and study the Ellipsoidal SLAM method[1], which avoids the use of the Gaussian distribution. Besides, we note that other methods based on sets (i.e. estimation errors modeled by sets) cannot solve large-scale mapping problems. Our method can.

- **Planning in dynamic and unknown environments.** In these types of environments, the problem of partial observability is important because planning has to make certain assumptions about the environment that cannot be or has not been observed. And these assumptions might be wrong. Even if we have a perfect mapping method, the map might have errors relative to real environment. Why is this possible? The reason is simple: environments are dynamic and all sensors have a limited range of observation. This type of problem is important to solve because it affects the optimality of the path length and completeness of navigation task (i.e. to be able to reach the goal when a solution exists). We present theorems of conditions sufficient to guarantee that a navigation system can complete its task. We propose the first methods to recover from false obstacles on the map in dynamic environments. And we propose a method to navigate more efficiently in unknown environments.

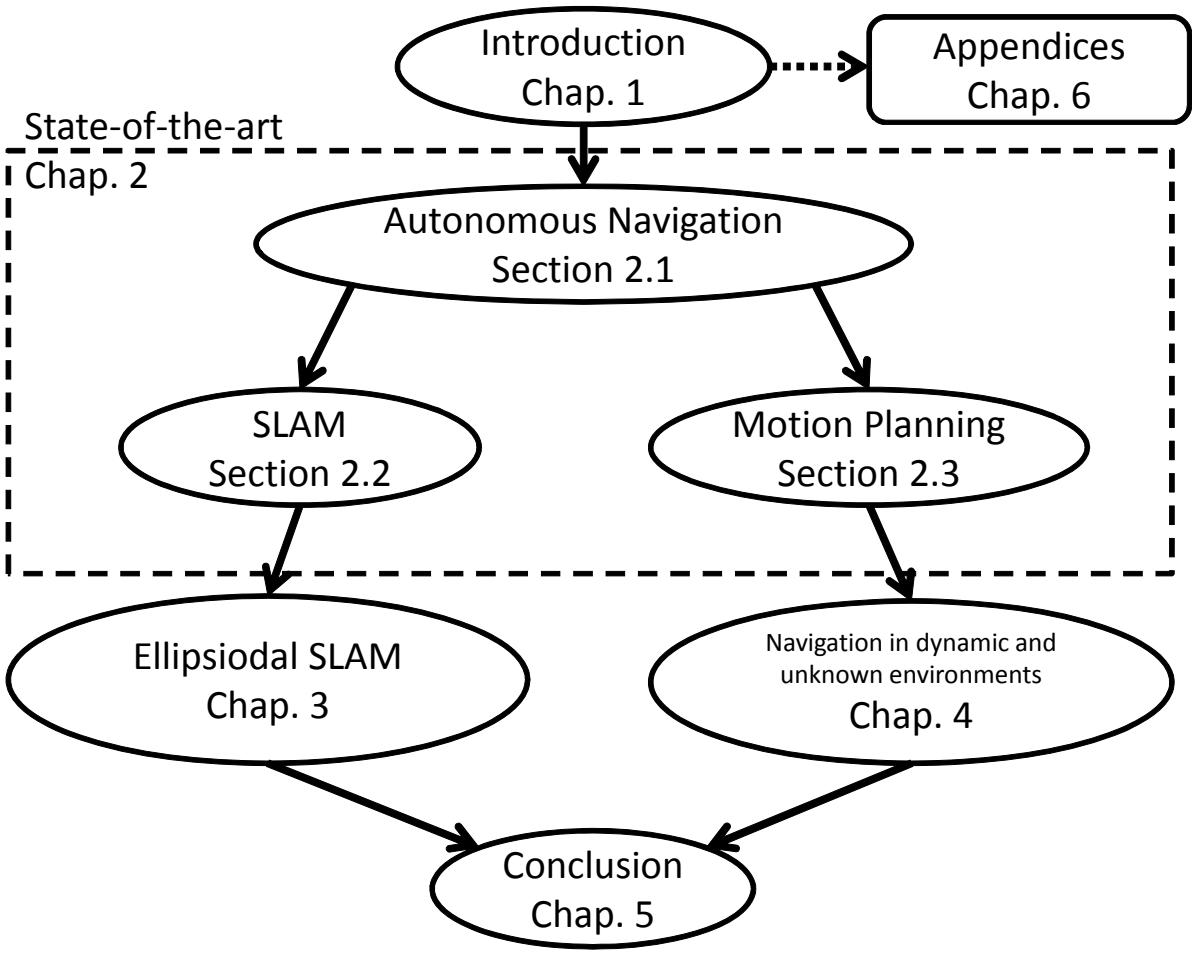


Figure 1.2: This is the general structure of the thesis.

1.2 Structure of the thesis

The thesis is organized as Figure 1.2 shows.

Chapter 2. We establish the state of the art of autonomous navigation (**section 2.1**), the map-building problem (**section 2.2**), and the planning problem **(section 2.3**). The first describes the big picture of autonomous robots, presenting the main achievements, the potential applications and the main technical problems. The second summarizes the different approaches to solve the map-building problem and shows why is important. The third explains what the planning problem is, discusses the various methods of planning and presents research work related to the problem of partial observability.

Chapter 3. We propose the first map building method based on ellipsoidal sets that can solve large-scale problems. We present the method and analyze its convergence and stability by a Lyapunov-like technique. We validate the Ellipsoidal SLAM with simulations and experiments using the Victoria Park dataset and Koala mobile robot.

Chapter 4. We analyze the assumptions that a robot could make in unobserved parts of the environment during navigation. We present the theorems of sufficient conditions to ensure a navigation system is complete. For dynamic environments, we propose the first methods to recover from false obstacles on the map. And for unknown environments, we propose a method to navigate more efficiently in unknown environments compared with the classical approach. We present Monte Carlo simulations to evaluate the performance of these algorithms and discuss their results.

Chapter 6 (Appendices). We present the two navigation systems that were implemented during the development of doctoral research. One of them was used in the experiments for the Ellipsoidal SLAM.

[1] In this thesis we consider the terms "map building" and "SLAM" synonyms, where SLAM means Simultaneous Localization And Mapping

1.3 Contributions

The main contributions of this thesis are the following:

- We propose the first map building method based on sets that can solve large-scale problems.
- We propose the first methods to recover from false obstacles and make the robot continue searching for the goal in dynamic environments.
- We propose a method to navigate more efficiently in unknown environments.

1.4 Products

1.4.1 Publications

We produce the following papers for publication:

In journals:

- Erik Zamora, Wen Yu, Recent Advances on Simultaneous Localization and Mapping (SLAM) for Mobile Robots, IETE Technical Review, 2013. (impact factor 0.925)
- Erik Zamora, Alberto Soria, Wen Yu, "Ellipsoid SLAM: A Novel Set Membership Method for Simultaneous Localization and Mapping", Autonomous Robots, Springer (in revision).
- Erik Zamora, Wen Yu, Novel Autonomous Navigation Algorithms in Dynamic and Unknown Environments, Cybernetics and Systems (in revision).
- Erik Zamora, Robots Autónomos: Navegación, Komputer Sapiens, SMIA, Enero 2015.

In conferences:

- Erik Zamora, Wen Yu, Novel Autonomous Navigation Algorithms in Dynamic and Unknown Environments, 2014 IEEE International Conference on Systems, Man, and Cybernetics, San Diego, California, USA, October 5-8, 2014.

- Erik Zamora, Wen Yu, Mobile Robot Navigation in Dynamic and Unknown Environments, 2014 IEEE Multi-conference on Systems and Control, October 8-10, 2014, Antibes, France.

- Erik Zamora, Wen Yu, Ellipsoid SLAM: A Novel On-line Set Membership Method for Simultaneous Localization and Mapping, 53rd IEEE Conference on Decision and Control, Los Angeles, California, USA, December 15-17, 2014.

1.4.2 Education of human resources

As part of teaching profession, the author of this thesis collaborated in the education and graduation of 11 engineers at the UPIITA-IPN (Unidad Interdisciplinaria Profesional en Ingeniería y Tecnologáas Avanzadas del Instituto Politecnico Nacional). Four terminal works related to autonomous robotics were done (see Figure 1.3).

- a) **Modelado y control de un cuadrirrotor para vuelo en entornos cerrados implementando visión artificial** (Modeling and control of a quadrotor flight in closed environments by implementing computer vision). They developed a system that allows navigating a quadrotor along a corridor using only visual information from a RGB camera. Anguiano Torres Iván Adrián, Garrido Reyes Miguel Daniel, Martínez Loredo Jonathan Emmanuelle, UPIITA-IPN, 2013.

- b) **Sistema de navegación evasor de obstáculos en un robot móvil** (Navigation system with an obstacles avoidance for a mobile robot). This project develops a reactive navigation system, which allows driving a mobile robot from a starting point to any point within the working area avoiding static obstacles along its path. Miguel Antonio Lagunes Fortiz, UPIITA-IPN, 2014.

- c) **Diseño, manufactura e implementación de un sistema de seguimiento de objetos mediante visión artificial para un robot humanoide NAO** (Design, manufacture and implementation of an object tracking system using artificial vision for a humanoid robot NAO). The terminal work aims to develop an embedded system for controlling movement of a humanoid, being able to locate and track an object specific color. Conde Rangel María Gisel, García Reséndiz Erick Gabriel, Serra Ramírez Jaime, Trujillo Sánchez Juan Carlos, 2013.

- d) **Implementación de visión artificial en un cuadrirrotor para el seguimiento de objetivos** (Implementation of artificial vision in quadrotor for tracking targets). This project involves the development of control algorithms for the flight of a quadrotor, taking into account the response of inertial sensors: accelerometer and gyroscope. In addition, it implements computer vision algorithms to follow a specific target on the ground. Hernández Espinosa Josué Israel, Montesinos Morales José Iván, Torres Vázquez Benjamín, 2012.

1.4.3 Autonomous navigation systems

During the research, we developed two autonomous navigation systems in order to put into practice some of the results of the thesis. Here we briefly present these systems, and we describe them in detail in the appendices of thesis.

1. **A basic autonomous navigation system**. At the beginning of the PhD program, it was necessary to develop an autonomous navigation system for robot Koala; in order to carry out experiments of the SLAM method proposed in this thesis. This robot is available in the laboratory of the Department of Automatic Control. The navigation system is comprised of four subsystems: localization, mapping, planning and controller. The localization is done by odometry that integrates the angular displacements of encoders. The mapping builds a 2-D occupancy grid map using odometry and information from a laser. The planning is conducted by the method of dynamic

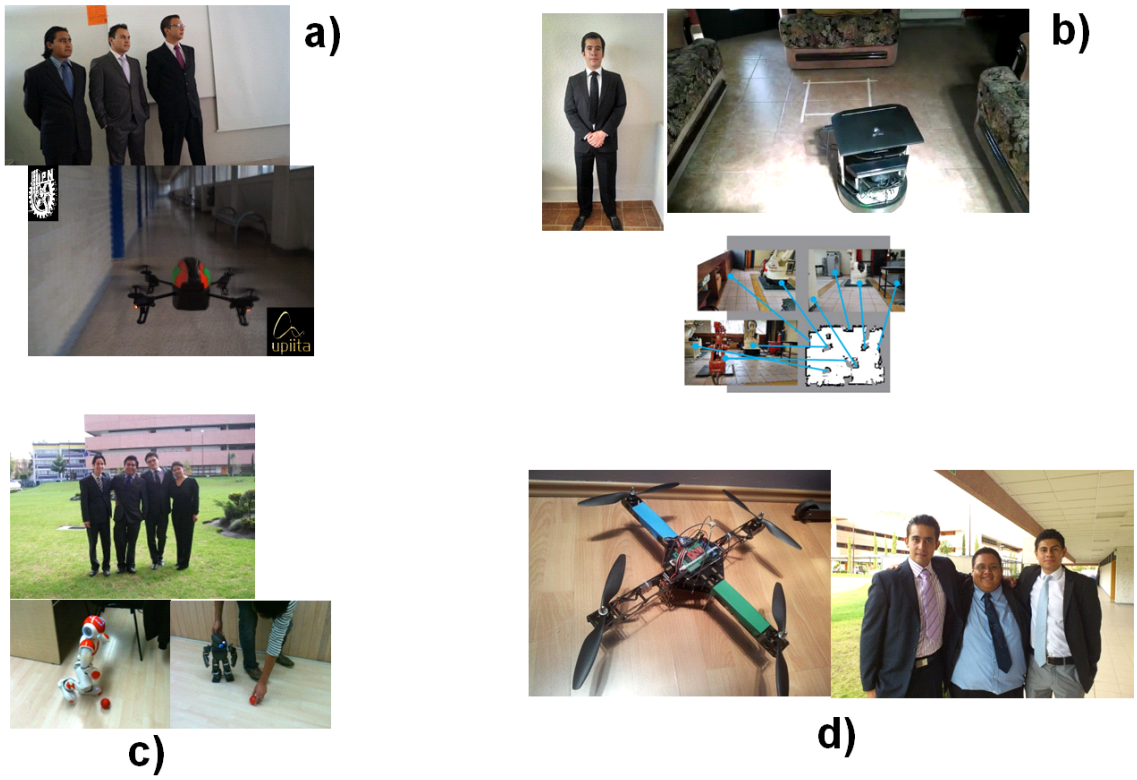


Figure 1.3: We show some snapshots of terminal projects and engineers that the author of this thesis has instructed in UPIITA-IPN during the development of his PhD research (see text for more details).

programming with a smoothing method for the generated path. The control is carried out by a nonlinear controller which follows the smoothed path. The software was written in Python and C/C++. This navigation system is very basic because it does not use a SLAM technique to correct the odometry errors on map. Therefore, this system cannot be used to navigate distances greater than 20 meters. However, it was useful for our experiments. Figure 1.4 shows how the robot navigates through a door along the planned path.

2. **Navigation in unknown environments guided by emergency signs**. As part of the PhD program, the author of this thesis participated in a research stay at the University of Bristol in England led by Walterio Mayol-Cuevas. The aim of the research stay was to develop a navigation system for a differential mobile robot (iRobot). The task was to look for the emergency exit guided by the existing emergency signs on the building. Figure 1.5 shows some examples of these emergency signs and the robot moving to the exit. The robot uses a RGBD camera (Kinect) to detect emergency signs and determine the direction that it should move. The same sensor provides information on obstacles such as chairs, walls, tables, people, etc. It uses a SLAM method (Gmapping method) to build a 2-D map as it moves; at the beginning the robot does not know the environment. It uses a planning method (A* search) to find the shortest path. This method was modified to integrate the information of emergency signs. The planned path is executed by a P controller and a method of obstacle avoidance (Smooth Nearness Diagram) was added. All software is written in Python and C/C++ on ROS platform (Robot Operating System). The most interesting part of this project was to implement the system of perception which must detect, recognize and interpret the emergency signs to know how to move. Furthermore, it was necessary to implement a system to prevent repeated detections which was coupled to the SLAM method. A video showing the operation of this navigation system is online at

   https://www.youtube.com/watch?v=RAj70AGsXls&feature=youtu.be**??**

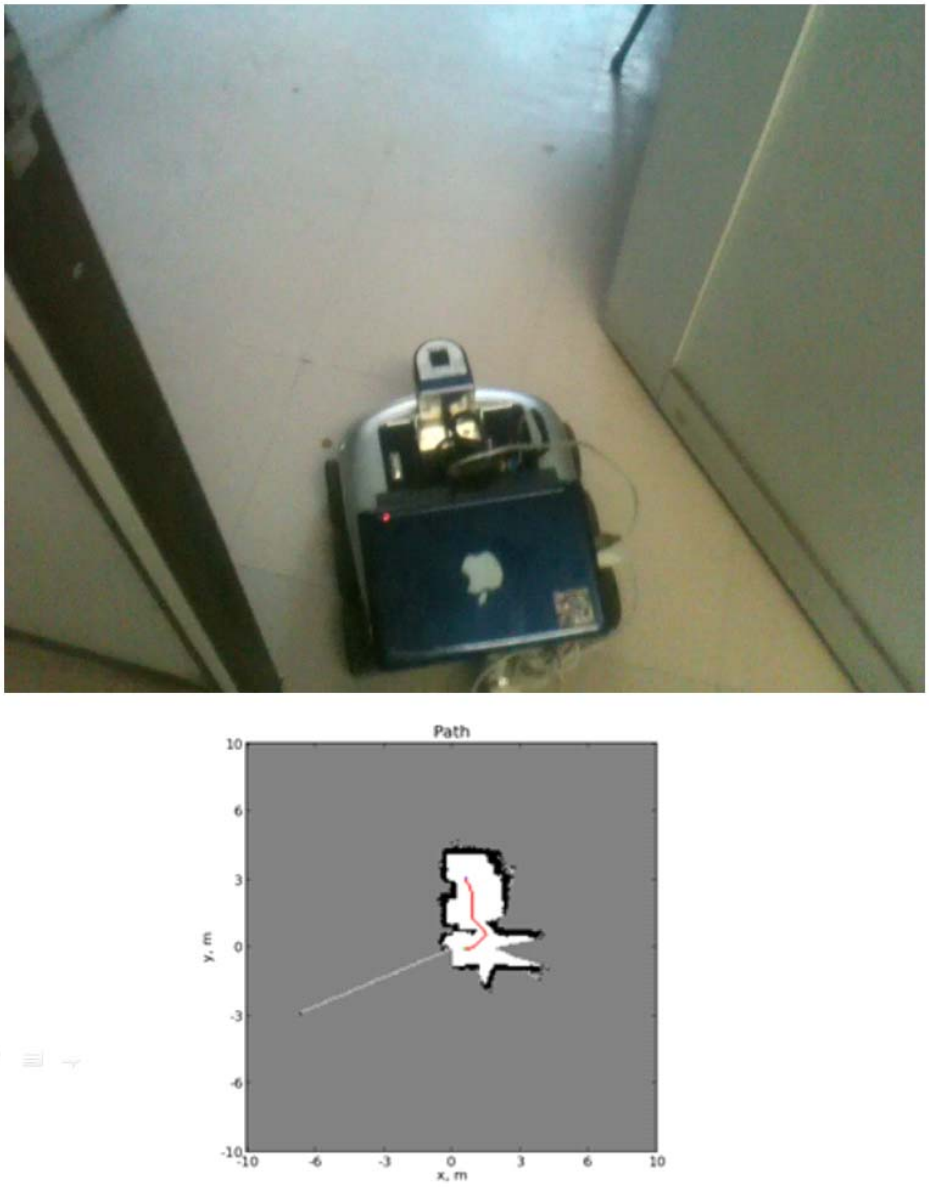


Figure 1.4: We show how the Koala robot can navigate through a door along the planned path using a map that was previously built. The map was constructed while controlling the robot manually.

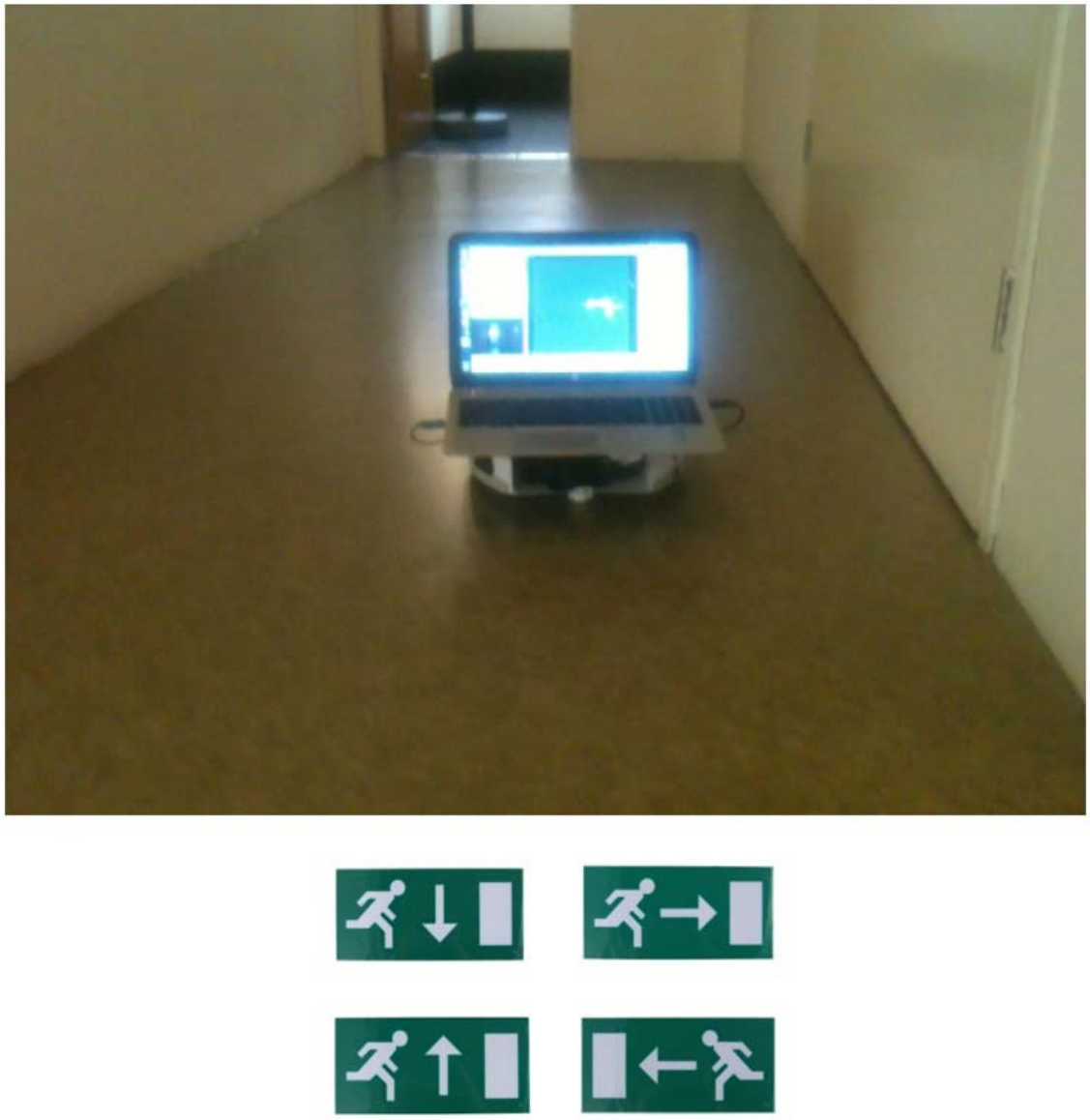


Figure 1.5: We show a snapshot of the robot reaching its goal: the emergency exit. We also show the emergency signs in the building in the University of Bristol. They were used to guide the robot to its goal.

Chapter 2

State of the art

The aim of this chapter is to show an overview of autonomous navigation, the map-building problem, and the planning problem. This chapter is divided into three main sections. The first describes the big picture of autonomous robots presenting the main achievements, the potential applications, and the main technical problems. The second summarizes the different approaches to solve the map-building problem and show why it is important. The third explains what the planning problem is, discusses the various methods of planning, and presents research work related to the problem of partial observability.

2.1 Autonomous navigation

> *"The world requires practical dreamers who can, and will, put their dreams into action." —Napoleon Hill*

Why is it important to develop autonomous navigation? Basically, there are three kinds of robots: operated, automatic, and autonomous. Operated robots are those that require control by humans (e.g., teleoperated robots for surgery or army exploration). Automatic robots do preprogrammed and repeated activities in controlled environments (e.g., robotic arms in car production lines or line follower robots). In contrast, autonomous robots do tasks in natural and unstructured environments and make their own decisions as a function

of the given task goal (e.g., courier robots in hospitals and driverless cars in cities). *The tendency is to give robots more autonomy, which means robots do tasks with as little human assistance as possible.*

A basic task that robots must do is to navigate in natural and human environments. If someday we wish to have robots that build our buildings and highways, clean our streets, grow and harvest our food, **it is crucial they are able to navigate in unstructured environments**. There have been incremental advancements in the last 15 years. In the following three subsections, we will revise some breakthroughs, mention some of the applications for autonomous robots and describe what the main problems are, respectively.

2.1.1 Breakthroughs and companies

In 2004 and 2005, several vehicles navigated autonomously through the Mojave Desert, travelling about 200 km, following a map of GPS coordinates and using a set of lasers and cameras to avoid the obstacles on the route. The robotic vehicles were built by universities and companies, and the driverless car challenge was organized by DARPA (Defense Advanced Research Projects Agency in the USA). In 2007, there was another competition, but in an urban environment, which is a harder challenge. The vehicles had to avoid crashing into cars, bikes, and pavement; to execute driving skills such as lane changes, U-turns, parking, and merging into moving traffic. The winners for the 2005 and 2007 challenges were Stanley [1] from Stanford University and Boss [2] from Carnegie Mellon University; you can see these vehicles in Figure 2.1.

Based on these results, Google Inc. has developed several autonomous cars that have been tested in cities and highways, in Nevada and California, USA, where the government already gives restricted driver license for autonomous cars. A disadvantage of Google cars is its high cost because of the 3-D sensor (Velodyne costs about $75,000) and high-precision GPS. Thus, the University of Oxford has launched the project RobotCar UK to replace this expensive sensor with cheaper lasers and cameras, using the spatial-visual information to localize the robot without GPS [3].

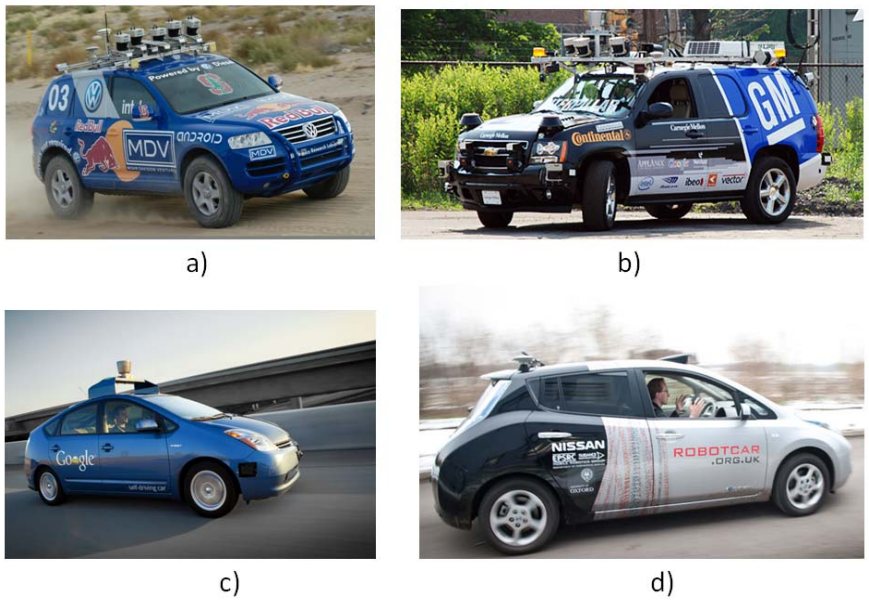


Figure 2.1: Some autonomous cars are able to navigate in deserts, cities, and highways: (a) Stanley, (b) Boss, (c) Google driverless car, (d) RobotCar UK.

Humanoid robots (see Figure 2.2) will evolve toward autonomy, but currently they show limited capabilities. Asimo has automatic skills, including jumping with only one foot, running, climbing stairs, opening bottles, pouring liquids from one glass to another. However, most activities are preprogrammed by humans and are not autonomous, so they cannot find and execute the suitable motions to solve the task by themselves. Other humanoids like Atlas, Justin, Reem, Charli or HRP-4C suffer from the same problem. But there are advancements in motion planning, learning, and perception to give them more autonomy. Some examples are the algorithms tested in PR2 [4] and iCub [5]; they can decide how to move their arms to grasp bottles without hitting the table or other objects. Due to the nuclear disaster in Fukushima, DARPA is organizing another competition, Darpa Robotics Challenge [6]. The goal is to develop algorithms that make humanoids assist humans in natural and man-made disasters. This challenge will bring new technologies to catapult forward development of autonomous robots.

Several autonomous robots exist in Mexico: Justina [7], Golem [8], Markovito [9], Donaxi [10] and Mex-One (see Figure 2.3). The first four can navigate autonomously indoors, moving toward previously visited places and avoiding obstacles. They can do some tasks, like recognizing objects, people, and voice; cleaning tables; grasping bottles; speaking some preprogrammed sentences. These robots have competed in Robocup@home, which is a competition of service robotics, where Golem won the prize for innovation in 2013. On the other hand, Mex-One will be the first Mexican biped robot, but it is still in development. It promises to be a great platform to test algorithms to transform it into an autonomous robot. Besides, there are Mexican robots that compete in RoboCup@Soccer and RoboCup Rescue; some of them do autonomous tasks, like play football or build a map inside a collapsed building.

Several companies produce autonomous robots. In the United States, three companies target infant industry (see Figure 2.4). Boston Dynamics has made a reputation with its impressive robotic mules able to travel on rough terrain. Willow Garage produces PR2, which has been tested if it can fold laundry or play billiards. But his most important contribution is continuing the development of the operating system for robots ROS (Robot Operating

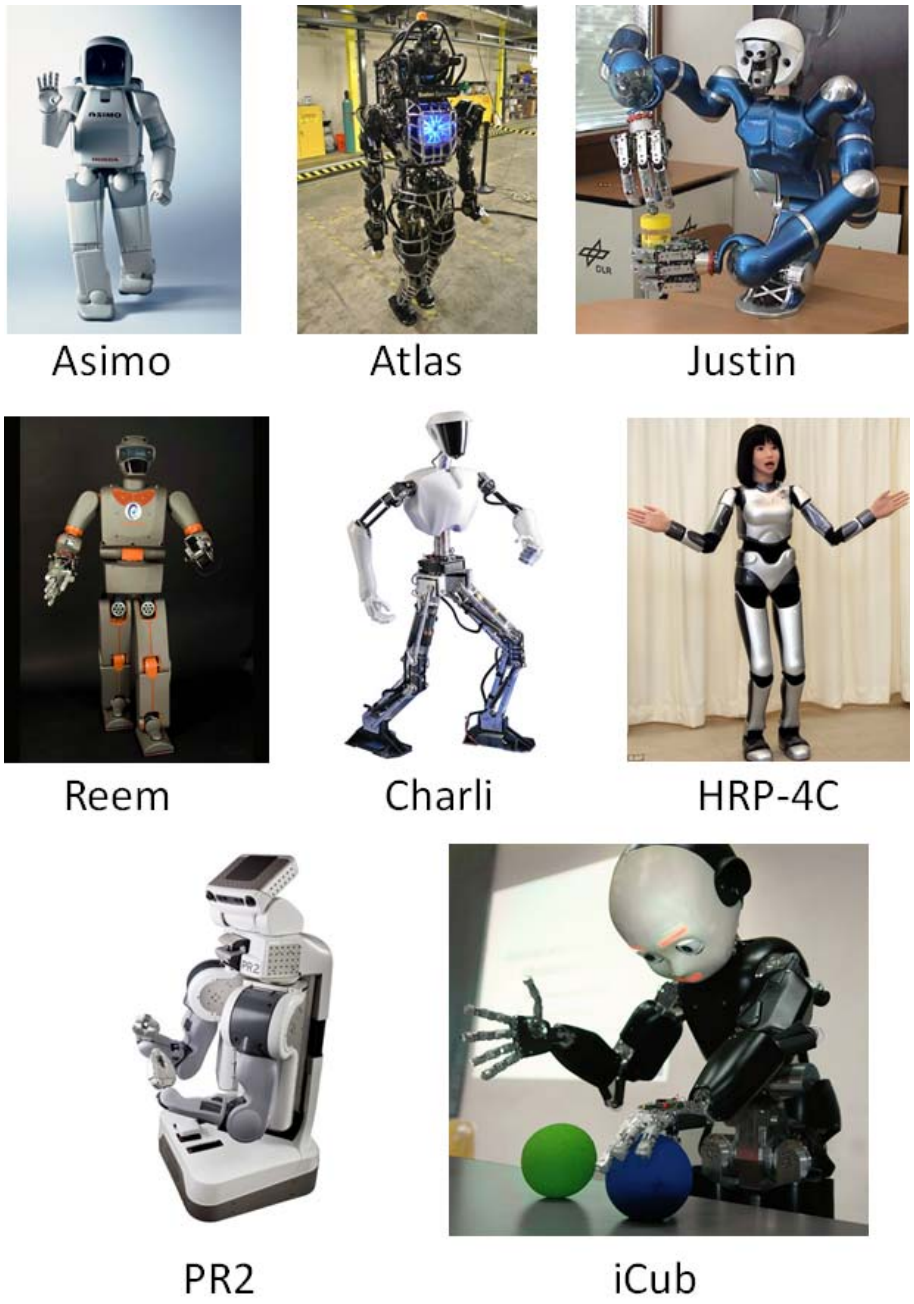


Figure 2.2: We show some humanoid robots useful for researching motion planning, perception, and learning algorithms, with the objective of giving them autonomy.

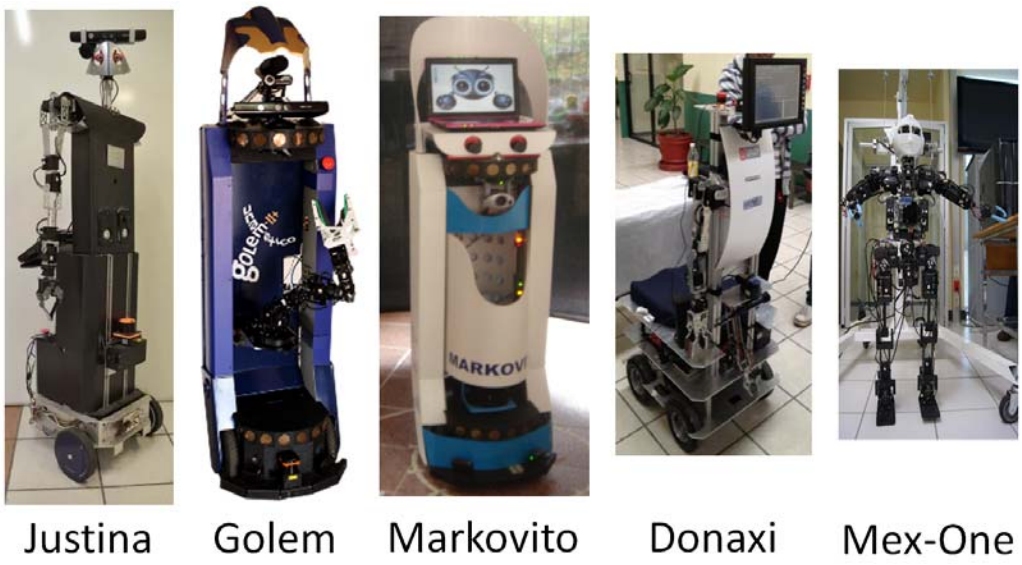


Figure 2.3: Mexican autonomous robots

System), whose license is free and supports a considerable number of commercial robots. Rethink Robotics sells Baxter, a manufacturing robot that does not require a specialist to code tasks since anyone can program it with a friendly graphical interface and moving robot arms to indicate the task. Moreover, in Germany, BlueBotics produces and sells mobile robots with autonomous navigation systems, guiding tourists in cities and museums. These are some examples of how the robotics industry moves toward autonomy.

The industry for autonomous robots is almost nil in Mexico. This is a great opportunity for investors and entrepreneurs since the market is virgin, waiting for someone to exploit it. The company, called 3D Robotics, has done this; it was cofounded by a Mexican and an American, and it produces and distributes unmanned aerial vehicles with a navigation system based on GPS. The rest of the Mexican industry sells robots and accessories, gives courses, and installs automatic robots for manufacturing.

Figure 2.4: Companies of autonomous robots.

2.1.2 Applications

Autonomous robots catapult the productivity and quality of various human activities. Here we will present some existing applications and imagine other possibilities.

- **Package delivery**. Imagine a robotic motorcycle or quadcopter that delivers pizza to your front door. In the near future, companies will begin to send small packages with fast food, bills, documents, books, or DVDs using autonomous robots. One advantage is that the flying vehicles can take the sky, unlike human messengers. In 2013, Amazon announced that it is willing to test this idea, creating a new delivery system to get packages into customers' hands using drones [11]. The Matternet project [12] proposes quadcopters forming a network to distribute food and medicine. There are already robotic messengers carrying medicine and documents between departments in hospitals [13] (see Figure 2.5).

- **Cleaning**. Cleanliness and order are important in any advanced civilization. But

cleaning and maintaining order have always been heavy and monotonous activities. Robots can perform these tasks for us at home and in public places. Take a look at the current progress: iRoomba cleans the floors of homes [14], Lely Discovery keeps the environment of cows hygienic at farms [15], and Ambrogio maintains lawn at the desired height in your garden [16]. This is just the beginning.

- **Agriculture**. The machinery added in the 20th century has allowed modern agriculture to increase its productivity and to free humans to do other activities that develop civilization. The company John Deere estimates that 90% of the U.S. population in 1848 was involved in agriculture, and today it is about 0.9%, partly because of the machinery for harvesting and planting. Why not automate it? The algorithms developed for autonomous navigation may allow the machinery to produce the same with minimal human intervention and supervision for repairs. See these references: in Australia [17][18] and Denmark [19], people are giving autonomy to agricultural machinery. In some years, we will see companies exploiting these opportunities; I wish they are Mexican.

- **Surveillance**. Algorithms to perceive and model the environment might be used to monitor human behavior. Imagine quadcopters looking for criminals on the streets and reporting it to the nearest police officer. Imagine watchdog robots at home that detect the entry of an unknown person or when a window breaks. The Japanese company Secom has developed two autonomous robots for surveillance and promises to put them on the market [20].

- **Search and Rescue**. In Mexico, one of the basic tasks of Plan DN-III-E is to search and rescue in disasters. The Mexican army could use mobile robots to find people trapped in dangerous places. With the information you collect, the robot can serve to create a better bailout. Currently, most robots are teleoperated [21]. But giving them the ability to navigate and search, a group of robots can cover the same area faster, increasing the likelihood of rescuing people. Search and rescue have several challenges: mobility on land with debris, enough power for long missions, identifying victims,

Figure 2.5: **Package delivery**: quadcopter and station to provide food and medicines to rural areas; RobotCourier to transport substances and documents in hospitals. **Cleaning**: iRoomba autonomous vacuum cleaner; Ambrogio, autonomous mower; Lely Discovery, cleaner for farms. **Agriculture**: autonomous agricultural machinery. **Surveillance**: Secom company's robots.

and others. The RoboCupRescue competition aims to overcome these limitations (see Figure 2.6). A team who took part in this competition, the Hector Team Darmstadt [22], has developed robots with search and rescue skills, including map building in not planar floors, detecting victims, planning motion, semantic mapping, etc.

- **Building**. The construction industry can benefit from robotics as the manufacturing industry has done. Robots would liberate humans from heavy labor, improving their life quality. A city could build people's homes faster and cheaper. Although construction robotics is emerging, there is some progress: space exploration robots carry metal bars to form walls [23], quadcopters build cubic structures [24], and mobile robots with mechanical arms build furniture [25].

- **Transportation**. Most goods you use were brought from distant places; this is possible by planes, cars, and boats which amplify human force. In the future, we will see those vehicles moving with minimal human assistance, increasing productivity, efficiency, and safety (reducing the number of accidents). Cities will have autonomous public transport (e.g., there was a demonstration of an autonomous minibus in the Intelligent Vehicles Symposium 2012). Also, short-distance transportation can benefit: Kiva company sells a robotic system that handles the inputs and outputs of a warehouse [26].

- **Guiding People**. Since 1997, there have been robots that interact with visitors in museums [27][28], explain the exhibits, and guide people. This technology can also be used in guiding blind people or for advertising in malls, football stadiums, and concerts. Imagine a sales robot that is appealing to the public and engages people to offer them a product. Imagine flying robots that align to form shapes in the air while they advertise a service.

Figure 2.6: **Search and Rescue**: Japanese robot to retrieve people; maritime quadcopter to monitor seas; robot to find people in collapsed buildings. **Building**: rovers stacking rods; quadcopters building cubic structures; mobile robots building a small table. **Transportation**: a minibus navigates autonomously; the Kika robot system that handles the inputs and outputs of a warehouse. **Guiding People**: Rhino, one of the first robots that guides visitors in a museum; a more current guide robot; robotic dog for blind people.

2.1.3 Main problems

Each of the above applications has specific challenges, but they share autonomous navigation as a problem. Even though navigation is easy for humans, it is a difficult task for robots. They must navigate in unstructured environments, reaching the given target without damage. To achieve this goal, the robot should solve different subproblems. Figure 2.7 shows a system of autonomous navigation.

The body of the robot has the following:

- Sensors send pieces of information about the environment and the robot's state (e.g., distance to the nearest object or robot speed).
- Actuators execute the motions that the computer commands (e.g., electric motors).

The robot's mind is a set of algorithms executed by the computer:

- **Perception**: Interprets the numbers sent by the sensors to recognize objects, places, and events that occur in the environment or in the robot. In this way, the robot can prevent damage, know where it is, or know how the environment is.
- **Map building**: Creates a numerical model of the environment around the robot. This allows the robot to make appropriate decisions and avoid damage.
- **Location**: Estimates the robot's position with respect to the map. This helps it plan and execute movements, and build a correct map of the environment.
- **Planning**: Decides the movements necessary to reach the goal without colliding and in minimum time or distance.
- **Control**: Ensures the planned movements are executed, despite unexpected disturbances.
- **Obstacle avoidance**: Avoids crashing into moving objects, such as people, animals, doors, furniture, or other robots that are not on the map.

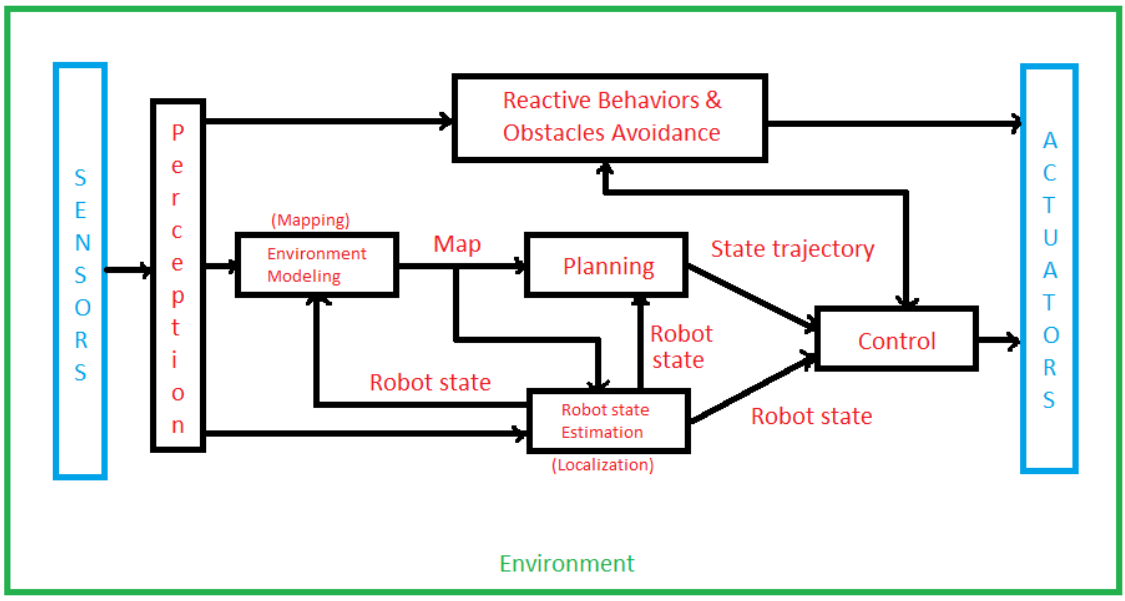


Figure 2.7: This illustrates the subsystems needed to solve the whole autonomous navigation problem.

Algorithms already exist for each of these tasks [29].

The problem of navigation in unstructured environments presents several challenges. However, this thesis only focuses on two specific problems.

1. Map building (SLAM, *Simultaneous Localization and Mapping*). The effectiveness and efficiency of navigation depends on having a map with no errors. Most map-building methods are based on the Gaussian assumption (i.e., estimation errors modeled by a Gaussian probability distribution). But this assumption is incorrect because the real error distributions are not Gaussian. So we propose and study a new method that avoids the use of the Gaussian distribution, and errors are modeled by ellipsoidal sets. Besides, none of the other methods based on sets has shown to be able to build large-scale maps. Our method can.

2. Planning in dynamic and unknown environments. In these types of environments, the problem of partial observability is important because planning has to make certain assumptions about the environment that cannot be or has not been observed. And these assumptions may be wrong. Even if we have a perfect mapping method, the map might have errors relative to the real environment. This type of problem is important to solve because it affects the optimality of the path length and the completeness of the navigation task (i.e., to be able to reach the goal when a solution exists). We present theorems of sufficient conditions to guarantee that a navigation system can complete its task. We propose the first methods to recover from false obstacles on the map in dynamic environments. And we propose a method to navigate more efficiently in unknown environments.

In the next two sections, we present the state of the art of these two problems: map building and planning. And we give more details on these issues.

2.2 SLAM

"It is only by learning from mistakes that progress is made." —Sir James Dyson

2.2.1 SLAM problem

The objective of simultaneous localization and mapping (SLAM) is to build a map and to locate the robot in that map at the same time. We should clarify that for the SLAM problem, it does not matter if the robot moves autonomously or is controlled by a human. The important thing is to build the map and locate the robot correctly.

The most basic way to locate a robot is using odometry. For a terrestrial mobile robot, odometry consists of integrating the displacements measured by encoders to estimate the position and orientation of the robot. The problem is that odometer error accumulates as the robot moves. Eventually, the error is so large that odometry no longer gives a good estimate of the state of the robot, as shown in Figure 2.8. Then the robot must make observations of some references in the environment to correct the odometer error, assuming the references are static. When the robot returns to observe, these references can reduce the accumulated error. The main work of the SLAM method is to correct the estimation of the robot state and the map. On the other hand, the SLAM is a chicken-and-egg problem because the robot needs a map to locate itself and the map needs the localization of the robot to build a consistent map. Thus SLAM methods must be recursive. This is why the SLAM problem is so difficult because errors in robot and map states affect each other, producing inconsistent maps. In the following section, we will give an overview of several SLAM methods.

2.2.2 SLAM solutions

In the last decade, many types of SLAM have been developed. They can be categorized by different criteria, such as state estimation techniques, type of map, real-time performance, the sensors, etc. However, we classify the techniques according to the type of problem they solve into the following four categories: Feature-based SLAM, Pose-based SLAM, Appearance-based SLAM, and Other SLAMs. In Figure 2.9, we compare these main SLAM paradigms.

Feature-based SLAM. Odometer error can be corrected by the use of landmarks in the environment as references. Consider a mobile robot that moves through an environment,

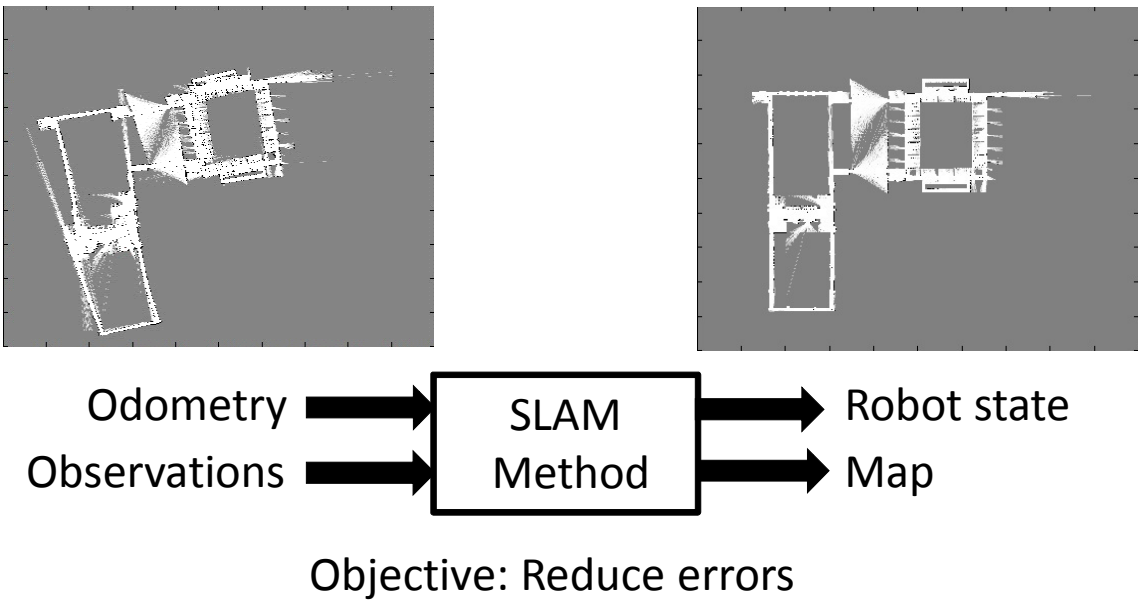


Figure 2.8: The goal of the SLAM method is to build a consistent map and locate the robot during its journey. In general, SLAM methods integrate the observations of some references in the environment with the odometry to reduce errors.

	Feature SLAM	Pose SLAM	Appearance SLAM
Loop-closure performance	++	++	+++
Wake-up robot problem	+	+	+++
Kidnapped robot problem	+	+	++
Metric localization	Yes	Yes	No
Type of map	Landmark or grid map	Landmark or grid map	Topological map

Figure 2.9: We compare the main SLAM paradigms in terms of their capacity to solve some problems and the type of map they build. Loop-closure performance measures the ability to close the loop when the robot revisits the same place. The wake-up robot problem, also called global localization, refers to a situation where a robot is carried to an arbitrary location and put to operation; the robot must localize itself without any prior knowledge. The kidnapped robot problem refers to a situation where a robot in operation is carried to an arbitrary location; the robot must localize itself, avoiding confusions.

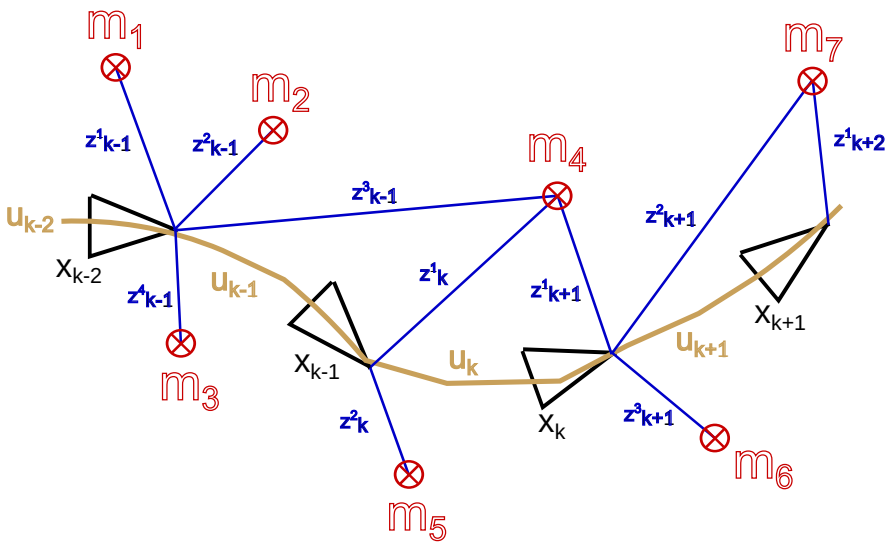


Figure 2.10: The robot simultaneously builds a map and localizes itself. At each time step k, the robot moves according to controls $\mathbf{u}_{k-1}$ and takes some observations $\mathbf{z}_k = \{\mathbf{z}_k^i, \mathbf{z}_k^j, ...\}$ at the pose $\mathbf{x}_k$.

as represented in Figure 2.10. In the beginning, we can take the initial robot position as zero. The robot must be able to detect some specific objects in the environments, which are called landmarks. This robot must also be able to measure the distance between its position $\mathbf{x}_k$ and landmark positions $\mathbf{m}_i$ in the environment and also measure the angle between the robot-landmark line and some reference line. At each time step k, the robot moves according to controls $\mathbf{u}_{k-1}$ and takes measurements $\mathbf{z}_k = \{\mathbf{z}_k^i, \mathbf{z}_k^j, ...\}$ at the state $\mathbf{x}_k$.

There are three tasks shown in Figure 2.11 that any feature-based SLAM method must do. (1) Landmark detection. The robot must recognize some specific objects in the environment; they are called landmarks. It is common to use a laser range finder or cameras to recognize landmarks, such as corners, lines, trees, etc. (2) Data association. Detected landmarks should be associated with the landmarks on the map. Because landmarks are not distinguishable, the association may be wrong, causing large errors on the map. Besides, the number of possible associations can grow exponentially over time; therefore, data association

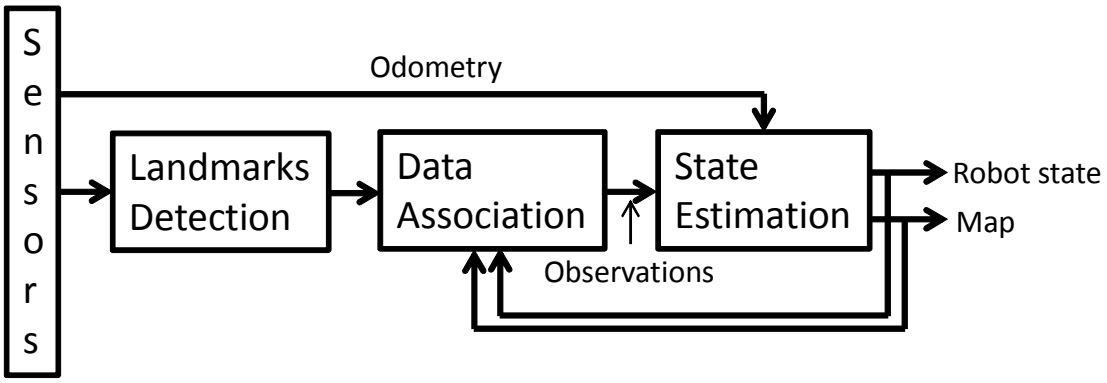


Figure 2.11: Any method of feature-based SLAM must do these tasks: detecting landmarks on the environment, associating the detected landmarks with the landmarks on the map, and improving the estimate of the robot's position and the map.

is a difficult task. (3) State estimation. It takes observations and odometry to reduce errors. The convergence, accuracy, and consistency of the state estimation are the most important properties. In this thesis, we focus on state estimation task.

The major difficulties of any SLAM method are the following:

- High dimensionality. Since the map dimension always grows when the robot explores the environment, the memory requirements and time processing of the state estimation increase. Some submapping techniques can be used to solve it (metric-metric approaches [52][53][54], topological-metric approaches [55][56][57]).

- Loop closure. When the robot revisits a past place, the accumulated odometry error might be large. Then the data association and landmark detection must be effective to correct the odometry. Place recognition techniques are used to cope with the loop closure problem [71][78].

- Dynamics in environment. State estimation and data association can be confused by the inconsistent measurements in the dynamic environment. There are some methods that try to deal with these environments [36][85][86][87].

Pose-based SLAM. It estimates only the robot's state. It is easier than feature-based SLAM because the landmark positions are not estimated. However, it must maintain the robot path, and it uses the landmarks to extract metric constraints to compensate the odometer error. Therefore, the high dimensionality limitation arises from the dimension growing by robot states rather than by the landmark states. Landmark detection, data association, loop closure, and dynamic environment still are relevant problems. Most pose-based SLAMs employ a laser range finder and the laser scans that form the occupancy grid maps of the environment (if the odometry is corrected, the grid map is right). Instead of using the landmark detector, the laser scan transforms the data association problem as an matching of laser scans for extracting the constraints. This problem is also called front-end problem and is typically hard due to potential ambiguities or symmetries in the environment. Path estimation can be performed by optimization techniques [33][62][63][64][65][66] [67][68], information filters [34][69], and particle filters [70][35].

Appearance-based SLAM. It does not use metric information. The robot path is not tracked in a metric sense; instead, it estimates a topological map of places. Topological maps are useful for global motion planning, but not for obstacles avoidance. The configurations of the landmarks of a given place can be used to recognize the place; visual images or spatial information are also utilized to recognize the place. The metric estimation problem is avoided, but data association is still an important problem to solve. Loop closure is performed in the topological space. High dimensionality is translated into the number of places in the topological map. It is very common that these appearance techniques are used complementary to any metric SLAM method to detect loop closures [36]**.** Some methods based on visual appearance are [71][72][73][74][75][76][77][78], and others based on spatial appearance are [79][80].

Other SLAMs. There are several variants of the SLAM problem that cannot be included in the above paradigms. Pedestrian SLAM employs light and cheap sensors and faces the obstacle of human movements, which are different to robot behavior [37][81][82][83]. SLAM with enriched maps uses auxiliary information on the map [38][84], such as temperature, terrain characteristics, and information form humans. Active SLAM derives a control

law for robot navigation in order to efficiently achieve a certain desired accuracy of the robot location and the map [39]. Multirobot SLAM uses many robots for large environments [40]. SLAM in dynamic environments [85][86][87] deals with moving objects and agents.

In this thesis, we are interested in the feature-based SLAM paradigm because it is the original form of the SLAM problem. The SLAM method we are proposing in this thesis belongs to this category. So we go into more detail about the methods to solve feature-based SLAM problems.

There are two versions for feature-based SLAM [41][42]: (1) Full SLAM. It estimates the current and the past states $\{x_1, ..., x_{k-1}, x_k\}$ in order to know the complete robot trajectory and to estimate the landmark's positions $m = \{m_1, m_2, ...\}$ at each time step (this is a smoothing problem). (2) Online SLAM. It estimates only the current robot state x_k and the landmark's positions m at each time step (this is a filtering problem). Our method (Ellipsoidal SLAM) is an Online SLAM.

An important point is that there are two ways of error modeling: by probability distributions or by sets that bound the error. Our method (Ellipsoidal SLAM) uses ellipsoidal sets to model the error. In the following sections, we present the main approaches for these categories and discuss some of their limitations.

Probabilistic approaches

In these methods, the odometer error, the sensors' error, and the modeling error (for example, linearization) are modeled by probability distributions. We will present an overview of the main probabilistic methods used in the SLAM problem. In Figure 2.12, we show a general comparison among them.

Kalman-based SLAM. Extended Kalman Filter SLAM (EKF-SLAM) is the first solution for the Online SLAM problem, and it is the most popular method. The complete discussion can be found in [30]. Since the Gaussian noise assumption is not realistic and causes fake landmarks on the map, EKF-SLAM requires additional techniques to manage the map to eliminate the fake landmarks. The consistency and convergence of the EKF-SLAM algorithm are studied in [43], and various techniques are reviewed to overcome the incon-

	Type of probability distribution	How to deal with nonlinearity	Type of SLAM problem	Gaussianity
Kalman	unimodal	linearization	online SLAM	Yes
Information	unimodal	linearization	online & full SLAM	Yes
Particles	multimodal	stochastic sampling	online & full SLAM	Yes, for landmarks
Graph (optimization)	none	with or without linearization	full SLAM	Yes

Figure 2.12: We compare the probabilistic methods used for feature-based SLAM problem. Note that all are assuming Gaussianity. Full SLAM estimates the complete trajectory of robot and the landmark's positions at each time step (this is a smoothing problem). Online SLAM estimates only the current robot pose and the landmark's positions m at each time step (this is a filtering problem).

sistency problem of EKF-SLAM in [44]. The biggest disadvantage of EKF-SLAM is that its computing time is quadratic over the number of landmarks due to the update of the covariance matrix. There are several ways to overcome this limitation: (1) limiting the number of landmarks to be estimated [31], (2) updating only the part of currently detected landmarks and postponing the complete update to a later time [45], and (3) making approximations in the covariance update stage [46].

Information-based SLAM. Motivated by reducing the computing time of EKF-SLAM in larger environments, the Extended Information Filter (EIF) was developed. The great advantage of EIF over EKF is that the computing time can be reduced due to the sparseness property of the information matrix. However, the recovery of the landmark positions and the covariance associated with the landmarks in the vicinity of the robot are needed for data association, map update, and robot localization. When the number of landmarks is small, it can be obtained by the inverse of the information matrix, but the computational cost of the inversion will be unacceptable with a large information matrix. Some methods that use the information filter are: Sparse Extended Information Filter [47], Exactly Sparse Extended Information Filter [48], and Decoupled SLAM [49].

Particles-based SLAM. Particle filter is a sequential Monte Carlo inference method that approximates the exact probability distribution through a set of state samples. Its advantage is that it can represent any multi-modal probability distribution. It does not need Gaussian assumption. Its main drawback is that the sampling in a high-dimensional space is computational inefficient. But if the problem estimation has "nice structure," we can reduce the size of the space using the Rao-Blackwellized Particle Filter (RBPF), which marginalizes out some of the random variables. The RBPF is used in conjunction with the Kalman filters to estimate the positions of the landmarks (one Kalman filter for each landmark). Some methods that use particle filter are: the FastSLAM algorithm [58] and its improved versions (the UFastSLAM algorithm [59] and the Gaussian mixture probability hypothesis density (PHD) filter [60]).

Graph-based SLAM. These methods use optimization techniques to transform the SLAM problem into a quadratic programming problem. The historical development of this

paradigm has been focused on pose-only approaches and using the landmark positions to obtain constraints for the robot path. The objective function to optimize is obtained assuming Gaussianity. Some methods are: GraphSLAM [32], Square Root SLAM [50], and Sliding Window Filter [61]. Their main disadvantage is the high computational time they take to solve the problem. So they are suitable to build maps off-line. They are used because they are more accurate.

Set-membership approaches

Another way to model the error is by sets. The literature reports a few SLAM solutions using sets.

Di Marco et al. [124] have developed an online algorithm. They obtain linear time complexity $O(N)$ with respect to the number of landmarks N by state decomposition and approximating the sets as aligned boxes. The correlation between robot states and landmark positions is lost due to the decomposition state. These correlations are crucial to build a consistent map, especially when the robot closes a loop [118]; this is why we think this method is not able to solve large-scale SLAM problems. Our method (Ellipsoidal SLAM) saves these correlations as Kalman filter does.

CuikSLAM [88] is a project based on interval analysis. They interpret the SLAM problem as a set of kinematic equations that are solved by interval arithmetic. Unfortunately, this work assumed that the association problem is solved and only present a simulation of a simple example to validate the algorithm.

Jaulin [122][123] represents the uncertainty with intervals of real numbers. It translates the range-only and full SLAM problems in terms of a constraint satisfaction problem. It uses interval analysis and contraction techniques to find the minimal envelope of robot trajectory and the minimal sets to enclose the landmarks. This method has some limitations. It lacks the association among landmarks and observations. In [122], it requires a human operator to make the association. The algorithms must work off-line due to the large computational cost of contraction. Besides, their operation is sensitive to outliers, yielding empty sets and stopping the estimation. These shortcomings mean it can not solve large-scale SLAM

problems. In contrast, our method can deal with the association problem without a human operator, and it has the potential to work online. And more importantly, we show that it is able to build large-scale maps.

2.2.3 Conclusion

In this section, we gave an overview of the SLAM problem and its solutions. We showed that the simplest way to model the error is with a Gaussian distribution. Most methods consider Gaussianity because we only require two parameters: mean and variance. However, we know that Gaussianity is not a realistic assumption, as we will show in the next chapter. In this thesis, we are looking for a method that does not assume this. And we note that none of the SLAM methods based on sets can solve large-scale problems (with high dimensionality in the state vector). This thesis proposes a method based on ellipsoidal sets (**Chapter 3**). The ellipsoid filter is similar to Kalman filter, and then we can extrapolate many techniques of EKF SLAM to deal with large environments, data association, and map management.

2.3 Motion planning

> *"The definition of insanity is doing the same thing over and over again and expecting different results." —Albert Einstein*

2.3.1 Planning problem

Motion planning decides what and how to move. It has the purpose of generating a trajectory (path) in state space to reach the goal, given an environment model (map), the current state of the robot, and a goal state (see Figure 2.13). It must avoid the obstacles in the environment and must take into account the kinematic and dynamic constraints of the robot and its size. Motion planning is a difficult task because the algorithms should search for a solution on a continuous and high-dimensional state space. They must discretize the state space to make decisions as soon as possible.

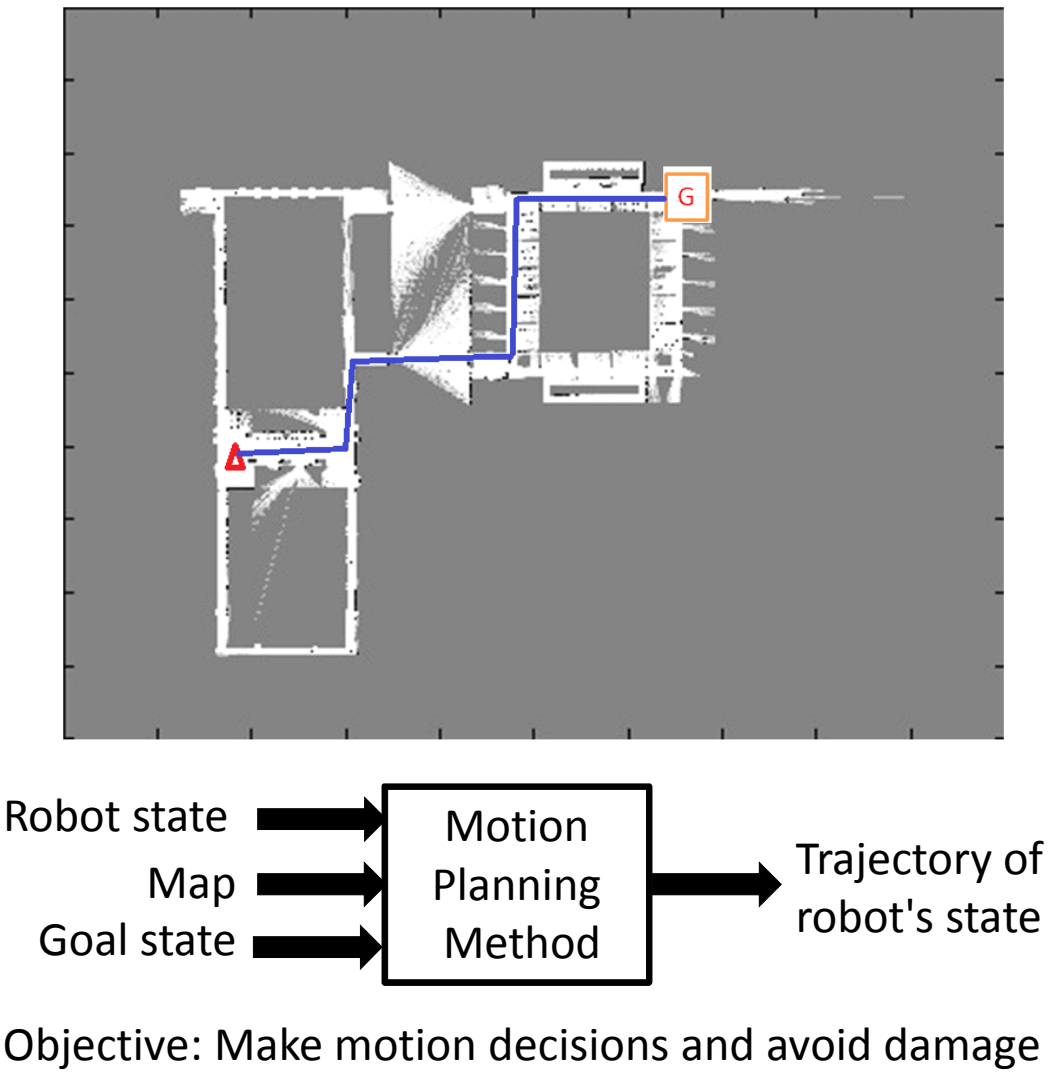


Figure 2.13: The motion planning method must generate a state trajectory (blue line), avoiding damage to the robot (red triangle) and reaching the goal state (G).

The motion planning problem can be stated as follows: given an environment model (an occupancy grid map, in our case), an initial state $\mathbf{x}_I$, and a goal state $\mathbf{x}_G$, the planning algorithm must calculate automatically a sequence of states $\mathbf{x}_1 = \mathbf{x}_I, \mathbf{x}_2, ..., \mathbf{x}_N = \mathbf{x}_G$, which conforms to a free-collision trajectory, with minimal length and enough smooth, and satisfies the kinematic and dynamic constraints of the robot.

2.3.2 Planning approaches

In general, there are two kinds of planners: *Graph Search*, where the trajectory is searching in a graph of nodes that discretely represent the state space of the robot; and *Controllers*, where instead of obtaining a path, they determine the control policy for every location in the state space. We briefly review the most important algorithms for each category. A detailed explanation about the following algorithms can be found in [99][29].

Graph Search algorithms discretize the state space by creating a graph [141][142][143] with nodes that represent robot states and edges that represent the robot actions. The graph formation can be too simple, like dividing the physical space into a grid [144], or more complicated, like randomly sampling the state space [145][146]. The most common Graph Search algorithms are the following:

- Breadth-First Search [100]. The nodes are explored in order of proximity. Proximity is defined as the shortest number of edge transitions. The generated path could be the minimum-cost path (i.e., it is the optimal path) if the cost of edges is a non-decreasing function of the depth of the node. Its large disadvantage is that the memory requirement is bigger as the depth of the graph increases.

- Depth-First Search [100]. In contrast to the previous algorithm, this explores each node up to the deepest level of the graph. The advantage is that its memory requirement is less than Breadth-First Search because it stores only a single path during execution. However, this search does not guarantee finding the optimal solution.

- A* search [101][102]. It is an improvement of Dijkstra's algorithm [103], adding a

heuristic function that encodes knowledge about the cost for reaching the goal. The optimal solution is guaranteed, if the heuristic is always an underestimated the cost. This heuristic reduces the number of node explorations with respect to Breadth- and Depth-First. Its main drawback is the high memory requirement because it keeps all generated nodes in memory. Some ways to face this difficulty is applying an Iterative Deepening strategy (IDA*) [100], Recursive Best-First Search (RBFS) [104], or Memory Bounded (MA* or simplified MA*) [105].

- D* algorithm [106](or D*Lite [107]). They are incremental replanning versions of A*, where the algorithm reuses the previous search efforts, instead of generating a new solution from scratch. Search time may decrease by one or two orders of magnitude. Replanning is important in dynamic environments or when the map is corrupted by noise or when the environment is unknown.

- Anytime planners A*[108] or D*[109]. The anytime planners are suitable when the planning time is limited. They find a feasible solution quickly (a suboptimal solution) and then continue to work on improving it until time runs out.

- Distance-propagating dynamic system [110][111]. Each node stores a current estimate of the distance to the nearest target and the neighbor node from which this distance was determined. The update of the distance estimate at each node is done using only the information gathered from the node's neighbors. The path is determined from the information at the robot's current location. This algorithm can deal with dynamic obstacles or dynamic goals.

- Higher geometry maze routing algorithm [112]. This algorithm is inspired by Lee's routing algorithm [113] in the grid plane. It can deal with any shape of robot to avoid obstacles.

- Sampling-based planners. All previous search algorithms are deterministic. In contrast, sampling-based planners are stochastic to solve complex high-dimensional motion problems for fly vehicles, humanoids, or robot manipulators in cluttered environments or

narrow passages. Some examples are Probabilistic Roadmap Planner [145], Expansive-Spaces Tree Planner [114], and Rapidly Exploring Random Trees (RRTs) [115].

The most important **Controllers** are the following:

- Potential field planning [93]. It is not a formal controller, but its output is a control policy of robot direction. It creates a gradient (a virtual force) across the map that directs the robot to the goal. This gradient is calculated by a function that continuously represents the occupied and free space of the map. The goal position is represented by the minimum of the function (attractive force), and obstacles are related to high values of the function (repulsive forces). Its main drawback is that the generated function can have local minimums depending on the shape of the obstacles (e.g., if the obstacles are concave, the robot could oscillate). To overcome this limitation, we can use the harmonic potential field method [94], which numerically solves a Laplace equation.

- Dynamic programming [96]. It is a numerical method to solve optimal control problems based on the principle of optimality developed by R. E. Bellman [95]. This method is used to generate control policy over the free space of the map. The main disadvantage is its high computational cost.

- Value Iteration [97]. It is a Markov decision process to determine a control policy, considering the uncertainty of robot actions. Value iteration recursively calculates the utility of each action relative to a payoff function. Its drawback is also its high computational cost.

- Model Predictive Control (MPC). It uses a reference model to predict the dynamic behavior of the robot in next time step, and using an optimization technique, it determines the best action controls. Tahirovic et al. [98] have studied the use of MPC with passivity-based control and a potential field representation of the environment to propose a navigation planner.

Planning is performed by A* search method in this thesis because we consider that the mobile robot only moves in a plane, to simplify the problem.

The effect of partial observability of environment on planning

In this thesis, we are interested in studying the effect of partial observability of the environment on planning. Planning requires making some assumptions about the environment that cannot be or has not been observed to generate paths in these regions in order to reach its goal. The problem is that these assumptions about the environment may be wrong, and can cause the planning to make bad decisions. This problem is more important when environments are dynamic or unknown because there is no way to guarantee that the map and the environment are the same. Most navigation systems reported in literature [147][78][148][142] assumes to know the environment without errors or changes. Or when the environment is previously unknown, it assumes the environment is free of obstacles [148][142][149]. There have been few studies on these assumptions even though they are fundamental for the navigation problem.

The assumption of free obstacles is studied in [153][150]. The aim of the authors is to analyze how suboptimal the path length is when the environment is initially unknown, focusing on the worst path length obtained by three planning methods Greedy Mappingg (GM), Dynamic A* (D*), and Chronological Backtracking (CBack). For the first two methods, the authors derived upper $O(|V|^{3/2})$ and lower $\Omega(\frac{\log|V|}{\log\log|V|}\,|V|)$ bounds on the worst-case path length as a function of the number V of vertices of the graph. And the same authors showed that GM and D* can present shorter path lengths than CBack for an average-case analysis. They assume that the known area of the map is identical to the environment, which is not true, especially when the environment is dynamic. In this thesis, we study what could happen in this case, showing that it is necessary to develop algorithms to recover from map errors. This problem has been ignored systematically in literature, with the thinking that the map is always equal to the environment.

2.3.3 Conclusion

In this section, we gave an overview of the planning problem and its solutions. Besides, we showed the few papers that have addressed the problem of partial observability of the

environment and its effects on planning. Inspired by this situation, this thesis studies how this partial observability affects navigation (**Chapter 4**), given formal definitions for the autonomous navigation problem. We present theorems of sufficient conditions to ensure a navigation system is complete. We propose the first methods to recover from false obstacles on the map in dynamic environments. And we propose a method to navigate more efficiently in unknown environments.

Chapter 3

Ellipsoidal SLAM

"I have not failed. I've just found 10,000 ways that won't work." —Edison

3.1 Introduction

Robotics science aims to build autonomous robots. One part of the problem consists of giving them autonomy for navigation. This implies, among other things, robots must build an environment model (i.e., a map). This problem is known as Simultaneous Localization and Mapping (SLAM) [118]: the robot must estimate the position of some landmarks in the environment and use them to estimate its state, and vice versa, use its state to estimate the landmark positions. Thus SLAM solutions are recursive.

In the last 15 years, many types of SLAM methods have been developed. They can be categorized as feature-based SLAM [47], pose-based SLAM [33], and appearance-based SLAM [36]. For a more complete revision of categories and techniques, we invite the reader to consult our review [119]. In this chapter, we focus on the feature-based SLAM problem.

There are some uncertainties in determining the robot's position and the landmarks' positions due to sensor and model errors. To compensate these errors, some techniques are used to estimate the states. Robotics community has focused on the probabilistic methods. The extended Kalman filter (EKF) is the most popular method [118]. It represents the robot

and environment states with Gaussian probability distributions. There have been other estimation techniques applied for SLAM, such as extended information filter (EIF) [47][49], particle filters [58][60], and optimization smoothers [51][61]. **The fundamental critique of probabilistic methods is that the real probability distributions of the state are usually unknown**. To facing this problem, it is commonly assumed that probability distributions are Gaussian. This is assumed by the Kalman, and the information filters, and the optimization smoothers. Even the particle filter [58] uses Gaussian assumption in sample motion models and landmark positions. As we will show, the assumption is wrong because **sensor and model errors are no Gaussian in real world for SLAM problem**.

In contrast, we can leave out the need to know probability distributions, if **we represent sensor and model errors by sets**. Set membership techniques are a helpful tool to solve the state estimation of dynamic systems with bounded disturbances [120]. There are two common methods: polytopes, and ellipsoids.

The *polytope-set method* has been applied for localization problem [121] and for SLAM problem [122][123][124]. In [122] and [123], the interval analysis, the contraction technique and the polytope sets are used, finding the minimal envelope of robot trajectory. The algorithms can only work off-line due to computational cost of the contraction. In [124], an online algorithm with polytope sets is developed. However, the correlations between robot and landmark positions are lost due to the decomposition simplification. These correlations are crucial to build a consistent map, especially when the robot closes a loop [118]. This algorithm is not suitable for large-scale SLAM problems.

In 1970s, the *ellipsoid method* was applied for state estimation with bounded noise [125][126][127]. The results caused great excitement, and stimulated many technical papers. [128] obtains confidence ellipsoids which are valid for a finite number of data points. [129] presents an ellipsoid propagation such that the new ellipsoid satisfies an affine relation with another ellipsoid. In [130], the ellipsoid algorithm is used as an optimization technique that takes into account the constraints on cluster coefficients. [131] describes several methods that can be used to derive an appropriate uncertainty ellipsoid for array response. [132] concerns asymptotic behavior of the ellipsoid estimation. Even though all of these

developments, the ellipsoidal method for SLAM is still not published from the best of our knowledge.

The main advantage of ellipsoid-set over polytope-set is the ellipsoid-set needs fewer information because sets can be described by matrices, like covariance matrix in the Kalman filter, but with no Gaussianity assumption. So we will show that **the ellipsoid filter can solve large-scale SLAM problems without Gaussian assumption;** this is our main contribution in this chapter. The main obstacle of this application is the ellipsoid filter is more complex than the Kalman filter; and it is difficult to analyze its convergence property.

In this chapter, we use ellipsoid algorithm for solving SLAM problem. The proposed Ellipsoidal SLAM has advantages in certain noise conditions, online realization, and large-scale problems. Furthermore, its performance is similar to EKF SLAM with no Gaussian assumption, which means we have more realistic error modeling. By bounded ellipsoid technique, we analyze the convergence and stability of the novel algorithm. Simulation and experimental results show that the Ellipsoidal SLAM is effective for on-line and large-scale problems, such as Victoria Park dataset. In this chapter, we modify the ellipsoid algorithm for SLAM in **Section 3.2**. We analyze convergence and stability of state estimation with the bounding ellipsoid algorithm by a Lyapunov-like technique [133] in **Section 3.3**. We validate the Ellipsoidal SLAM with a simulation test, Victoria Park dataset and Koala mobile robot in **Section 3.4**. Finally, we give our conclusions in **Section 3.5**.

3.2 Ellipsoidal SLAM

The main motivation to use the Ellipsoidal SLAM is that the uncertainty in SLAM problem does not satisfy Gaussian distribution. The odometry error of the mobile robot Koala [138] are shown in Figure 3.1. The position and angle comes from the encoders of the robot. Here (a) shows the marginal distributions of odometry error for (x, y, θ). There are great differences between the real distributions (blue lines) and Gaussian distributions (red lines); it is clear that Gaussianity is false. However, if we bound the error distribution by an ellipsoid (without Gaussianity) as you see Figure 3.1-(b), this assumption is more realistic because

real errors have finite variability.

3.2.1 Ellipsoid filter for SLAM

Consider a mobile robot moving through an environment at time k, the state vector of robot is defined as $\mathbf{x}_k^r = (x_k, y_k, \theta_k)$; here (x_k, y_k) describes location, and θ_k is the orientation. The control vector is $\mathbf{u}_k$. The vector $\mathbf{x}_k^m = \left(\mathbf{m}_k^1, \mathbf{m}_k^2, ..., \mathbf{m}_k^L\right)^T$ describes the location of the landmarks; $\mathbf{m}_k^i = (x_k^i, y_k^i)^T$ is the location of the ith landmark, whose true location is assumed static. The state $\mathbf{x}_k$ of a SLAM system is composed into two parts: the states of the robot $\mathbf{x}_k^r$ and the states of the landmarks $\mathbf{x}_k^m$.

The state equation of SLAM is

$$\mathbf{x}_{k+1} = \begin{pmatrix} \mathbf{x}_{k+1}^r \\ \mathbf{x}_{k+1}^m \end{pmatrix} = \begin{pmatrix} \mathbf{f}(\mathbf{x}_k^r, \mathbf{u}_k) + \mathbf{w}_k \\ \mathbf{x}_k^m \end{pmatrix} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k) + [\mathbf{w}_k, 0]^T \tag{3.1}$$

where $\mathbf{f}()$ is the vehicle kinematics, $\mathbf{w}_k$ is the motion disturbances, $\mathbf{u}_k$ is the control vector. Since landmark positions $\mathbf{x}_k^m$ is not affected by the motion disturbances, we use the vector $[\mathbf{w}_k, 0]^T$ to indicate this.

We define $\mathbf{z}_k$ as the observation between the robot and landmark locations. The observation model for the ith landmark is

$$\mathbf{z}_k^i = \mathbf{h}(\mathbf{x}_k^r, \mathbf{m}_k^i) + \mathbf{v}_k^i \tag{3.2}$$

where $\mathbf{h}()$ depends on the geometry of the observation; the vector $\mathbf{v}_k$ is the observation error; $\mathbf{w}_k$ and $\mathbf{v}_k^i$ are not assumed to be zero mean Gaussian noises.

We use the ellipsoid method to estimate the robot state $\mathbf{x}_k^r$ and the landmark $\mathbf{x}_k^m$. The following definitions and assumptions are used.

Definition 3.1 *A real $n-$dimensional ellipsoid set, centered on x^*, can be described as*

$$E\left(\mathbf{x}^*, \mathbf{P}\right) = \left\{\mathbf{x} \in \Re^n \mid \left(\mathbf{x} - \mathbf{x}^*\right)^T \mathbf{P}^{-1} \left(\mathbf{x} - \mathbf{x}^*\right) \leq 1\right\}$$

where $\mathbf{P} \in \Re^{n \times n}$ is a positive-definite symmetric matrix. The volume of $E\left(\mathbf{x}^*, \mathbf{P}\right)$ is defined as in [127] and [129],

$$Vol(E\left(\mathbf{x}^*, \mathbf{P}\right)) = \sqrt{\det\left(\mathbf{P}\right)} U \tag{3.3}$$

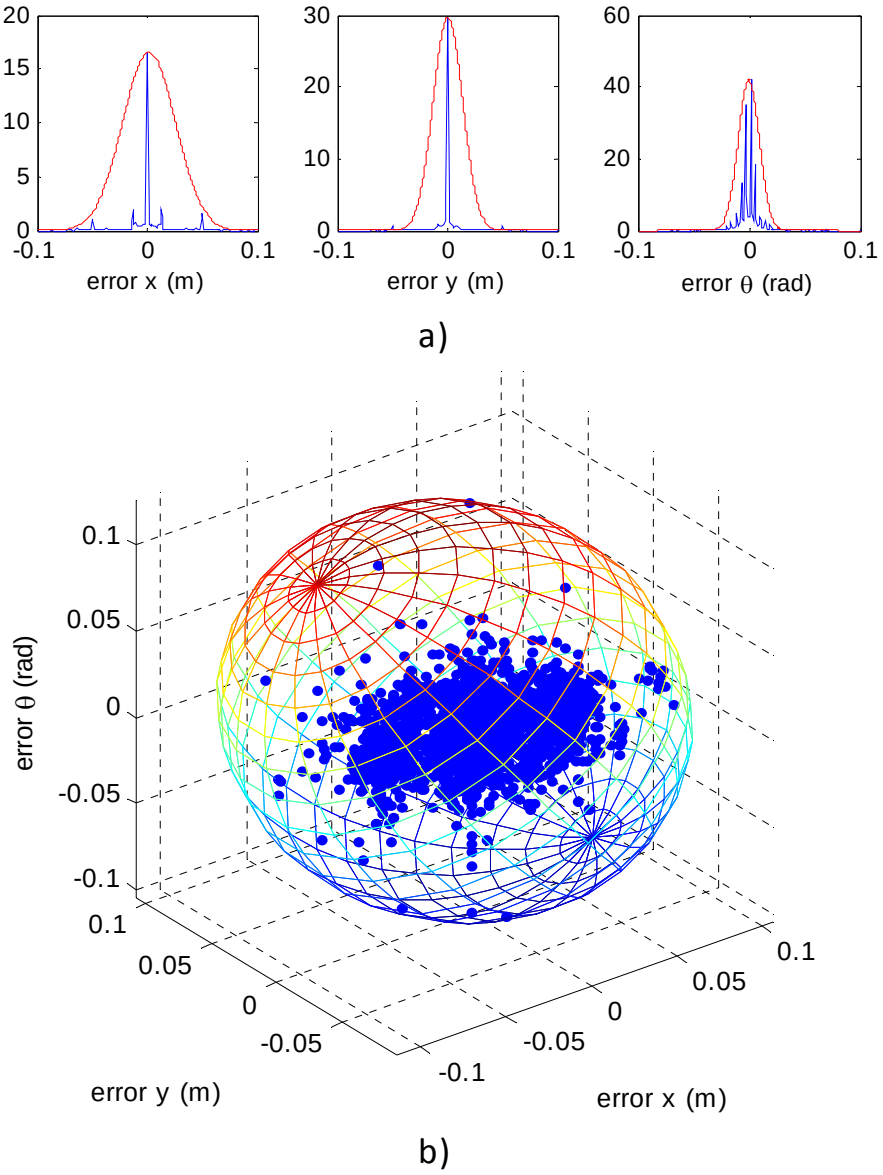


Figure 3.1: (a) Gaussianity is false for odometer error of a differential robot. Blue lines are marginal histograms of real odometer error. Red lines represent the maximum-likelihood Gaussian distributions adjusting to data. (b) The odometer error is bounded by an ellipsoid. A better assumption is to consider that the error is bounded and can follow any probabilistic distribution.

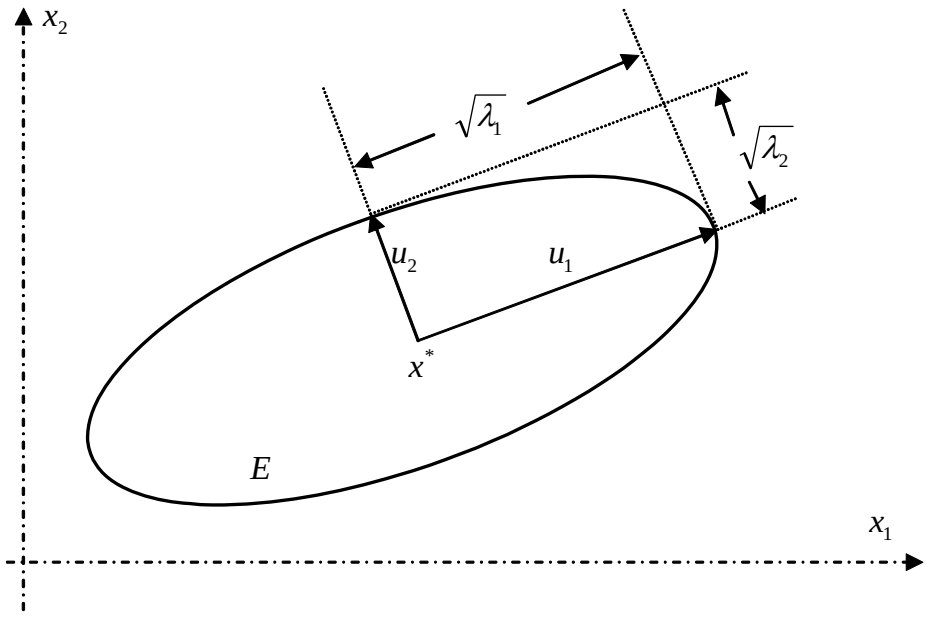


Figure 3.2: An ellipsoid showing its eigenvectors.

where U is the constant represents the volume of the unit ball in $\Re^n$.

The orientation (direction of axis) of the ellipsoid E is determined by the eigenvectors $[\mathbf{u}_1 \cdots \mathbf{u}_n]^T$ of $\mathbf{P}$, the lengths of the semimajor axes of E are determined by the eigenvalues $[\lambda_1 \cdots \lambda_n]^T$of $\mathbf{P}$. A two dimensional ellipsoid is shown in Figure 3.2.

Definition 3.2 *The ellipsoid intersection of two ellipsoid sets $E_a(\mathbf{x}_1, \mathbf{P}_1)$ and $E_b(\mathbf{x}_2, \mathbf{P}_2)$ is another ellipsoid set [126], defined as E_c,*

$$E_a \cap_e E_b = E_c(\lambda) = \{x \in R^n \mid \lambda(\mathbf{x} - \mathbf{x}_1)^T \mathbf{P}_1^{-1}(\mathbf{x} - \mathbf{x}_1) + ... \\ ... + (1-\lambda)(\mathbf{x} - \mathbf{x}_2)^T \mathbf{P}_2^{-1}(\mathbf{x} - \mathbf{x}_2) \leq 1\}$$

where $0 \leq \lambda \leq 1$, $\mathbf{P}_1$ and $\mathbf{P}_2$ are positive definite symmetric matrices, $\cap_e$ defines as the ellipsoid intersection.

In general, the intersection of the two ellipsoid sets $E_a \cap E_b$ is not an ellipsoid set. The definition guarantees the ellipsoid set E_c contains the intersection; Figure 3.3 shows this idea.

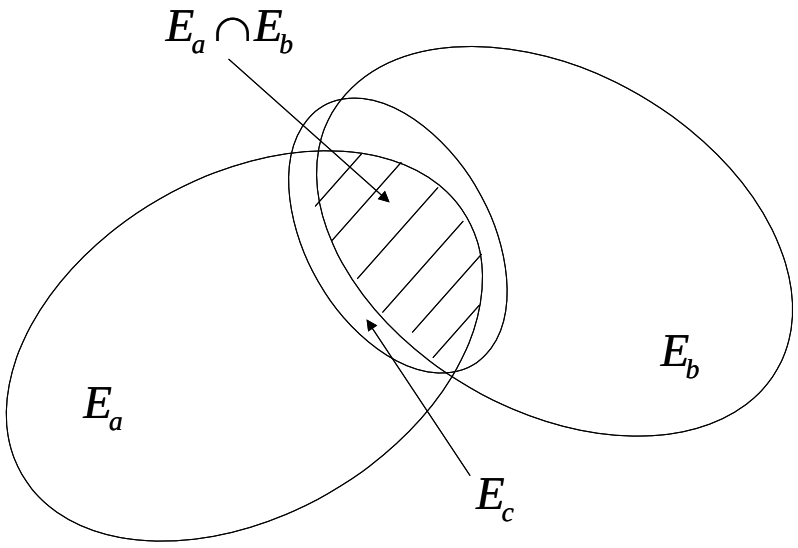


Figure 3.3: The intersection of two ellipsoid sets is not an ellipsoid set, but we can find an ellipsoid set that it can enclose it.

Many possible values of λ can be used, but there is a minimal volume ellipsoid corresponding to λ^*, called the optimal bounding ellipsoid (OBE) (see [129] and [126]). In this work, we will not try to find λ^*, but we will propose an algorithm such that the volume of the new ellipsoid intersection will not increase.

Linearizing (3.1) about the estimated state $\hat{\mathbf{x}}_k$ yields

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k) \mid_{\mathbf{x}_k=\hat{\mathbf{x}}_k} + \nabla\mathbf{F}_k \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_k) + O_1\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2\right] + [\mathbf{w}_k, 0]^T \tag{3.4}$$

where $\nabla\mathbf{F}_k = \frac{\partial \mathbf{F}}{\partial \mathbf{x}_k}|_{\mathbf{x}_k=\hat{\mathbf{x}}_k}$, $O_1\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2\right]$ is the linearization remainder. Note that the traditional EKF considers $\mathbf{w}_k$ as a zero mean uncorrelated Gaussian noise. The EKF method also ignores the linearization error term $O_1\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2\right]$.

Similarly, we linearize the observation equation

$$\mathbf{z}_k^i = \mathbf{h}(\mathbf{x}_k) \mid_{\mathbf{x}_k=\hat{\mathbf{x}}_k} + \nabla\mathbf{h}_k \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_k) + O_2\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2\right] + \mathbf{v}_k \tag{3.5}$$

where $\nabla \mathbf{h}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}|_{\mathbf{x}_k=\hat{\mathbf{x}}_k}$.

Using the ellipsoid definition for the state estimation, we define the state estimation error ellipsoid E_k as

$$E_k = \left\{ \mathbf{x}_k \mid \tilde{\mathbf{x}}_k^T \mathbf{P}_k^{-1} \tilde{\mathbf{x}}_k \leq 1 \right\} \tag{3.6}$$

where $\mathbf{P}_k = \mathbf{P}_k^T > 0, \tilde{\mathbf{x}}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k,$.

In this chapter, we use the following two assumptions.

A1 It is assumed that $[O_1 [(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2] + \mathbf{w}_k]$ and $[O_2(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2 + \mathbf{v}_k]$ belong to ellipsoid sets S_{1k} and S_k,

$$\begin{aligned} S_{1k} &= \left\{ \mathbf{x}_k \mid \left\| O_1 [(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2] + [\mathbf{w}_k, 0]^T \right\|^2 \leq \gamma_1 \right\} \\ S_k &= \left\{ \mathbf{x}_k \mid \left\| O_2 [(\mathbf{x}_k - \hat{\mathbf{x}}_k)^2] + \mathbf{v}_k \right\|^2 \leq \gamma \right\} \end{aligned} \tag{3.7}$$

where γ_1 and γ are positive constants.

A2 It is assumed that the initial estimation errors are inside an ellipsoid E_1

$$E_1 = \left\{ \mathbf{x}_1 \mid \tilde{\mathbf{x}}_1^T \mathbf{P}_1^{-1} \tilde{\mathbf{x}}_1 \leq 1 \right\} \tag{3.8}$$

where $\mathbf{P}_1 = \mathbf{P}_1^T > 0$, $\tilde{\mathbf{x}}_1 = \mathbf{x}_1 - \hat{\mathbf{x}}_1$.

In this chapter, we discuss open-loop estimation. We assume that all states in the models (3.4) and (3.5) are bounded. So Assumption **A1** is satisfied. Assumption **A2** requires the initial states to be bounded. Obviously, this assumption is established.

3.2.2 Recursive algorithm

Similar to other SLAM methods, the Ellipsoidal SLAM also has two-step recursive form. The following algorithms are based on our previous work in [133].

Prediction step. Given the past state $\hat{\mathbf{x}}_{k|k}$, the past matrix $\mathbf{P}_{k|k}$, and the past control $\mathbf{u}_k$, the states of the robot and the landmarks are calculated by

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k} &= \mathbf{F}(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k) \\ \mathbf{P}_{k+1|k} &= \nabla \mathbf{F}_k \frac{\mathbf{P}_{k|k}}{1-\beta_k} \nabla \mathbf{F}_k^T + \frac{\lambda_1 \gamma_1}{\beta_k} \mathbf{I} \end{aligned} \tag{3.9}$$

where λ_1 is a positive constant which is selected such that $\lambda_1 < 1$ and $\lambda_1 \gamma_1 < 1$, γ_1 is defined in (3.7). We use the free parameter β_k to adjust the size of ellipsoid $\mathbf{P}_{k+1|k}$. Here λ_1 and γ_1

are initial values. In the correction step, λ_i will be updated. γ_i are previous defined, they can be constants as $\gamma_i = \gamma$, see (3.7).

Correction step. Given predicted states $\hat{\mathbf{x}}_{k+1|k}$, predicted matrix $\mathbf{P}_{k+1|k}$, and current observations $\mathbf{z}_k^i$, the states of the robot and landmarks are calculated by

$$\begin{aligned}
&\lambda_{j,k} = \tfrac{\lambda_j \gamma_j}{1+\nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k} \nabla \mathbf{h}_{j,k}} \\
&(1-\lambda_{j,k})\,\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \tfrac{\lambda_{j,k}}{(1-\lambda_{j,k})\gamma_j + \lambda_{j,k} \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k} \nabla \mathbf{h}_{j,k}} \mathbf{P}_{k+1|k} \nabla \mathbf{h}_{j,k} \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k} \\
&e_{j,k} = z_{j,k}^i - h_j(\hat{\mathbf{x}}_{k+1|k}) \\
&\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \tfrac{\lambda_{j,k}}{\gamma_j} \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k}
\end{aligned} \tag{3.10}$$

where $0 < \lambda_{j,k} < 1$, $0 < \lambda_j < 1 - \lambda_{j,k} < 1$ and $\lambda_j \gamma_j < 1$. i represents ith landmark, j represents jth component of the observation $\mathbf{z}_k$, k is time. So at each time k, the landmark i has j observations. $\nabla \mathbf{h}_{j,k}$ is the gradient of function $\mathbf{h}()$ for component j.

3.2.3 Convergence property of the Ellipsoidal SLAM

The problem of state estimation (3.10) is to find a minimum set of E_k which satisfies (3.6) and (3.7). From Definition 2 we known the ellipsoid intersection E_k and S_k satisfies

$$(1-\lambda_{j,k})\,\tilde{\mathbf{x}}_{k+1|k}^T \mathbf{P}_{k+1|k}^{-1} \tilde{\mathbf{x}}_{k+1|k} + \frac{\lambda_{j,k}}{\gamma_j} \left([O_2\left(\tilde{\mathbf{x}}_{k+1|k}^2\right)]_j + v_{j,k}^2 \right)^2 \leq 1 \tag{3.11}$$

where $\tilde{\mathbf{x}}_{k+1|k} = \mathbf{x}_k - \hat{\mathbf{x}}_{k+1|k}$.

Next theorem shows that the recursive algorithms (3.9) and (3.10) make new ellipsoid E_{k+1} as a bounding ellipsoid set, if E_k is a bounding ellipsoid set. This means if initial state estimation error E_1 is an ellipsoid set, then all state estimation errors E_k are ellipsoid sets.

Theorem 3.1 *If E_k in (3.6) is an ellipsoid set, we use (3.10) to update $\mathbf{P}$ and $\hat{\mathbf{x}}$ with initial condition (3.8), then E_{k+1} is also an ellipsoid set,*

$$E_{k+1} = \left\{ \mathbf{x}_{k+1} \mid \tilde{\mathbf{x}}_{k+1|k+1}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k+1} \leq 1 \right\} \tag{3.12}$$

and the volume of the ellipsoid set E_k converges to the volume of the uncertainty ellipsoid S_k,

$$\lim_{k \to \infty} \left[vol\left(E_k\right) - vol\left(S_k\right) \right] = 0 \tag{3.13}$$

Proof. The first part of the proof shows how the updating law (3.10) can assure E_{k+1} to be an ellipsoid set (3.12). Since E_k is assumed to be an ellipsoid set, from (3.6), $\tilde{\mathbf{x}}_k^T \mathbf{P}_k^{-1} \tilde{\mathbf{x}}_k \leq 1$. In order to verify $\tilde{\mathbf{x}}_{k+1|k+1}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k+1} \leq 1$, we should calculate $\tilde{\mathbf{x}}_{k+1|k+1}$ and $\mathbf{P}_{k+1|k+1}^{-1}$ using (3.10)

$$\begin{aligned} \tilde{\mathbf{x}}_{k+1|k+1} &= \mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1|k} - \frac{\lambda_{j,k}}{\gamma_j} \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k} \\ &= \tilde{\mathbf{x}}_{k+1|k} - \frac{\lambda_{j,k}}{\gamma_j} \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k} \end{aligned} \tag{3.14}$$

We apply the matrix inversion lemma [140] to calculate $\mathbf{P}_{k+1|k+1}^{-1}$ in the following way,

$$\left(A^{-1} - A^{-1} B \left(D A^{-1} B + C^{-1} \right)^{-1} D A^{-1} \right)^{-1} = A + BCD \tag{3.15}$$

Re-writing the equation (3.10),

$$\frac{\mathbf{P}_{k+1|k+1}^{-1}}{1 - \lambda_{j,k}} = [\mathbf{P}_{k+1|k} - \mathbf{P}_{k+1|k} \left(\nabla \mathbf{h}_{j,k} \frac{\lambda_{j,k}}{\gamma_j} \right) \times \frac{1}{(1 - \lambda_{j,k}) + \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k} \left(\nabla \mathbf{h}_{j,k} \frac{\lambda_{j,k}}{\gamma_j} \right)} \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k}]^{-1} \tag{3.16}$$

We take $A^{-1} = \mathbf{P}_{k+1|k}$, $B = \nabla \mathbf{h}_{j,k} \frac{\lambda_{j,k}}{\gamma_j}$, $C^{-1} = 1 - \lambda_{j,k}$ and $D = \nabla \mathbf{h}_{j,k}^T$, then it gives:

$$\mathbf{P}_{k+1|k+1}^{-1} = (1 - \lambda_{j,k}) \mathbf{P}_{k+1|k}^{-1} + \frac{\lambda_{j,k}}{\gamma_j} \nabla \mathbf{h}_{j,k} \nabla \mathbf{h}_{j,k}^T \tag{3.17}$$

Substitute $\tilde{\mathbf{x}}_{k+1|k+1}^T$ (3.14) into E_{k+1}

$$\begin{aligned} &\tilde{\mathbf{x}}_{k+1|k+1}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k+1} \\ &= \left[\tilde{\mathbf{x}}_{k+1|k} - \frac{\lambda_{j,k}}{\gamma_j} \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k} \right]^T \mathbf{P}_{k+1|k+1}^{-1} \left[\tilde{\mathbf{x}}_{k+1|k} - \frac{\lambda_{j,k}}{\gamma_j} \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k} \right] \\ &= \tilde{\mathbf{x}}_{k+1|k}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k} - 2 \frac{\lambda_{j,k}}{\gamma_j} \tilde{\mathbf{x}}_{k+1|k}^T \nabla \mathbf{h}_{j,k} e_{j,k} + \frac{\lambda_{j,k}^2}{\gamma_j^2} \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k}^2 \end{aligned} \tag{3.18}$$

Substitute $\mathbf{P}_{k+1|k+1}^{-1}$ (3.17) into E_{k+1}

$$\begin{aligned} &\tilde{\mathbf{x}}_{k+1|k+1}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k+1} = (1 - \lambda_{j,k}) \tilde{\mathbf{x}}_{k+1|k}^T \mathbf{P}_{k+1|k}^{-1} \tilde{\mathbf{x}}_{k+1|k} + \frac{\lambda_{j,k}}{\gamma_j} \tilde{\mathbf{x}}_{k+1|k}^T \nabla \mathbf{h}_{j,k} \nabla \mathbf{h}_{j,k}^T \tilde{\mathbf{x}}_{k+1|k} \ldots \\ &\ldots - 2 \frac{\lambda_{j,k}}{\gamma_j} \tilde{\mathbf{x}}_{k+1|k}^T \nabla \mathbf{h}_{j,k} e_{j,k} + \frac{\lambda_{j,k}^2}{\gamma_j^2} \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k+1} \nabla \mathbf{h}_{j,k} e_{j,k}^2 \end{aligned} \tag{3.19}$$

By the intersection property (3.11),

$$(1-\lambda_{j,k})\,E_k^T \le 1-\frac{\lambda_{j,k}}{\gamma_j}\left\|z_{j,k}^i - h_j(\hat{\mathbf{x}}_{k+1|k})\right\|^2 \tag{3.20}$$

Now we combine (3.18) and (3.19),

$$\begin{aligned}&\tilde{\mathbf{x}}_{k+1|k+1}^T\mathbf{P}_{k+1|k+1}^{-1}\tilde{\mathbf{x}}_{k+1|k+1} \le 1-\tfrac{\lambda_{j,k}}{\gamma_j}[\left\|z_{j,k}^i - h_j(\hat{\mathbf{x}}_{k+1|k})\right\|^2 ...\\ &... - \tilde{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}\nabla\mathbf{h}_{j,k}^T\tilde{\mathbf{x}}_{k+1|k} + 2\tilde{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}e_{j,k}] + ...\\ &... + \tfrac{\lambda_{j,k}^2}{\gamma_j^2}\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k+1}\nabla\mathbf{h}_{j,k}e_{j,k}^2\end{aligned} \tag{3.21}$$

We use $\tilde{\mathbf{x}}_{k+1|k} = \mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1|k}$, and $e_{j,k} = z_{j,k}^i - h_j(\hat{\mathbf{x}}_{k+1|k})$, the second term of the above equation can be calculated as

$$\begin{aligned}&\left\|z_{j,k}^i - h_j(\hat{\mathbf{x}}_{k+1|k})\right\|^2 - \tilde{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}\nabla\mathbf{h}_{j,k}^T\tilde{\mathbf{x}}_{k+1|k} + 2\tilde{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}e_{j,k} =\\ &= \left(z_{j,k}^i\right)^2 - 2\hat{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}z_{j,k}^i + \hat{\mathbf{x}}_{k+1|k}^T\nabla\mathbf{h}_{j,k}\nabla\mathbf{h}_{j,k}^T\hat{\mathbf{x}}_{k+1|k}\\ &= \left(z_{j,k}^i - \nabla\mathbf{h}_{j,k}^T\hat{\mathbf{x}}_{k+1|k}\right)^2\end{aligned} \tag{3.22}$$

Substitute (3.22) and $\lambda_{j,k}$ (3.10) into (3.21)

$$\begin{aligned}&\tilde{\mathbf{x}}_{k+1|k+1}^T\mathbf{P}_{k+1|k+1}^{-1}\tilde{\mathbf{x}}_{k+1|k+1}\\ &\le 1-\tfrac{\lambda_{j,k}}{\gamma_j}\left(1-\tfrac{\lambda_{j,k}}{\gamma_j}\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k+1}\nabla\mathbf{h}_{j,k}\right)e_{j,k}^2\\ &\le 1-\tfrac{\lambda_{j,k}}{\gamma_j}\left(1-\lambda_j\tfrac{\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k+1}\nabla\mathbf{h}_{j,k}}{1+\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}}\right)e_{j,k}^2\\ &= 1-\tfrac{\lambda_{j,k}}{\gamma_j}\left(\tfrac{1+\nabla\mathbf{h}_{j,k}^T\left(\mathbf{P}_{k+1|k}-\lambda_j\mathbf{P}_{k+1|k+1}\right)\nabla\mathbf{h}_{j,k}}{1+\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}}\right)e_{j,k}^2\end{aligned} \tag{3.23}$$

In order to prove $\tilde{\mathbf{x}}_{k+1|k+1}^T\mathbf{P}_{k+1|k+1}^{-1}\tilde{\mathbf{x}}_{k+1|k+1} \le 1$, we need $\left(\mathbf{P}_{k+1|k} - \lambda_j\mathbf{P}_{k+1|k+1}\right)$ is positive definite. Re-order the terms in (3.10), we obtain

$$\begin{aligned}&\lambda_j\mathbf{P}_{k+1|k+1} - \mathbf{P}_{k+1|k} = \tfrac{\lambda_j}{1-\lambda_{j,k}}[\left(1-\tfrac{1-\lambda_{j,k}}{\lambda_j}\right)\mathbf{P}_{k+1|k} - \tfrac{\lambda_{j,k}}{\left(1-\lambda_{j,k}\right)\gamma_j+\lambda_{j,k}\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}} \times ...\\ &... \times \mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}\left(\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}\right)^T]\end{aligned}$$

Because $\lambda_j < 1$ and $0 < \lambda_{j,k} < 1$, $1-\frac{1-\lambda_{j,k}}{\lambda_j} < 0$, $\frac{\lambda_{j,k}}{\left(1-\lambda_{j,k}\right)\gamma_j+\lambda_{j,k}\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}} > 0$, and the matrices $\mathbf{P}_{k+1|k}$ and $\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}\left(\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}\right)^T$ are positive definite, then $\lambda_j\mathbf{P}_{k+1|k+1} - \mathbf{P}_{k+1|k}$ is negative definite,

$$\nabla\mathbf{h}_{j,k}^T\left(\mathbf{P}_{k+1|k} - \lambda_j\mathbf{P}_{k+1|k+1}\right)\nabla\mathbf{h}_{j,k} \ge 0 \tag{3.24}$$

Therefore,

$$\tilde{\mathbf{x}}_{k+1|k+1}^T \mathbf{P}_{k+1|k+1}^{-1} \tilde{\mathbf{x}}_{k+1|k+1} \leq 1 - \frac{\lambda_{j,k}}{\gamma_j} \left(\frac{1}{1 + \nabla \mathbf{h}_{j,k}^T \mathbf{P}_{k+1|k} \nabla \mathbf{h}_{j,k}} \right) e_{j,k}^2 \leq 1 \tag{3.25}$$

E_{k+1} is an ellipsoid set. The ellipsoid intersection of E_k in (3.6) and S_k in (3.7) is (3.11), which is also ellipsoid. We defined this ellipsoid as

$$\begin{aligned} \Pi_k &= \left\{ \tilde{\mathbf{x}}_{k+1|k} \mid \tilde{\mathbf{x}}_{k+1|k}^T \Sigma^{-1} \tilde{\mathbf{x}}_{k+1|k}^2 \leq 1 \right\} \\ \Pi_k &= \left\{ \tilde{\mathbf{x}}_{k+1|k} \mid (1 - \lambda_{j,k-1}) \, \tilde{\mathbf{x}}_{k+1|k}^T \mathbf{P}_{k+1|k}^{-1} \tilde{\mathbf{x}}_{k+1|k} + \lambda_{j,k-1} S_k \leq 1 \right\} \end{aligned} \tag{3.26}$$

where Σ is a positive definite matrix. Although we cannot assure that the center of the the ellipsoid intersection Π_k is also on $\mathbf{x}_k$, since the centers of E_k in (3.6) and S_{1k} in (3.7) are $\mathbf{x}_k$, we can guarantee $\mathbf{x}_k$ is inside of Π_k. Now we discuss the relation between E_k and Π_k.Since

$$\Pi_{k+1} = \left\{ \tilde{\mathbf{x}}_{k+1|k+1} \mid (1 - \lambda_{j,k}) \, E_{k+1} + \lambda_{j,k} S_{k+1} \leq 1 \right\} \tag{3.27}$$

From (3.10) we know

$$\det \left[(1 - \lambda_{j,k}) \, \mathbf{P}_{k+1|k+1} \right] \leq \det \left(\mathbf{P}_{k+1|k} \right)$$

By definition (3.3), the volume of the part of E_{k+1}, which is inside the ellipsoid intersection Π_{k+1} in (3.27), is smaller than the volume of E_k, see Figure 3.4. Usually the volume of E_k is bigger than the volume of S_k. This behavior of (3.27) can be explained in two directions:

1. If $\lambda_{j,k} \to 0$, the main part of Π_{k+1} is E_{k+1}. In order to assure the main part of the intersection of E_{k+1} and S_{k+1} belongs to E_{k+1}, and the volume of this part is smaller than the volume of E_k, the volume of E_{k+1} should be smaller than the volume of E_k, see Figure 3.4-(a). So the volume of E_k becomes smaller with (3.10) in this case.

2. If $\lambda_{j,k} \to 1$, the main part of Π_{k+1} is S_{k+1}. In order to assure the main part of the intersection of E_k and S_k belongs to the smaller set S_k, E_k must moves to S_k, see Figure 3.4.

With the above two change directions and $0 < \lambda_{j,k} < 1$, (3.13) is established. ■

(3.13) means the estimation error of the Ellipsoidal SLAM proposed is not bigger than the upper bounds of the unknown uncertainties defined in (3.7).

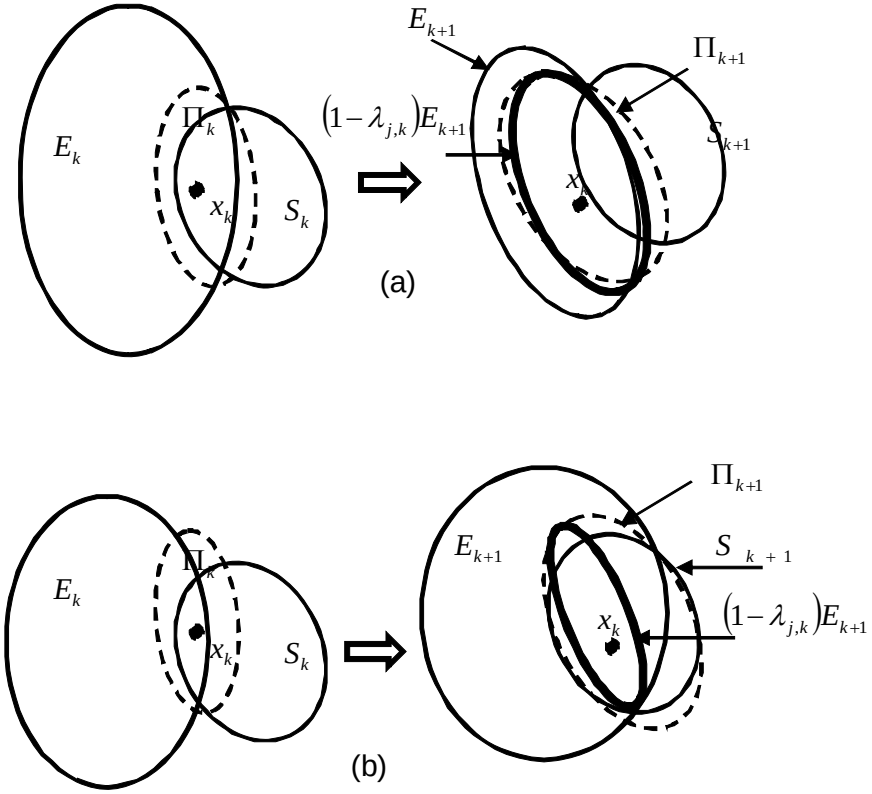


Figure 3.4: The ellipsoid intersections with the ellipsoid SLAM algorithm.

Remark 3.1 *If* $\mathbf{P}_{k+1|k+1}$ *does not change as* $\mathbf{P}_{k+1|k+1} = \mathbf{P}$*, it becomes gradient descent algorithm. The time varying gain* $\frac{\lambda_k}{\gamma_i}\mathbf{P}_{k+1|k+1}$ *in the ellipsoid algorithm may speed up the estimation process.*

The ellipsoid algorithm (3.10) has similar structure to the extended Kalman filter algorithm [118][47]

$$\begin{aligned}\hat{\mathbf{x}}_{k+1|k+1} &= \hat{\mathbf{x}}_{k+1|k} + \mathbf{P}_{k+1|k}\nabla\mathbf{h}_x^T\left(R_2 + \nabla\mathbf{h}_x^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_x\right)^{-1}\mathbf{e}_k \\ \mathbf{P}_{k+1|k+1} &= \mathbf{P}_{k+1|k} - \tfrac{\lambda_2}{\gamma_{2k}}\mathbf{P}_{k+1|k}\nabla\mathbf{h}_k\left(R_2 + \nabla\mathbf{h}_x^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_x\right)^{-1}\nabla\mathbf{h}_k^T\mathbf{P}_{k+1|k} + R_1\end{aligned} \tag{3.28}$$

where R_1 and R_2 are the covariances of the process noise. When $R_1 = 0$, it becomes the least square algorithm [140]. If $R_1 = 0$ and $R_2 + \nabla\mathbf{h}_x^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_x = \frac{\lambda_2}{\gamma_{2k}}$, in (3.28) is the ellipsoid algorithm (3.10). But there is a large difference, the ellipsoid algorithm does not require the disturbance is zero mean un-correlated Gaussian noises.

3.2.4 Algorithm of the Ellipsoidal SLAM

Every observation must be associated to a landmark on the map or be considered a new landmark. Due to the similarity between Kalman and ellipsoidal approaches, we can use the Mahalanobis distance as association criteria. The individual compatibility nearest neighbor is applied to calculate every distance between the observation and the predicted observations of all landmarks on the map [134]. The landmark closer to observation is associated if the distance is below to a threshold and the association is unique. If all landmarks are far from the observation, then it is considered a new landmark. (In Algorithm 4.1, $\mathbf{a}_k$ contains the results of associations).

At the beginning of map-building, the vector $\hat{\mathbf{x}}_k$ just contains the robot's state with no landmarks. As exploration increases, the robot detects landmarks and decides if it should add this new landmark for the state estimation.

If the new landmark is far from the other landmarks on the map, then the landmark is added, otherwise, it is ignored. We define a critical distance $d_{\min}$ to limit the maximal landmark density. This can reduce false positives in data association and avoid overload with the useless landmarks.

If the distance between the new landmark $\mathbf{x}_k^{new} = [x_{m+1}, y_{m+1}]$ and the others is bigger than $d_{\min}$, the landmark should be added into $\mathbf{x}_k$. Given the current state $\mathbf{x}_k^r$ and observation $\mathbf{z}_k$, we can calculate the new landmark position with respect to the absolute framework by

$$\mathbf{x}_k^{new} = g(\mathbf{x}_k^r, \mathbf{z}_k) \tag{3.29}$$

We can define the transformation in terms of the complete state

$$\mathbf{x}_{k+1} = \begin{pmatrix} \mathbf{x}_k \\ g(\mathbf{x}_k^r, \mathbf{z}_k) \end{pmatrix} = \mathbf{T}(\mathbf{x}_k, \mathbf{z}_k) \tag{3.30}$$

The nonlinear transformation function $\mathbf{T}$ also applies to the uncertainty set $\mathbf{P}_k$. Since a linear transformation of an ellipsoidal set is always ellipsoid [126], we approximate the transformation $\mathbf{T}$ by a linearization. Thus $\mathbf{P}_k$ in absolute framework can be expressed as

$$\mathbf{P}_k = \begin{pmatrix} \mathbf{P}_k^r & \mathbf{P}_k^{rm} & \mathbf{0} \\ (\mathbf{P}_k^{rm})^T & \mathbf{P}_k^m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{V}_k \end{pmatrix} \tag{3.31}$$

Similar to EKF SLAM [135],

$$\mathbf{P}_k = \nabla\mathbf{T}\mathbf{P}_k\nabla\mathbf{T}^T$$

where $\nabla\mathbf{T} = \begin{pmatrix} \mathbf{I}_r & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m & \mathbf{0} \\ \nabla\mathbf{g}_x & \mathbf{0} & \nabla\mathbf{g}_z \end{pmatrix}$, $\nabla\mathbf{g}_x := \frac{\partial \mathbf{g}}{\partial \mathbf{x}_k^r}(\mathbf{x}_k, \mathbf{z}_k)$, $\nabla\mathbf{g}_z := \frac{\partial \mathbf{g}}{\partial \mathbf{z}}(\mathbf{x}_k, \mathbf{z}_k)$.

[31] shows that it is possible to prune landmarks from estimation process without making it inconsistent in EKF SLAM. We exploit the same property in Ellipsoidal SLAM. Every K_z times, the landmarks with less number of corrections are removed from state vector and ellipsoid matrix. The vector $\mathbf{c}_k$ (see algorithm 3.1) saves this number for each landmark since its initialization.

The final Ellipsoidal SLAM algorithm can be described as

Algorithm 3.1 ***Ellipsoidal SLAM***

$\hat{\mathbf{x}}_1 = \mathbf{0}, \mathbf{P}_{1|1} = \alpha\mathbf{I}, k = 1, \alpha \gg 1$

$\mathbf{u}_1 = get\_controls$
$\mathbf{z}_1 = get\_observations; k_z = 1$
$[\hat{\mathbf{x}}_1, \mathbf{P}_1] = add\_features(\hat{\mathbf{x}}_1, \mathbf{P}_1, \mathbf{z}_1)$ *(3.31)*
while *not_stop*
  if *controls_are_available*
    $\left[\hat{\mathbf{x}}_{k+1|k}, \mathbf{P}_{k+1|k}\right] = prediction\left(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, \mathbf{u}_k\right)$ *(3.9)*
    $\mathbf{u}_k = get\_controls$
  end if
  if *observations_are_available*
    $\mathbf{z}_k = get\_observations$
   $\mathbf{a}_k = data\_association\left(\mathbf{z}_k, \hat{\mathbf{x}}_{k+1|k}, \mathbf{P}_{k+1|k}\right)$
    $\left[\hat{\mathbf{x}}_{k+1|k+1}, \mathbf{P}_{k+1|k+1}, \mathbf{c}_k\right]$
    $= correction\left(\hat{\mathbf{x}}_{k+1|k}, \mathbf{P}_{k+1|k}, \mathbf{z}_k\right)$ *(3.10)*
    $\left[\hat{\mathbf{x}}_{k+1|k+1}, \mathbf{P}_{k+1|k+1}\right]$
    $= add\_features\left(\hat{\mathbf{x}}_{k+1|k+1}, \mathbf{P}_{k+1|k+1}, \mathbf{z}_k\right)$ *(3.31)*
    $k_z = k_z + 1$
  end if
  if $\mathrm{mod}\,(k_z, K_z) = 0$
    $\left[\hat{\mathbf{x}}_{k+1|k+1}, \mathbf{P}_{k+1|k+1}\right]$
    $= pruning\left(\hat{\mathbf{x}}_{k+1|k+1}, \mathbf{P}_{k+1|k+1}, \mathbf{c}_k, \mathbf{a}_k\right)$
  end if
  $k = k + 1$
end while

3.3 Stability analysis

Theorem 2.1 tells us $E_{k+1} = \left\{\mathbf{x}_{k+1} \mid \tilde{\mathbf{x}}^T_{k+1|k+1}\mathbf{P}^{-1}_{k+1|k+1}\tilde{\mathbf{x}}_{k+1|k+1} \leq 1\right\}$ is a bounding ellipsoid if **A2** is satisfied. So the estimated states are bounded with the algorithm (3.10). The next theorem gives the bound of the estimation error $e^2_{j,k}$ which is defined in (3.10).

Theorem 3.2 *If we use the algorithm (3.10), then the normalization of the estimation error is bounded by:*

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{k=1}^{T}\frac{e_{j,k}^2}{1+\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}}\leq\frac{1}{\lambda_j\left(1-\lambda_j\right)} \tag{3.32}$$

where $0<\lambda_j<1$.

Proof. We define the following Lyapunov function

$$V(k)=\tilde{\mathbf{x}}_{k+1|k}^T\mathbf{P}_{k+1|k}^{-1}\tilde{\mathbf{x}}_{k+1|k} \tag{3.33}$$

Evaluating $\Delta V(k)$ as

$$\Delta V(k)=\tilde{\mathbf{x}}_{k+1|k+1}^T\mathbf{P}_{k+1|k+1}^{-1}\tilde{\mathbf{x}}_{k+1|k+1}-\tilde{\mathbf{x}}_{k+1|k}^T\mathbf{P}_{k+1|k}^{-1}\tilde{\mathbf{x}}_{k+1|k} \tag{3.34}$$

By (3.23) we have

$$\tilde{\mathbf{x}}_{k+1|k+1}^T\mathbf{P}_{k+1|k+1}^{-1}\tilde{\mathbf{x}}_{k+1|k+1}\leq 1-\frac{\lambda_{j,k}}{\gamma_j}\left(1-\lambda_j\right)e_{j,k}^2 \tag{3.35}$$

Substitute in (3.34)

$$\Delta V(k)\leq-\frac{\lambda_{j,k}}{\gamma_j}\left(1-\lambda_j\right)e_{j,k}^2+1-\tilde{\mathbf{x}}_{k+1|k}^T\mathbf{P}_{k+1|k}^{-1}\tilde{\mathbf{x}}_{k+1|k} \tag{3.36}$$

We know $1\geq 1-\tilde{\mathbf{x}}_{k+1|k}^T\mathbf{P}_{k+1|k}^{-1}\tilde{\mathbf{x}}_{k+1|k}\geq 0$. Then

$$\Delta V(k)\leq-\frac{\lambda_{j,k}}{\gamma_j}\left(1-\lambda_j\right)e_{j,k}^2+1 \tag{3.37}$$

Summarize (3.37) from 1 to T:

$$\sum_{k=1}^{T}\frac{\lambda_{j,k}}{\gamma_j}\left(1-\lambda_j\right)e_{j,k}^2\leq V(1)-V(T+1)+T\leq V(1)+T \tag{3.38}$$

Since $V(1)$ is constant and $\lambda_{j,k}=\frac{\lambda_j\gamma_j}{1+\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}}$,

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{k=1}^{T}\frac{e_{j,k}^2}{1+\nabla\mathbf{h}_{j,k}^T\mathbf{P}_{k+1|k}\nabla\mathbf{h}_{j,k}}\leq\frac{1}{\lambda_j\left(1-\lambda_j\right)} \tag{3.39}$$

It is (3.32). ■

Please note that even if the estimated states converge to their real values with the algorithm (3.10), from (3.4), we know that there always exist unmodeled dynamics (structure error). So we cannot reach $e_{j,k} = z^i_{j,k} - h_j(\hat{\mathbf{x}}_{k+1|k}) \to 0$.

3.4 Simulations and experiments

In this section, we evaluate the effectiveness of the Ellipsoidal SLAM through simulations and experiments with real data. We compare the novel Ellipsoidal SLAM with the extended Kalman filter (EKF) SLAM. The experiments are implemented in Matlab in a PC with the Intel Core i3-3.3Ghz processor.

3.4.1 A simple example

We will show that the Ellipsoidal SLAM has better performances than EKF SLAM in some conditions. In the case of a linear system, the ellipsoid filter has smaller errors when process noise has skewness and bias [136]. However, the SLAM model is a nonlinear system. So we ran a set of Monte Carlo simulations to test this property. We use the following metrics and conditions in our simulations.

Metrics. The three metrics for evaluating the performance over N_{mc} number of Monte Carlo simulations are defined as

$$\begin{aligned} E_d &= \tfrac{1}{N_{mc}} \textstyle\sum_j^{N_{mc}} \sum_k \sqrt{\left(x^r_{k,j} - x^{r*}_{k,j}\right)^2 + \left(y^r_{k,j} - y^{r*}_{k,j}\right)^2} \\ E_a &= \tfrac{1}{N_{mc}} \textstyle\sum_j^{N_{mc}} \sum_k \left|\theta^r_{k,j} - \theta^{r*}_{k,j}\right| \\ E_l &= \tfrac{1}{N_{mc}} \textstyle\sum_j^{N_{mc}} \sum_k \sum_i \sqrt{\left(x^i_{k,j} - x^i_{k,j}\right)^2 + \left(y^i_{k,j} - y^i_{k.j}\right)^2} \end{aligned} \tag{3.40}$$

where j is a particular simulation, k is the time step, i is a specific landmark, $*$ represents the true values, E_d and E_a are the average of Euclidean and orientation errors, E_l is the average of Euclidean errors of landmarks.

Conditions. We assume that the data association is known and the process model error can be explained totally by noise with respect to controls. The virtual vehicle executes

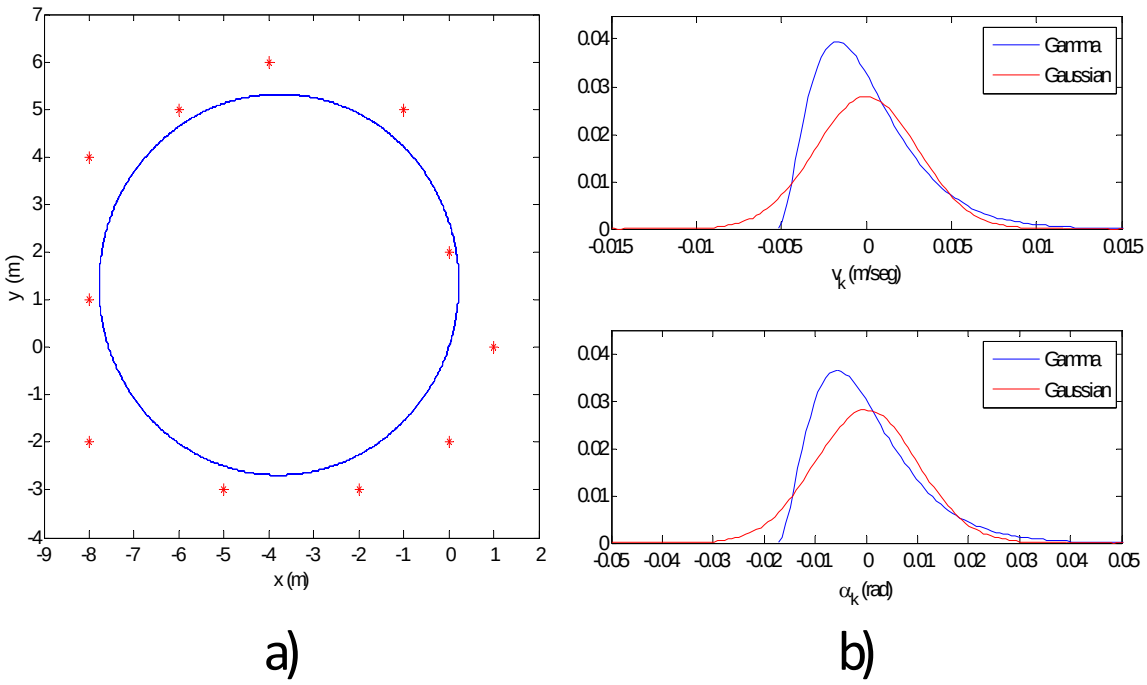


Figure 3.5: (a) The virtual environment and robot path, (b) Gamma and Gaussian probability distributions for controls.

a circle path to evaluate the long-term performance, where controls are $\alpha_k = 1.0rad$ and $v_k = 0.3m/seg$. The range-bearing sensor has a field of view determined by $r \in [0.5, 30]\, m$ and $\phi \in [0, \pi]\, rad$. The sample time for filtering is $T_k = 0.1seg$. The landmark distribution is shown in Figure 3.5. We adjust manually the covariance and ellipsoid parameters before tests. Thus the results are the best we could find for each method.

We explore three conditions of noise for motion model, where control noises are independent.

Condition 1 (Gaussian + bias). Gaussian noise with the standard deviation is 1% of the control magnitudes, *i.e.*, $\sigma_\alpha = 0.003rad$, $\sigma_v = 0.01m/seg$. We also add biases $\mu_\alpha = 0.05rad$ and $\mu_v = 0.05m/seg$ to the Gaussian noise.

Condition 2 (Gamma). We select the gamma distribution to study the effect of skewness. Their parameters are chosen such that the standard deviations are the same as the above Gaussian noise. The mean is compensated to be zero. Figure 3.5 shows the differences between Gaussian and Gamma distributions for both controls. Each control noise has about 1.55 skewness. The specific parameters are: $a_\alpha = 3$, $b_\alpha = 5.773 \times 10^{-3}$, $a_v = 3$, $b_v = 1.732 \times 10^{-3}$.

Condition 3 (Gamma + bias). Here we study the jointly effect of skewness and bias, using the above gamma distributions together with bias $\mu_\alpha = 0.05rad$ and $\mu_v = 0.05m/seg$.

The observation noise is assumed to be Gaussian, and the same for all three conditions, *i.e.*, $\sigma_r = 0.1m$, $\sigma_\phi = 0.01rad$, $\mu_r = \mu_\phi = \sigma_{r\phi} = 0$.

Figure 3.6 shows the results of $N_{mc} = 50$ simulations for every condition. We note that the EKF-SLAM generates small errors when there is no bias in control noises (Condition 2). But our method is better when the control noise contains bias under Condition 1 and Condition 3. Figure 3.7 also shows how the estimation errors of the EKF SLAM increase under Gaussian noise and bias (Condition 1), and the estimation errors of the Ellipsoidal SLAM keep the same after the third turn (a turn over circle is completed in 40 second). So the Ellipsoidal SLAM is more robust when there is bias in control noises.

	Experiment	Ed	Ea	El
1	EKF-SLAM	1.92E+03	504.74	2.41E+04
	Ellipsoidal	1.13E+03	296.67	1.25E+04
2	EKF-SLAM	299.2	78.25	3.69E+03
	Ellipsoidal	493.07	96.97	6.07E+03
3	EKF-SLAM	1.91E+03	501.05	2.39E+04
	Ellipsoidal	997.68	270.22	1.08E+04

Figure 3.6: We show the errors of EKF SLAM and Ellipsoidal SLAM for the three different conditions over 50 Monte Carlo simulations. Our method is more robust under bias in control noises.

3.4.2 Victoria Park dataset

For validating the Ellipsoidal SLAM with real and noisy measurements, we test it with a benchmark, called Victoria Park dataset [137]. This dataset is a classical benchmark for feature-based SLAM methods, because the environment is large ($250\times300m^2$), the trajectory is long ($4.5km$), and there are many loops (~14 loops). Besides, the observations have much spurious detections of trees. All these qualities make it a really difficult problem. It is used in the robotics community to make comparisons.

First, we present the models used for this experiment. The model of the SLAM system consists: motion and observation equations. For motion model, we have an Ackerman vehicle described by the unicycle model as

$$\begin{pmatrix} x_k^r \\ y_k^r \\ \theta_k^r \end{pmatrix} = \begin{pmatrix} x_{k-1}^r + T_{k-1} v_{k-1} \cos\theta_{k-1}^r \\ y_{k-1}^r + T_{k-1} v_{k-1} \sin\theta_{k-1}^r \\ \theta_{k-1}^r + T_{k-1} \frac{v_{k-1}}{b_a} \tan\alpha_{k-1} \end{pmatrix} + \mathbf{w}_k \tag{3.41}$$

where $\mathbf{w}_k \in \Omega(0, \mathbf{W}_k)$ is the process noise, v_k is the linear velocity, α_k the steering angle, and T_k the sample time.

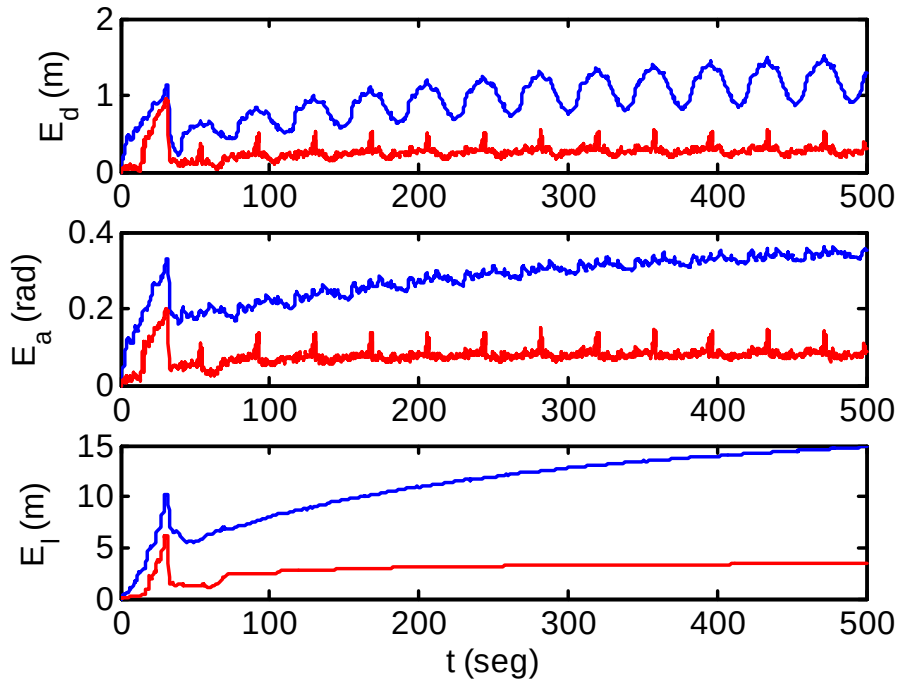


Figure 3.7: An simulation under condition 1 (red lines - Ellipsoidal SLAM, blue lines - EKF-SLAM). We see that errors remain bounded for our method, while the errors for EKF-SLAM increase over time.

The observation model is

$$z_k^i = \begin{pmatrix} r_k^i \\ \phi_k^i \end{pmatrix} = \begin{pmatrix} \sqrt{\left(x_k^i - x_k^r\right)^2 + \left(y_k^i - y_k^r\right)^2} \\ \arctan\left(\frac{y_k^i - y_k^r}{x_k^i - x_k^r}\right) - \theta_k^r + \frac{\pi}{2} \end{pmatrix} + \mathbf{v}_k \tag{3.42}$$

The observation function (3.29) is

$$g(\mathbf{x}_k^r, \mathbf{z}) = \begin{pmatrix} x_k^r + r_k^i \cos\left(\phi_k^i + \theta_k^r - \frac{\pi}{2}\right) \\ y_k^r + r_k^i \sin\left(\phi_k^i + \theta_k^r - \frac{\pi}{2}\right) \end{pmatrix} \tag{3.43}$$

Figure 3.8 shows the map and trajectory generated by our Ellipsoidal SLAM, and the comparison results with the EKF SLAM. In both methods, the free parameters (covariance matrices of noises and errors bounds) were chosen by trial and error.

From the best of our knowledge, it is the first time that such large-scale SLAM problem is solved by a non-probabilistic technique. The other SLAM methods based on sets have been validated only with small environments and trajectories [124], or with manual data association [122], while the Ellipsoidal SLAM associates data automatically and can solve large-scale problems, such as Victoria Park benchmark.

In some parts, the Ellipsoidal SLAM has similar results to the EKF SLAM. But the Kalman approach is a little more exact than the Ellipsoidal SLAM (e.g., in upper-left corner of Figure 3.8). There are some disadvantages for the Ellipsoidal SLAM. (1) When the noise level is small or the noise behavior is like Gaussian, the EKF SLAM is better because Kalman filter is the optimal estimation technique with respect to linear system and white noise, while the ellipsoid filter is not optimal in any case. (2) The EKF SLAM solves faster than the Ellipsoidal SLAM. In this example, the EKF SLAM uses $788seg$ to map 111 landmarks, while the Ellipsoidal SLAM uses $1086seg$ to map 123 landmarks. Since the vehicle takes $1548seg$ for recording the dataset, this is enough to run on-line. And (3) the trial and error method to choose the free parameters could be more difficult for Ellipsoidal SLAM.

3.4.3 Koala mobile robot

In order to validate the performances of the Ellipsoidal SLAM, we use the mobile robot Koala [138], which is a mid-size robot designed for real-world applications, in two types of

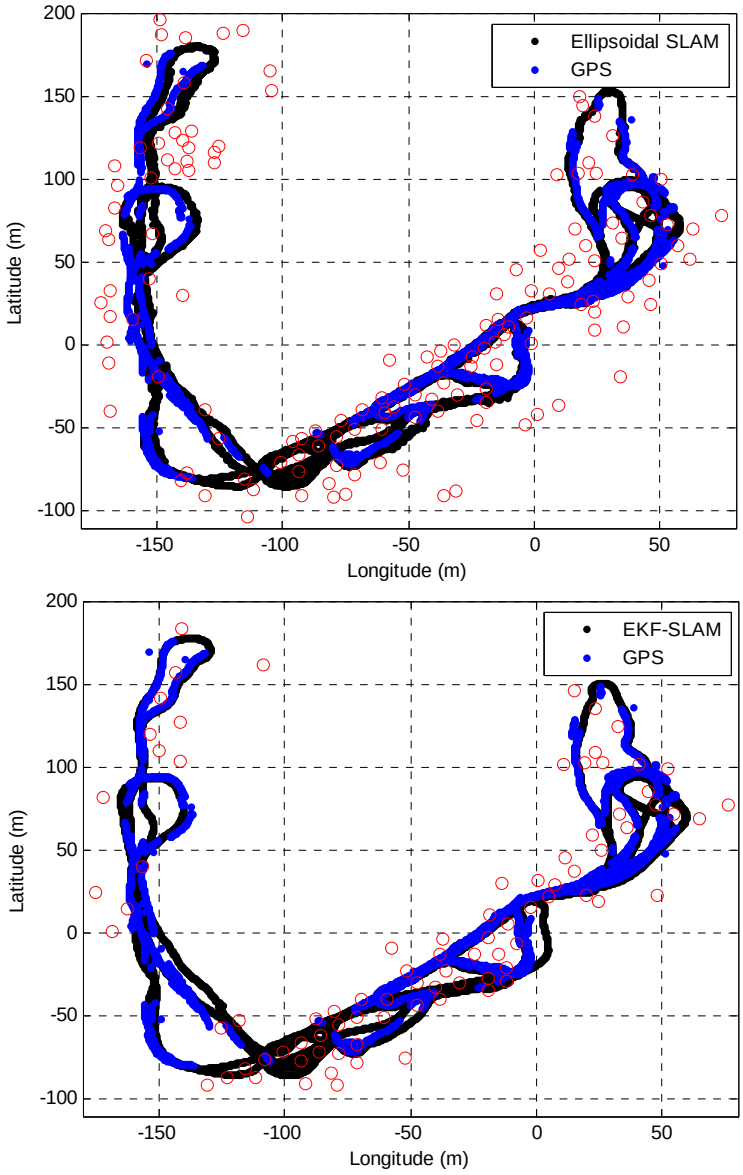


Figure 3.8: Comparison of the proposed Ellipsoidal SLAM and EKF SLAM with Victoria Park dataset. The red circles are landmarks detected during the journey, most of them represent trees in the Victoria Park. In some parts, ellipsoid SLAM has similar results as EKF SLAM. But the Kalman approach is a little more exact than Ellipsoidal SLAM

Figure 3.9: The experiment in an office environment equiped a motion capture system

environments. The Koala robot has two encoders to reconstruct odometry, and one laser range finder to detect natural features in the environment.

The first experiment is inside a room (see Figure 3.9). The room contains some chairs, tables, and one wastebasket. In order to verify our algorithm, we use a motion capture system, OptiTrack [139], in the room. It includes 16 cameras and one center processing computer. The position accuracy of this motion capture system is $\pm 5mm$. The Ellipsoidal SLAM uses the measurement of the odometer and laser scan of Koala to generate estimation states. The results of the Ellipsoidal SLAM are shown in Figure 3.10. Here, "+" is initial position, "o" is the end position, and the circles are the landmark detected by the SLAM. We find that there is odometer error. By the data association and landmark detection of our Ellipsoidal SLAM, the loop closure problem is overcome.

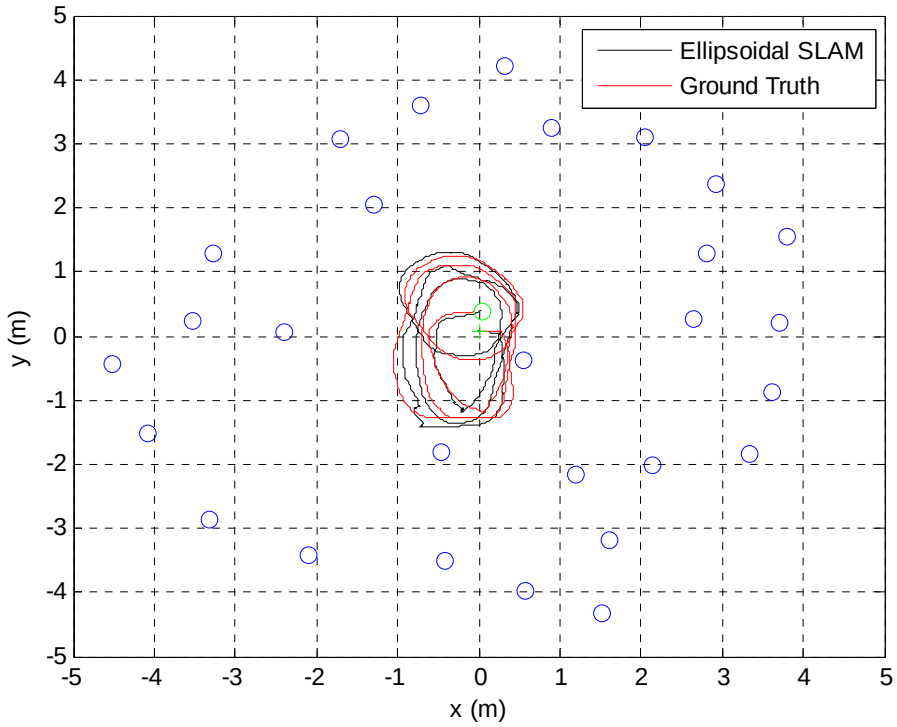


Figure 3.10: The map and trajectories generated by the ellipsoidal SLAM.

Figure 3.11 shows the square of the estimation error. The sensor accuracy of the laser range finder is $\leq 0.1m$. The accuracy of the Ellipsoidal SLAM is $\leq 0.31m$. The results are satisfied. It can also been seen, even if the environment is small, the algorithm can detect enough number of landmarks (28 landmarks).

The second experiment is in a larger natural environment, see Figure 3.12-(a). It is a yard with trees and light poles. The soil is irregular with depressions. This environment is harder than the office environment because of the ground's irregularity. The laser rays can hit on the ground, and create false detection of landmarks. The odometer is also affected by these depressions.

In this experiment, we do not have truth map and positions. We only check the loop

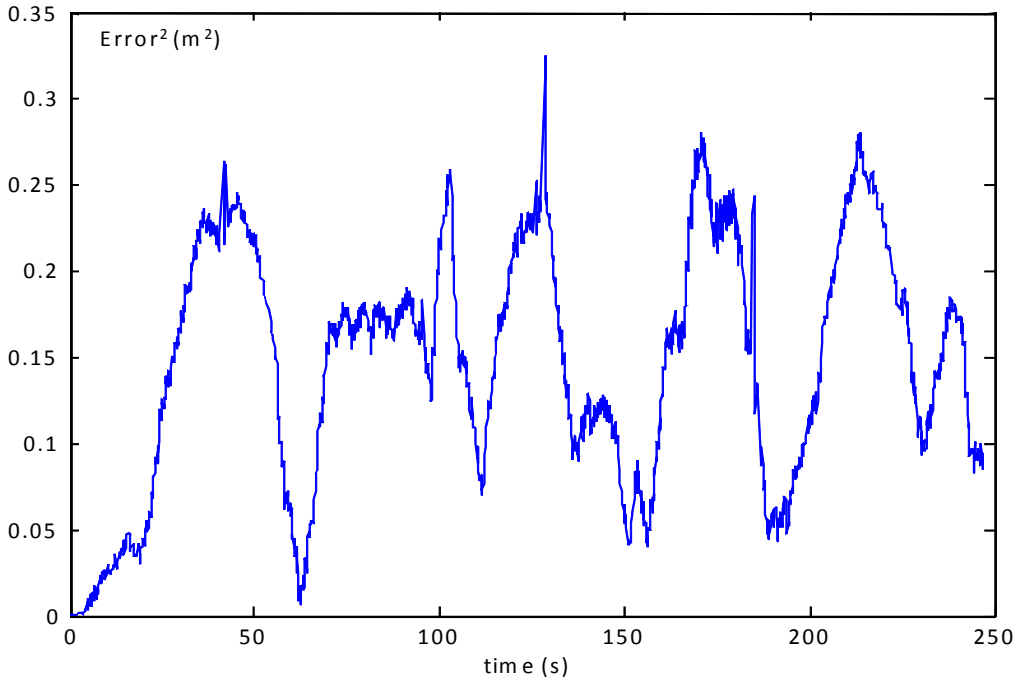


Figure 3.11: We show the square of the estimation error of the ellipsoidal SLAM.

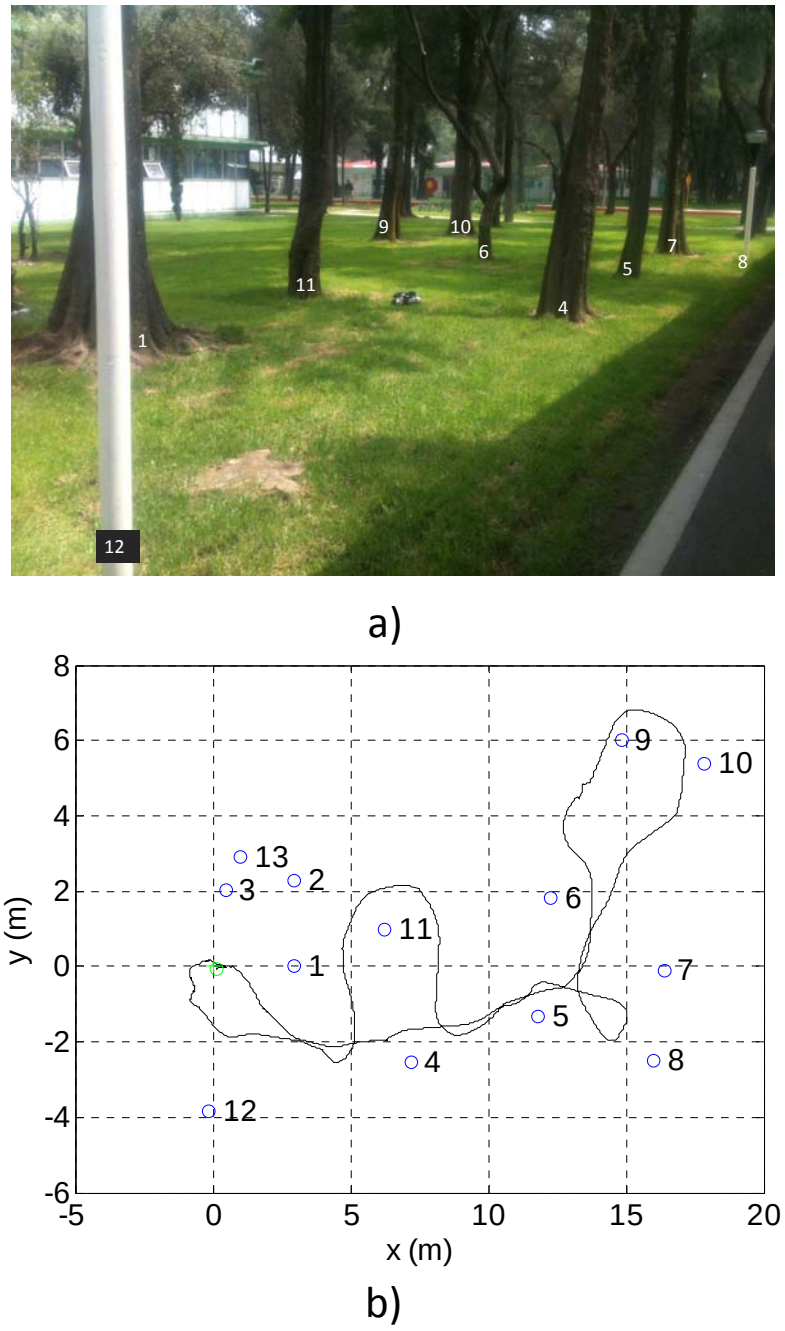


Figure 3.12: The experiment in (a) outdoor environment and (b) the map and trajectory generated by the mobile robot. The landmarks (blue circles) qualitatively correspond to the trees and light poles in the environment.

closure problem. Figure 3.12-(b) shows the map and trajectory of the mobile robot. We can see that every landmark has a number that relates to the trees or the light poles in the picture of environment. The relations between the landmarks and objects in the environment are qualitatively correct. In Figure 3.12-(b), we show that the estimate trajectory returns to its start point.

From these two experiments, we conclude that the Ellipsoidal SLAM can map different environments and estimate the robot's position with accuracy less than $0.4m$.

3.5 Conclusion

Unlike the other SLAM methods based on sets, the Ellipsoidal SLAM proposed is able to solve large-scale problems. Furthermore, its performance is similar to the EKF SLAM with no Gaussian assumption, which means we have more realistic error modeling. One contribution of this work is that the convergence of the ellipsoid algorithm is proven. We also propose a data association method which is similar to the individual compatibility nearest neighbor test in the EKF SLAM. The simulated results showed that the Ellipsoidal SLAM could be more robust to motion model errors with bias. And the experimental results show it is capable of building maps indoors and outdoors with enough number landmarks.

This chapter introduces the idea that large-scale SLAM problems can be solved in real-time by not assuming Gaussianity. However, there are still technical problems, like how to calibrate the Ellipsoidal SLAM parameters and how to improve accuracy. We hope someone interest in this work could retake it and solve these problems.

Chapter 4

Autonomous navigation in dynamic and unknown environments

"Our outdated education system defines intelligence as a process of memorizing old answers to avoid errors. True intelligence is to learn to solve our problems to face greater challenges. True intelligence is about the joy of learning and not with the fear of failure." —Robert Kiyosaki

4.1 Introduction

This chapter is related to motion planning and how to deal with **the partial observability of the environment**. The problem of autonomous navigation in static environments is currently solved [142]. Navigation systems are complete and optimal in static environments. Complete means that the navigation system ensures achieving the goal, if a feasible path exists in the environment to reach the goal. Optimal means that the navigation system is able to find the shortest path to reach the goal.

In contrast, dynamic and unknown environments represent a great challenge due to the partial observability of the environment. Dynamic environments contain objects that can change its shape, position, or orientation (e.g., furniture, doors, people, animals, cars).

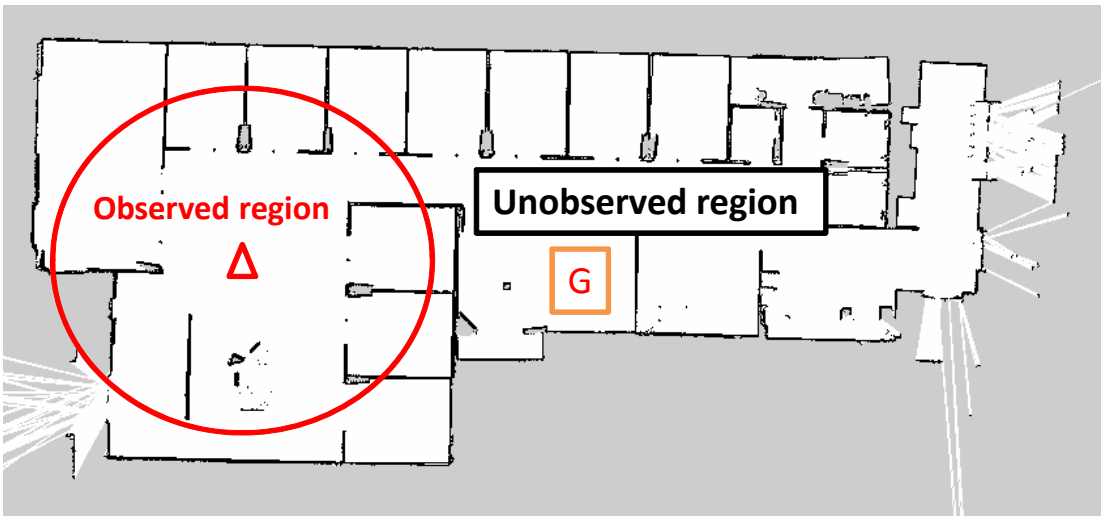


Figure 4.1: This figure illustrates the partial observability of the environment. The robot is represented by the triangle and the observed region by a circle. Due to the partial observability, a conventional navigation system is neither optimal nor complete.

An unknown environment is when the navigation system has no information about the environment; it does not have a map. In both cases, the navigation system needs to make certain assumptions about how the environment that has not been observed is. This is because all the sensors have limits on its measuring range—for example, a laser can only see within the range of 0.05m to 10m. So the robot can only sense what is around it; this region of the environment is called the observed region (see Figure 4.1). Thus, navigation systems must have certain beliefs about the unobserved region in order to complete the navigation task. The problem is that such beliefs may be erroneous when the environment is dynamic or unknown. In other words, there may be differences between the map (belief) and the environment (reality). **These differences (errors) cause any navigation system to no longer be optimal nor complete**.

There are only three possible assumptions that a navigation system can consider when it plans movements in the unobserved region (see Figure 4.2):

1. Known Space Assumption (KSA). The whole space is supposed to be known, including the unobserved space. This means that the whole environment is known via a map generated previously [147][148]. The navigation system can plan according to this map.

2. Free Space Assumption (FSA). This concept was introduced by S. Koenig et al. [153]. The unobserved space is supposed to be free of obstacles. It is often used when there is no previous map [142][149] to plan paths to the goal. The robot executes the plan until it finds an obstacle in its path, then plans a new path to avoid the obstacle. This process is repeated until the goal is reached.

3. Unknown Space Assumption (USA). The unobserved space is accepted as unknown, so we cannot make plans over that region. It is often used for autonomous map-building tasks [151][152], not for autonomous navigation tasks. The robot plans to expand the observed region. So it picks a point on the border between the observed and the unobserved regions as a partial goal. Once the robot has approached this partial goal, the observed region has increased. This process is repeated until the main goal is within the observed region and the robot can reach it.

Most navigation systems assume the workspace is known (KSA), the environment does not change, and the map does not have errors [147][148]. Other times when the map is not previously built, it assumes the environment is free of obstacles (FSA) [142][149], allowing the robot to make plans moving toward the unobserved area and finding the actual obstacles. And finally, when the robot must build a map [151][152] instead of reaching a specific goal state, we use the Unknown Space Assumption, restricting the robot to plan its movements only inside the known area. Although these three assumptions have been used widely, there are no studies focused on these assumptions themselves, to the best of our knowledge.

In this chapter, we discuss how these assumptions affect navigation, given formal definitions for the autonomous navigation problem. In **section 4.2**, we present the theorems of sufficient conditions to guarantee a navigation system is complete. In **section 4.3**, we propose the first methods to recover from false obstacles on the map in dynamic environments. In **section 4.4**, we propose a method to navigate more efficiently in unknown environments

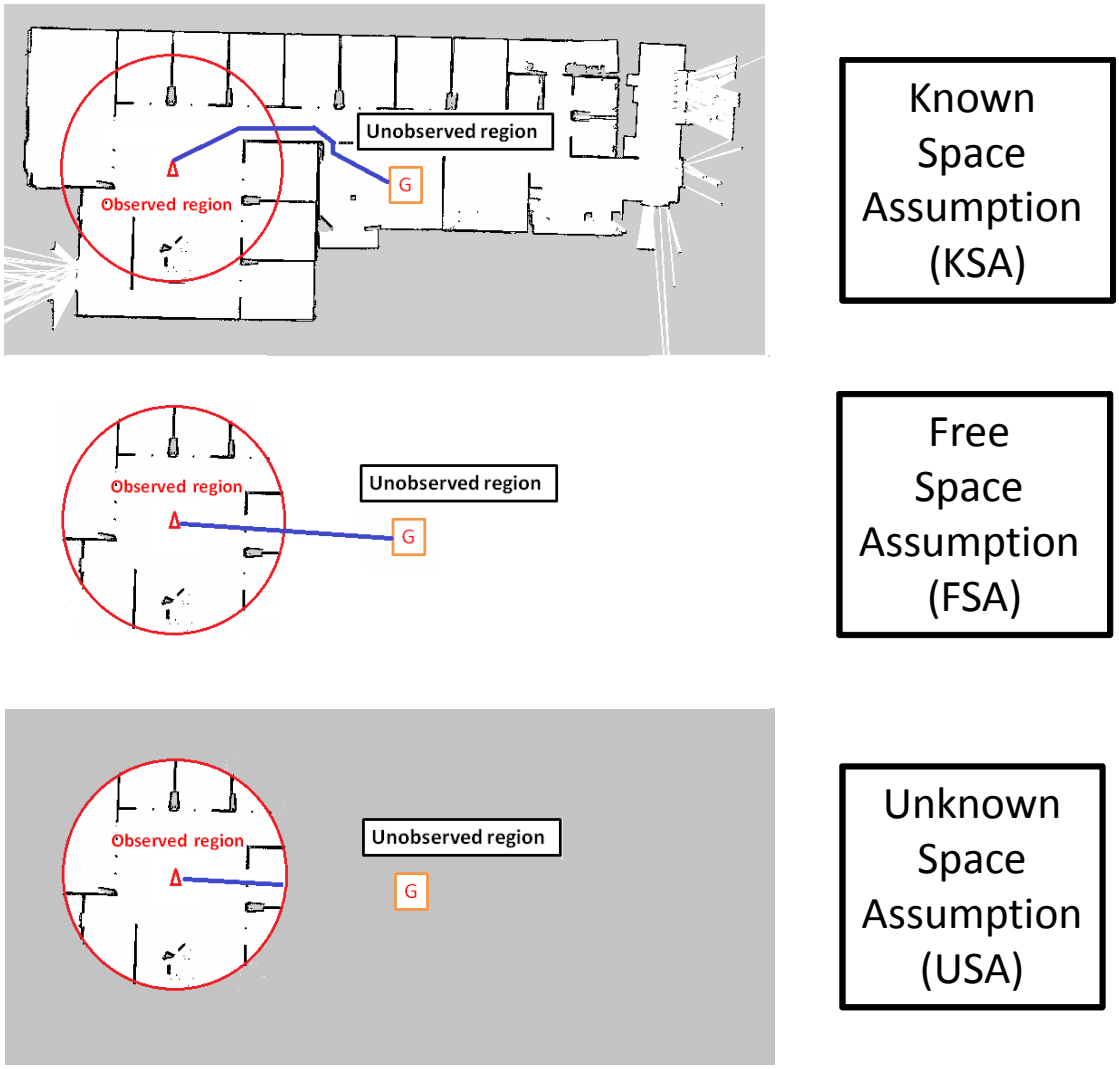


Figure 4.2: These are the possible assumptions that a navigation system can consider when planning movements in the unobserved region. The unobserved region is supposed to be known (KSA), free of obstacles (FSA), or unknown (USA). These assumptions change the way a navigation system works (robot [triangle], observed region [circle], blue line [path planning], goal [G]).

and we compare with the classical method of the FSA. Finally, we give our conclusions in **section 4.5**.

4.2 Analysis for navigation

Due to the partial observability of the environment, there could be two types of errors in a map: false obstacles and false free spaces (see Figure 4.3). False obstacles are obstacles that appears on the map but they do not exist in the real environment. False free spaces are real obstacles in the environment, but they do not appear on the map. Both types of errors cause the planning system to produce suboptimal paths. But that is not the biggest problem. In addition, the planning system may fail to find a path to reach the goal due to false obstacles on map. In this case, a conventional navigation system (a system without the methods that will be presented in the next section) would be incomplete (i.e., could not reach the goal even if a viable route exists in the real environment). In this section, we will give sufficient conditions to ensure a navigation system is complete.

We use the graph theory to describe the autonomous navigation problem. So we analyze the sufficient conditions to guarantee a navigation system reaches its goal when the map can have errors. For simplicity, our analysis considers the environment is static. But this is not a limitation, since the results give us a practical idea of what happens in dynamic environments. First, we introduce a series of definitions. Later, we present the theorems for sufficient conditions.

The main elements of autonomous navigation are the following:

- Environment. It is defined by an undirected graph $G_{env}(t) = (S_{env}, E_{env}(t))$. Here S_{env} is the discrete version of the state space, and the edges $E_{env}(t)$ represent validated state transformation actions of the robot; $G_{env}(t)$ means the graph is dynamic. So the obstacles can be easily represented by the absence of edges within a state. The environment is assumed to be finite.

- Navigation task. It consist of a given initial state s_I, a goal state s_G, and a map of

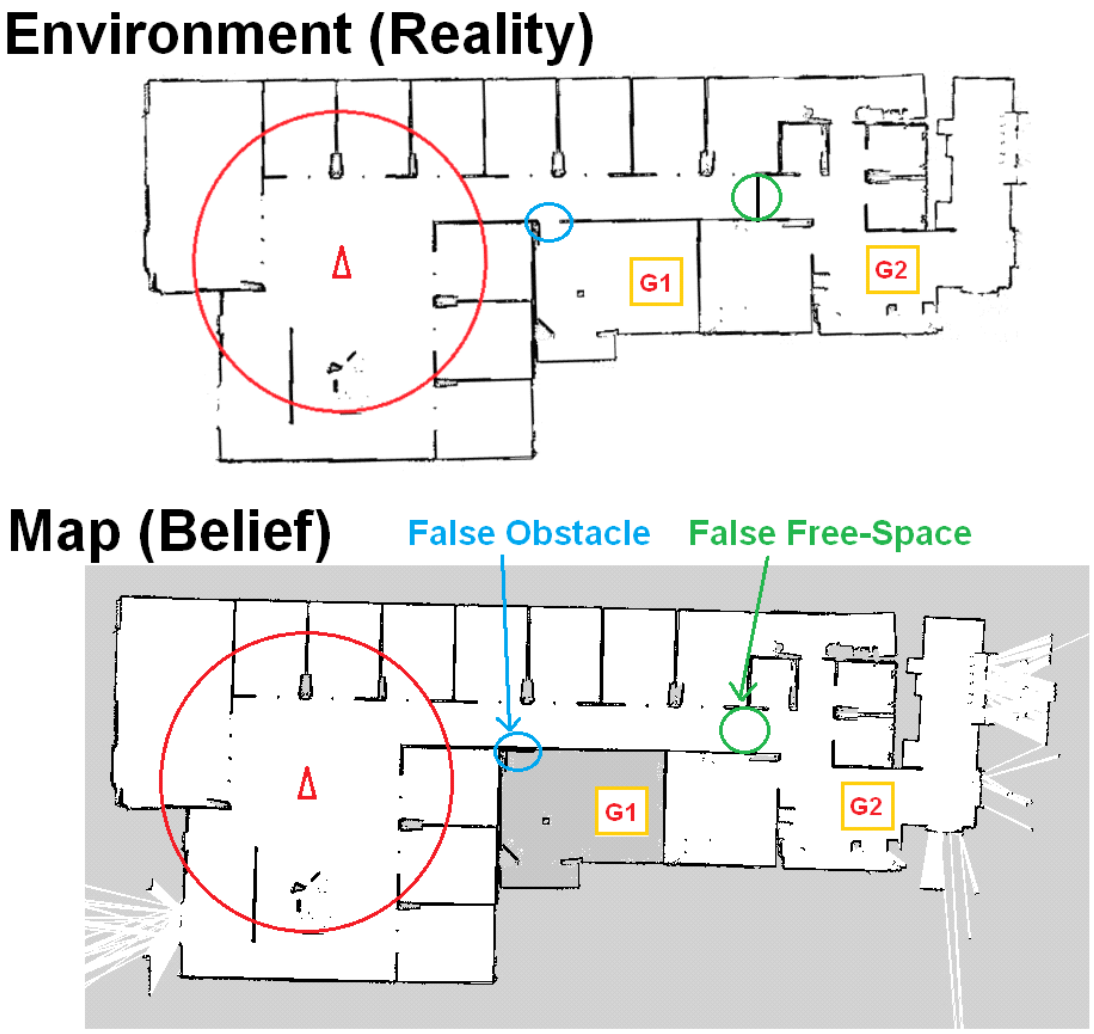


Figure 4.3: This figure illustrates the two types of errors that could be on a map because of the partial observability of the environment. Note the differences between the belief of the robot (map) and the reality (environment).

environment $G_{map}(t) = (S_{map}, E_{map}(t))$. The objective of navigation is that the robot can reach the goal state in a finite time. The map $G_{map}(t)$ is an undirected graph corresponding to the environment $G_{env}(t)$.

- Robot. It is an entity that does the navigation task.

In order to reach the goal, some assumptions have to be made to plan a path. Figure 4.4 shows the graphic explanations of these assumptions. We give formal definitions for them as follows:

1. Known Space Assumption (KSA). The whole environment is known. The accessibility or obstacle function $acc(s_i)$ is known for $\forall s_i$, where $acc(\cdot)$ represents the obstacles on the map and the environment. We define

$$acc(s) = \begin{cases} 1 & deg(s) > 0 \\ 0 & deg(s) = 0 \end{cases}$$

where $deg(\cdot)$ is the number of incident edges (the edges that are connected to a state), so $acc(s) = 0$ means the state s is an obstacle.

2. Free Space Assumption (FSA). The unobserved region is free of obstacles. The $acc(s_i)$ is known for $\forall s_i \in S_O(t)$ and $acc(s_i) = 1$ for $\forall s_i \in S_{UO}(t)$. Where $S_{UO}(t)$ is the unobserved region and $S_O(t)$ is the observed region. We define $S_O(t_k) = S_{t_0} \cup S_{t_1} \cup ... \cup S_{t_k}$, where every S_t is a subset of states that are observed by the mapping system around the robot at time t.

3. Unknown Space Assumption (USA). The unobserved region is unknown. The $acc(s_i)$ is known for $\forall s_i \in S_O(t)$, and $acc(s_i)$ is unknown for $\forall s_i \in S_{UO}(t)$.

Usually, the map $G_{map}(t)$ may be different with the environment $G_{env}(t)$, due to partial observability. In other words, map and environment can have errors due to the mobile objects in the environment and the errors in localization, in mapping, or in perception (e.g., some of

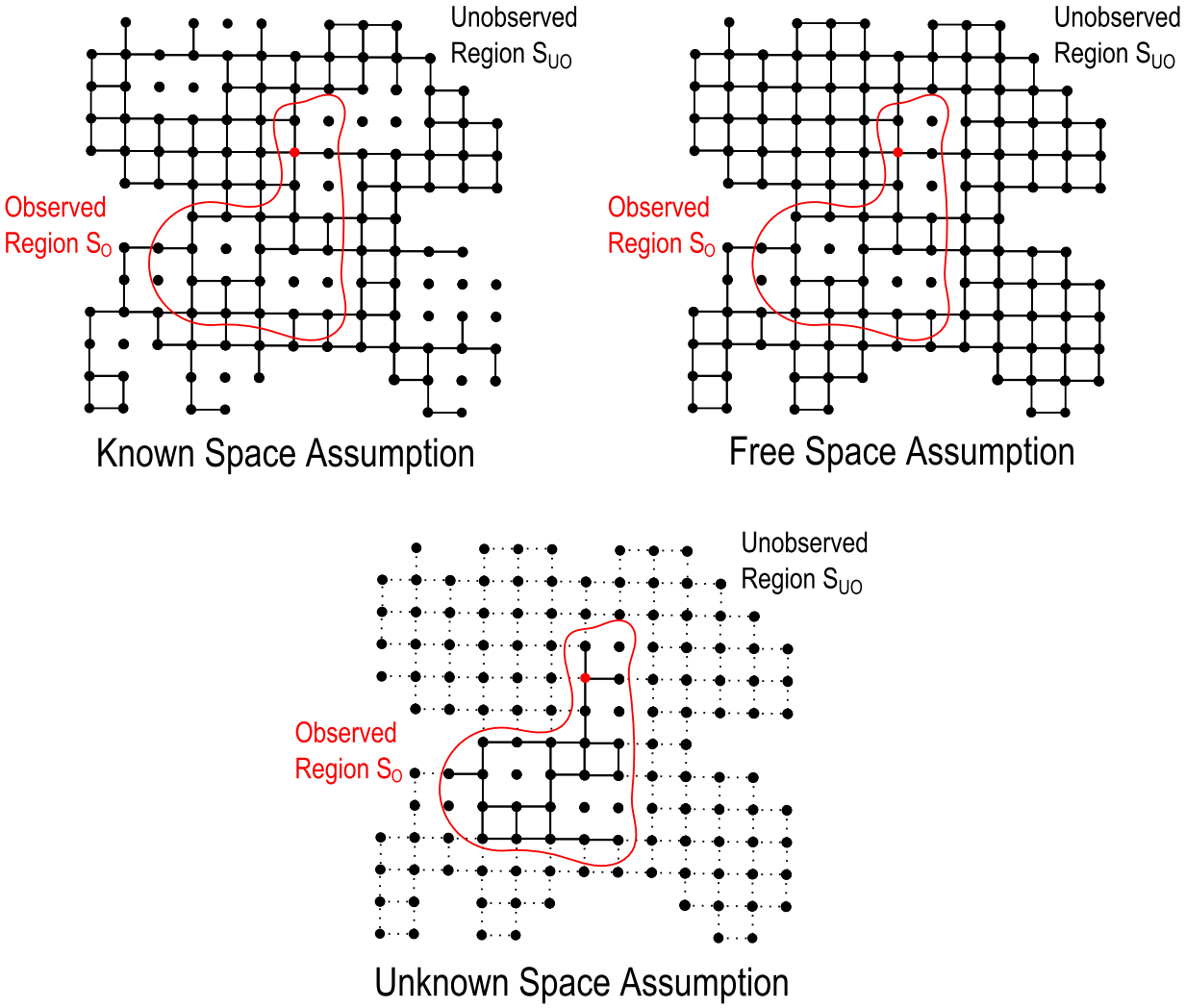


Figure 4.4: Assumptions for the unobserved part of the environment using graph representation.

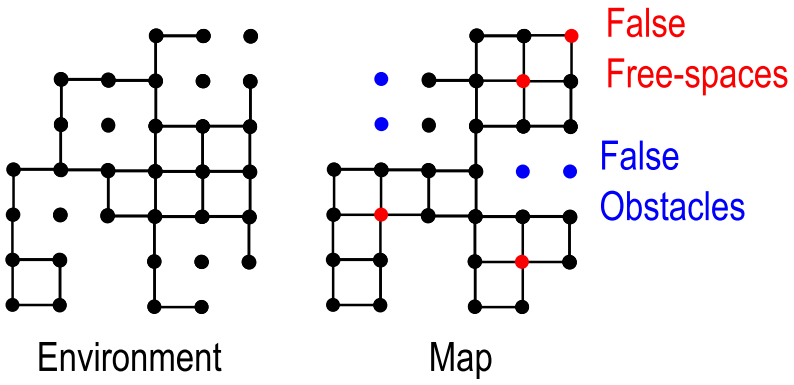


Figure 4.5: The two types of errors in the map are represented in a graph.

current SLAM techniques are not reliable for the long term [154]). Since the robot cannot observe the whole environment, it is impossible to obtain the optimal path, and even worse, the planning system could not find a path, even if a path exists. We consider two types of errors on the map (see Figure 4.5):

- We define the number of false obstacles as $FO(G_{map}(t)) = \sum_{\forall s \in S_{map}(t)} f_{FO}(s)$, where
$$f_{FO}(s_i) = \begin{cases} 1 & acc(s_i^{map}) = 0 \wedge acc(s_i^{env}) = 1 \\ 0 & otherwise \end{cases}$$
- We define the number of false free spaces as $FF(G_{map}(t)) = \sum_{\forall s \in S_{map}(t)} f_{FF}(s)$, where
$$f_{FF}(s_i) = \begin{cases} 1 & acc(s_i^{map}) = 1 \wedge acc(s_i^{env}) = 0 \\ 0 & otherwise \end{cases}$$

Because a navigation system consists of several subsystems, and we just want to analyze the problem of errors on the map, we will consider the following assumptions and definitions.

1. Effective Localization. $s_r^{map}(t) = s_r^{env}(t)$ for $\forall t$; here s_r^G means the current state of the robot in graph G.

2. Effective Mapping. $acc(s_i^{map}) = acc(s_i^{env})$ for any observed state $s_i \in S_t \neq \varnothing$ at any time t.

3. Complete Planner. If $P_{sol}^{map} \neq \varnothing$, then the planner always gives a path solution $p_{sol}^{map} = \{s_I = s_0, s_1, ..., s_n = s_G\}$; here P_{sol}^{G} is the set of vertices of a connected component of G that contains both s_I and s_G.

4. Effective Execution. After robot has a path plan $p_{sol}^{map} = \{s_I = s_0, s_1, ..., s_n = s_G\}$, it executes the first k states in G_{env} until it reaches s_G or it finds an obstacle in the path (i.e., $\exists s_i \in p_{sol}^{map}$ such that $acc(s_i^{env}) = 0$ for $i > k$).

5. Effective Planner. It is a complete planner that generates a new p_{sol}^{map} every time it finds a new obstacle or tells that there is no path.

6. Effective Robot. A robot is effective if localization, mapping, planner, and execution are effective.

7. The navigation system is complete for a navigation task $(s_I, s_G, G_{map}(0))$ if the robot can reach $s_r^{env} = s_G$ in finite time when a path to goal exists (i.e., $P_{sol}^{env} \neq \varnothing$ for $\forall t$).

The following three theorems analyze the sufficient conditions to guarantee a navigation system is complete.

4.2.1 Known Space Assumption

Theorem 4.1 *For an Effective Robot under Known Space Assumption, the navigation system in a static environment is complete if* $FO(G_{map}(0)) = 0$.

Proof. If there are no false obstacles on the map at initial time $FO(G_{map}(0)) = 0$, and there is a navigation solution in the environment $P_{sol}^{env} \neq \varnothing$, then the map always has solutions $P_{sol}^{map}(t) \supseteq P_{sol}^{env}$ for $\forall t$. Since the environment is static, it does not show false obstacles at $t > 0$. This guarantees completion of the navigation task, because an Effective Robot can search all possible paths in $P_{sol}^{map}(t)$ until it reaches it. ■

The false free spaces on the map produce false paths, and the false obstacles block the paths to the goal. The conventional path-planning algorithms can easily deal with false free

spaces because they directly modify the path as soon as they are found. However, it might not work with the false obstacles. In the worst case, if all paths on the map are blocked, the conventional planning algorithms cannot obtain a path to the goal, even if there is a path in the environment. Later in this chapter, we will propose and study algorithms to try to solve this problem.

4.2.2 Free Space Assumption

Theorem 4.2 *For an Effective Robot under Free Space Assumption, the navigation system in a static environment is complete if* $FO(S_o(0)) = 0$.

Proof. The FSA means if $acc(s_i) = 1$ for $\forall s_i \in S_{UO}(0)$, then there are no false obstacles in $S_{UO}(0)$ because there are no obstacles at all. Taking into account there are no false obstacles in $S_O(0)$, thus $FO(G_{map}(0)) = 0$ on the map at the initial point. Because mapping is effective and the environment is static, no false obstacles appear later (i.e., $FO(G_{map}(t)) = 0$ for $\forall t$). Therefore, $P_{sol}^{map}(t) \supseteq P_{sol}^{env} \neq \varnothing$ for $\forall t$. So an Effective Robot eventually can reach the goal. ■

The Free Space Assumption is usually used when the whole environment is initially unknown [153][150]. In this case, the condition $FO(S_o(0)) = 0$ is satisfied because $S_O(0) = \varnothing$. The navigation task is guaranteed to be complete, if a solution exists. However, in dynamic environments, the FSA might have false obstacles in the observed space $S_O(t)$. The FSA cannot guarantee completion of the navigation task in dynamic environments, but we can use the algorithms presented in **section 4.3** to deal with the false obstacles.

4.2.3 Unknown Space Assumption

We often find the use of Known Space and Free Space Assumptions in robotics literature [147][148][142], but Unknown Space Assumption is less usual. In fact, it is used for map-building tasks [151][152], not for navigation tasks. The aim of a map-building task is for the robot to acquire a map of the whole environment. The common approach is to detect

the frontier states between observed and unobserved regions and to choose a state near to the frontier to expand the observed region. We want to study the above approach, applying it for navigation tasks. Thus we need to find a way to choose a subgoal in the frontier to expand the frontier and make progress toward the main goal.

We consider the following definitions to study the completeness of a navigation system with Unknown Space Assumption.

Frontier. A state s_f is in frontier set $S_F(t)$ if and only if $acc(s_f) = 1$ and $\exists s_i \in (S_{neighbors}(s_f) \cap S_{UO}(t))$, where $S_{neighbors}(s_f)$ is the set of all immediate neighbor states of s_f. The first condition guarantees the state s_f can be reached by a robot, and the second condition ensures that the state s_f has at least one neighbor in the unobserved region.

Goal planner for Unknown Space Assumption. When $s_G \in S_{UO}(t)$ for some $t > t_0$, the goal planner chooses a state $s_g(t) \in S_F(t)$ as a partial goal every time that the previous selected $s_g(t_0)$ is no longer in $S_F(t)$. And when $s_G \in S_O(t)$ for some t, the goal planner always chooses s_G to complete the navigation task.

Theorem 4.3 *For an Effective Robot under Unknown Space Assumption, the navigation system in a static environment is complete if* $FO(S_O(0)) = 0$.

Proof. Because there is no assumption on obstacles in the unobserved region, $FO(S_{UO}(0)) = 0$. Taking into account no false obstacles in $S_O(0)$, $FO(G_{map}(0)) = 0$ on the map at initial time. Because mapping is effective and the environment is static, $FO(G_{map}(t)) = 0$ for $\forall t$. Besides, an Effective Robot with a goal planner for Unknown Space Assumption guarantees the expansion of observed space at every new partial goal s_g; therefore, $\exists t > 0$, $P_{sol}^{map}(t) \cap P_{sol}^{env} \neq \varnothing$. So an Effective Robot can eventually reach the goal s_G. ■

If we use a goal planner for Unknown Space Assumption when the whole environment is initially unknown, the condition $FO(S_o(0)) = 0$ is trivially satisfied because $S_o(0) = \varnothing$. The navigation task is guaranteed to be complete, if a solution exists. If the sufficient condition is not satisfied $FO(S_O(0)) \neq 0$ or the environment is dynamic, false obstacles can be created over $S_O(t)$, we can apply any of the strategies to deal with false obstacles, which we show in the next section.

The strong conclusion in this section is the relevance of false obstacles. They can determine whether a navigation task can be complete or not. Besides, we introduce how to use Unknown Space Assumption for navigation tasks, which will be used in **section 4.4**.

4.3 Navigation in dynamic environments

In the previous section, we saw that if there are no false obstacles, we can ensure that the navigation system can find a path to reach the goal for static environments. However, false obstacles can be generated in dynamic environments. So it is necessary to invent methods to ensure that a navigation system can find the path to its goal, even if there are false obstacles. In this section, **we develop a couple of algorithms to recover from map errors**. To the best of our knowledge, these algorithms are the first reported in literature.

When a conventional navigation systems gets stuck because the planning system does not find a path to the goal, there are at least two basic methods to solve this problem:

1. Method of deleting the unobserved region. Create a new map with the observed region $S_O(t = t_1)$ and assume the unobserved region is free space (see Figure 4.6). Then the planning system can generate paths to reach the goal. Many of these paths will be erroneous due to false free spaces, but eventually the navigation system could find a path to the goal if it exists.

2. Method of verification of barrier. Find the barrier of obstacles enclosing the goal in the unobserved region $S_{UO}(t = t_1)$ on the map and verify that the barrier actually exists in the environment (see Figure 4.7). If any part of the barrier is false, the robot could find a path reaching the goal. If it turns out that the barrier is real, then the robot will know that there is no route to the goal.

The first method is easy to implement, and needs to track the recently observed region. Since the previous map information in S_{UO} is ignored, it only conserves the recently observed region $S_O(t_1)$. The second method saves the whole previous map until the verification process

deletes the false obstacles. It needs more computation time to detect the barrier and verify obstacles. It can be useful when the errors on the map are small. The implementation is more complex.

Any of the two methods makes the navigation system recover from map errors. The first method deletes false obstacles and uses Free Space Assumption. The second method verifies the barrier on the map exhaustively. If a false obstacle exists, it will be found and deleted, and a path to the goal might be formed. Both methods can work with any path-planning algorithm.

4.3.1 Algorithms

Algorithm 4.1 *Main()*

$s_r := s_I$; $s_g := s_G$; $S_O := \varnothing$
while $s_r \neq s_G$
  $path :=$ *ComputeShortestPath*(G^{map},s_r,s_g)
  if $path = \varnothing$
    $s_p :=$*ComputeGoalToVerifyBarrier()*
    if $s_p \neq \varnothing$, $s_g := s_p$, *else, return* $false$
  else
    while PathIsFree()
      $s_r :=$*planexecution(path)*
      $S_O, G^{map} :=$*mapupdate()*
      if $s_g = s_G$
        if $s_r = s_G$
          break
      else
        if BarrierIsOpen()
          $s_g := s_G$
          break

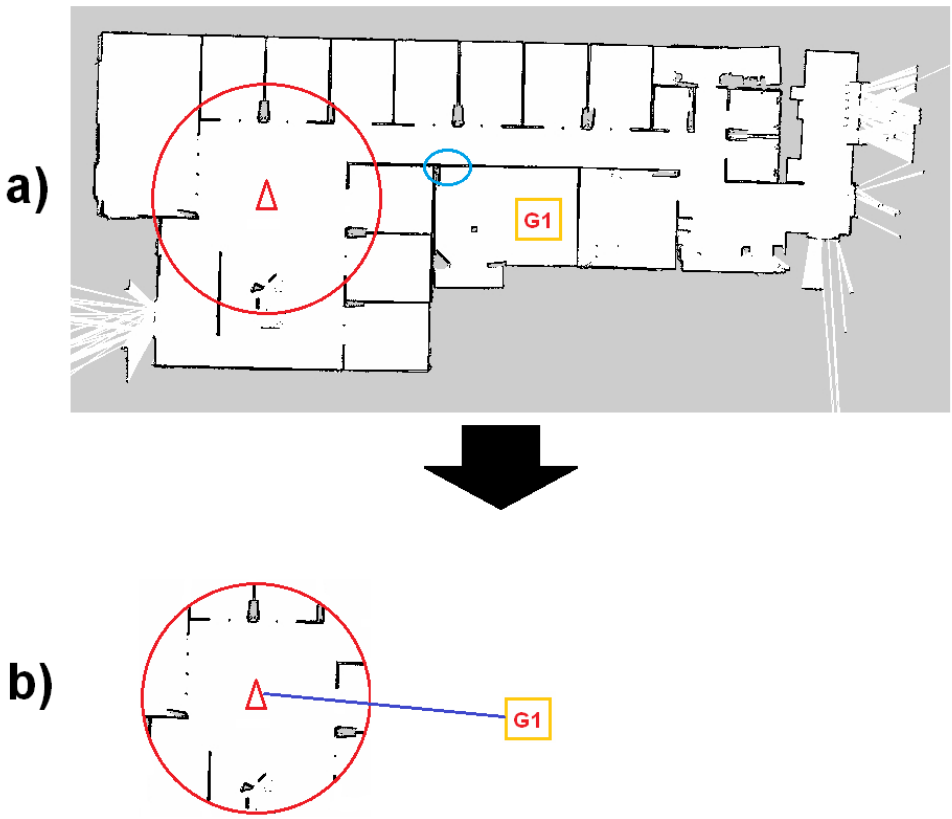


Figure 4.6: (a) If the navigation system cannot find a way to reach the goal, then (b) it will delete all obstacles in the unobserved region to prevent false obstacles. This figure illustrates this method.

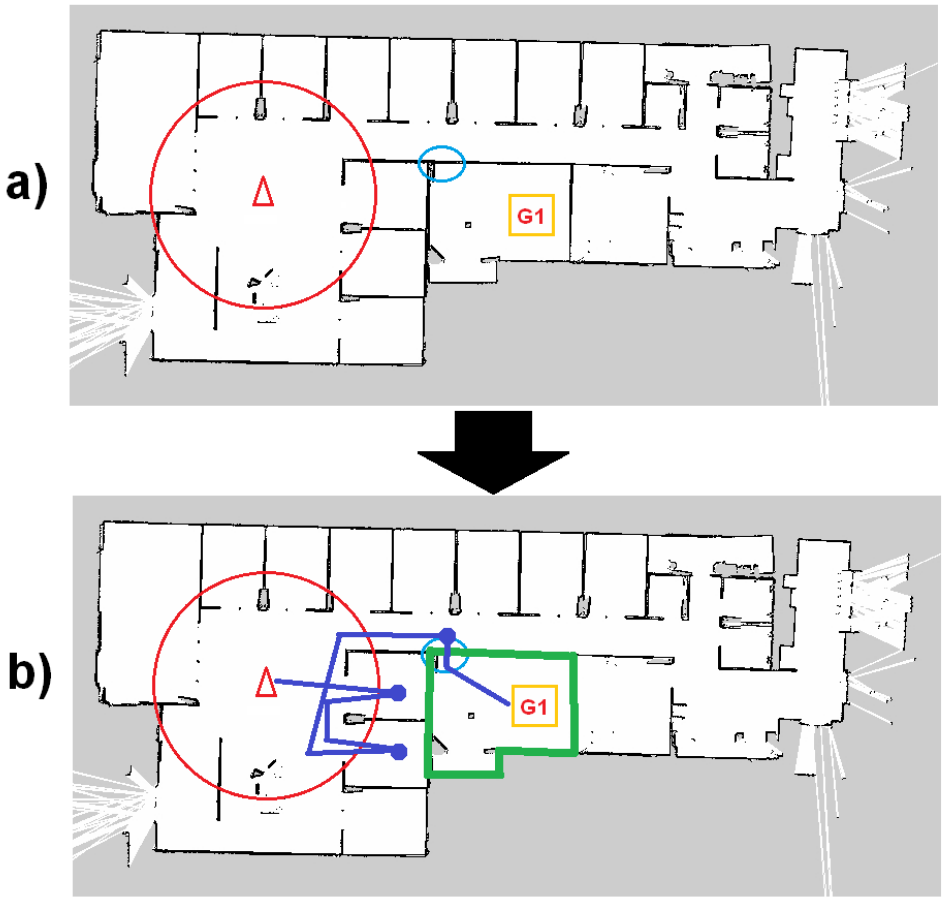


Figure 4.7: (a) If the navigation system cannot find a way to reach the goal, then (b) it will find the barrier of obstacles (green line) enclosing the goal on the map, and it will check if the barrier actually exists. This figure illustrates this method.

elseif $s_r = s_g$
  s_p :=*ComputeGoalToVerifyBarrier()*
  if $s_p \neq \varnothing$, $s_g := s_p$, *break, else, return* $false$

We only present the algorithm for the second method because the first is too simple. Algorithm 4.1 describes the main loop to complete the navigation task and to cope with false obstacles. When the planner (i.e., "ComputeShortestPath"), cannot find a path to the goal s_G, the function "ComputeGoalToVerifyBarrier" computes the unobserved barrier and determines the best state in the barrier s_p to be checked.

We use the planner to find the path to s_p. If the barrier is open during path execution, the algorithm comes back to the main goal s_G. If the state s_p is reached and the barrier has not been opened, it recomputes the barrier and chooses another state in the barrier to be checked out. This process is repeated until eventually the barrier is open or there is no unobserved barrier.

Algorithm 4.2 *ComputeGoalToVerifyBarrier()*

for all $s \in S^{map}$
  $fclose(s) := 0$
  $fobstacles(s) = 0$
  $parent(s) := None$
  $g(s) := \infty$
$open := \varnothing$
enqueue $[0, s_r]$ *onto open*
while $open \neq \varnothing$
  g_{old}, $s \longleftarrow open.dequeue()$
  $fclose(s) := 1$
  for all $s' \in neighbors(s)$
    if $fclose(s') = 0$
      $g' := g_{old} + c(s, s')$
      if $acc(s') = 0$ *and* $s' \in S_{UO}$

```
            if  g' < g(s')
                fobstacles(s') := 1
                parent(s') := s
                g(s') := g'
        elseif acc(s') = 1
            if  g' < g(s')
                enqueue [g', s'] onto open
                g(s') := g'
for all s ∈ S^map
    fopen(s) := 0
open := ∅
barrier := ∅
enqueue [s_G] onto open
fopen(s_G) := 1
while open ≠ ∅
  s ⟵ open.dequeue()
  fclose(s) := 1
  for all s' ∈ neighbors(s)
     if fopen(s') = 0 and s' ∈ S_UO
        if fobstacles(s') = 1
            enqueue [g(s') + c(s', s_G), s'] onto barrier
        else
            enqueue [s'] onto open
        fopen(s') = 1

if barrier ≠ ∅
  s_b = arg min_{s∈barrier}{g(s) + c(s, s_G)}
  s_p = parent(s_b)
else
```

$s_p = \varnothing$
return s_p

Algorithm 4.2 explains how to compute the unobserved barrier and how to choose a state to be checked. It detects the unobserved barrier using two wave expansions based on breadth-first search (BFS). The first expansion detects all the obstacles in the unobserved region around the robot´s position. The array $fobstacles(s)$ stores all these obstacles. The second expansion initializes from s_G and ends when it finds the zone reached by the first expansion. This last expansion takes into account only the obstacles detected by the first expansion. Finally it stores the unobserved barrier states in a list called *barrier*. This process is illustrated in Figure 4.8.

Finally, the algorithm chooses a state in the barrier minimizing $f(s) = g(s) + c(s, s_G)$, which is equal to the function used in the A* search. The first term corresponds to the actual distance of $s_r \rightarrow s_{barrier}$ and the latter term is a heuristic distance of $s_{barrier} \rightarrow s_G$, ignoring the obstacles on the map. We try to select the state to be observed, minimizing the distance the robot must move during barrier verification and finding the closest state to the goal. Figure 4.8 shows an example of this minimization.

4.3.2 Experiments

Navigation systems with the proposed methods can recover from errors on the map. In this section, we wonder which of the two methods is closer to optimality. We want to know which one generates shorter paths. To answer this question, we make a series of Monte Carlo simulations and we introduce some new concepts: the suboptimality of the path length, the obstacle density, the success index, and the dissimilarity of the map.

We define the suboptimality of the path length l_{sub} as

$$l_{sub} = \frac{\text{length of the effective path}}{\text{length of the optimal path}} \tag{4.1}$$

where effective path is the path that the robot has followed to reach the goal and the optimal path is the shortest path to reach the goal. $l_{sub} \geq 1$ means that the effective path is longer

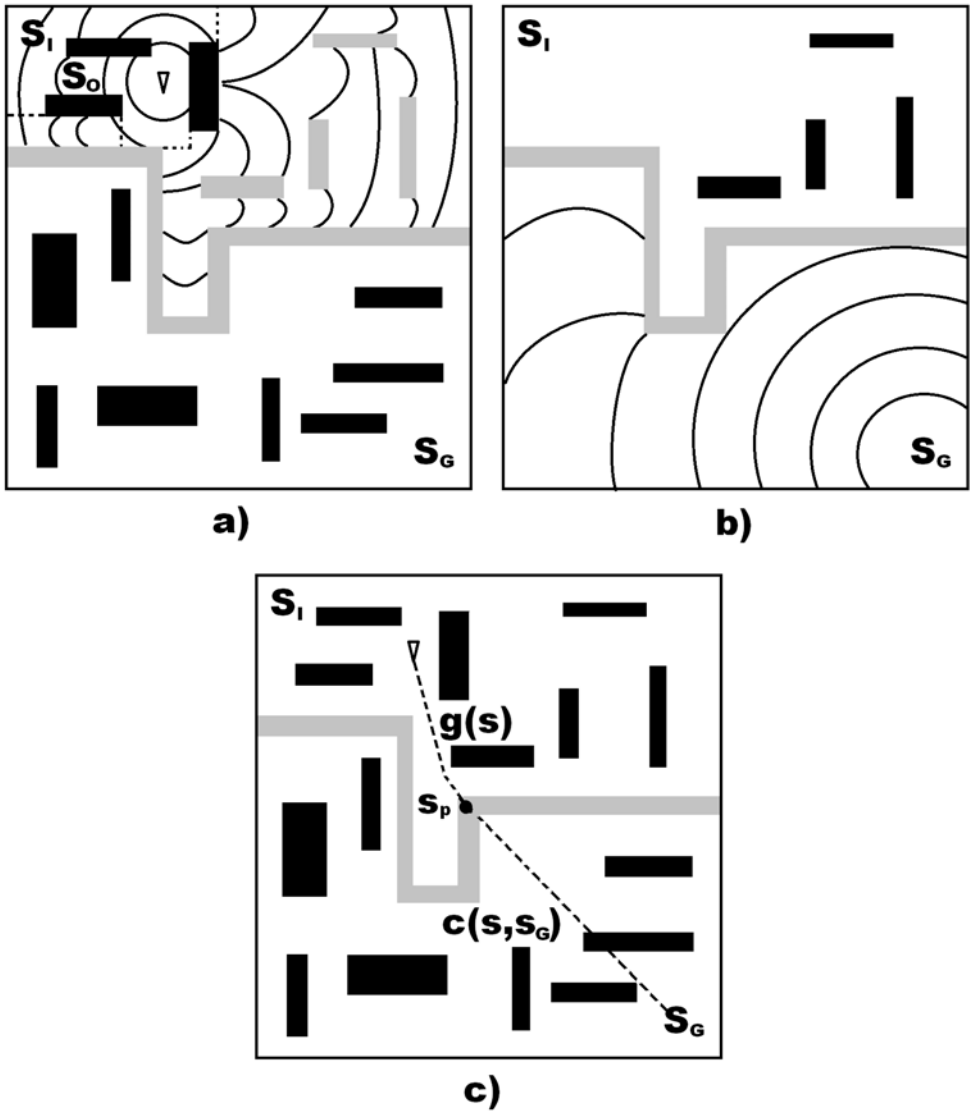


Figure 4.8: This illustrates the method of verification of the barrier. In (a) it finds only the obstacles outside the observed region S_O via a BFS expansion; such obstacles are grey. The S_O is delimited by dashes. In (b) the second expansion from s_G finds the barrier (grey). In (c) it chooses the state in the barrier that minimizes $f(s) = g(s) + c(s, s_G)$. These distances are marked by dashes.

than the optimal path (see an example in Figure 4.9). The average of l_{sub} in a series of Monte Carlo simulations, defined by $E[l_{sub}]$, represents the effectiveness of the navigation system.

We define the obstacle density d_{obs} as

$$d_{obs} = \frac{\text{Obstacles Area}}{\text{Total Area}} = \frac{\sum_{s \in S_{env}} acc(s)}{|S_{env}|} \tag{4.2}$$

If $d_{obs} = 0$, the environment is empty; if $d_{obs} = 1$, the whole environment is occupied.

We define the success index s_{ind} as

$$s_{ind} = \frac{\text{Number of successful navigations}}{\text{Total number of navigations}} \tag{4.3}$$

It measures the probability of reaching the goal when the map has false obstacles.

We introduce the dissimilarity of map d_s as

$$d_s = \frac{\text{False Obstacles Area + False Free Spaces Area}}{\text{Total Area}} = \frac{FF(G_{map}) + FO(G_{map})}{|S_{env}|} \tag{4.4}$$

It measures how many errors the map has. If $d_s = 0$, the map matches the environment perfectly. If $d_s = 1$, the map has a perfect interchange between obstacles and free spaces.

The proposed algorithms are evaluated through Monte Carlo simulations. We use a two-dimensional grid space with 100×100 states as environments and maps. The initial point of the robot s_I and the goal state s_G are set near the contra corners, as we illustrate in Figure 4.9.

Navigation under Free Space Assumption in unknown environments

We first study the suboptimality l_{sub} for initial unknown environments using Free Space Assumption. We randomly generate 500 environments with uniform distribution. The range of obstacle density (4.2) is $[0.1, 0.6]$, and we fix this maximum because when $d_{obs} > 0.6$, no path exists between s_I and s_G. The optimal path is obtained by a normal A* search, assuming the map is known and correct. The effective path is produced by the same A* search algorithm, but assuming the environment is initially unknown and the whole initial map is set as free space.

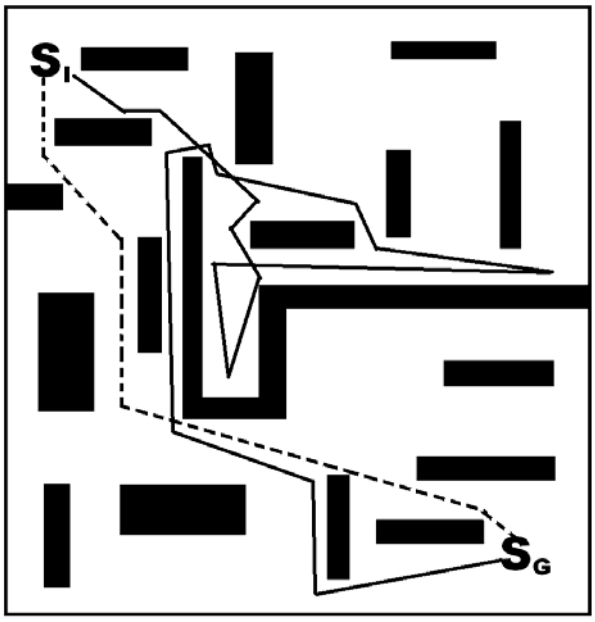


Figure 4.9: This shows an example of the difference between the effective path (solid line) and the optimal path (dash line). The first is produced because initially the environment is unknown. The second is determined when the environment is known completely and correctly.

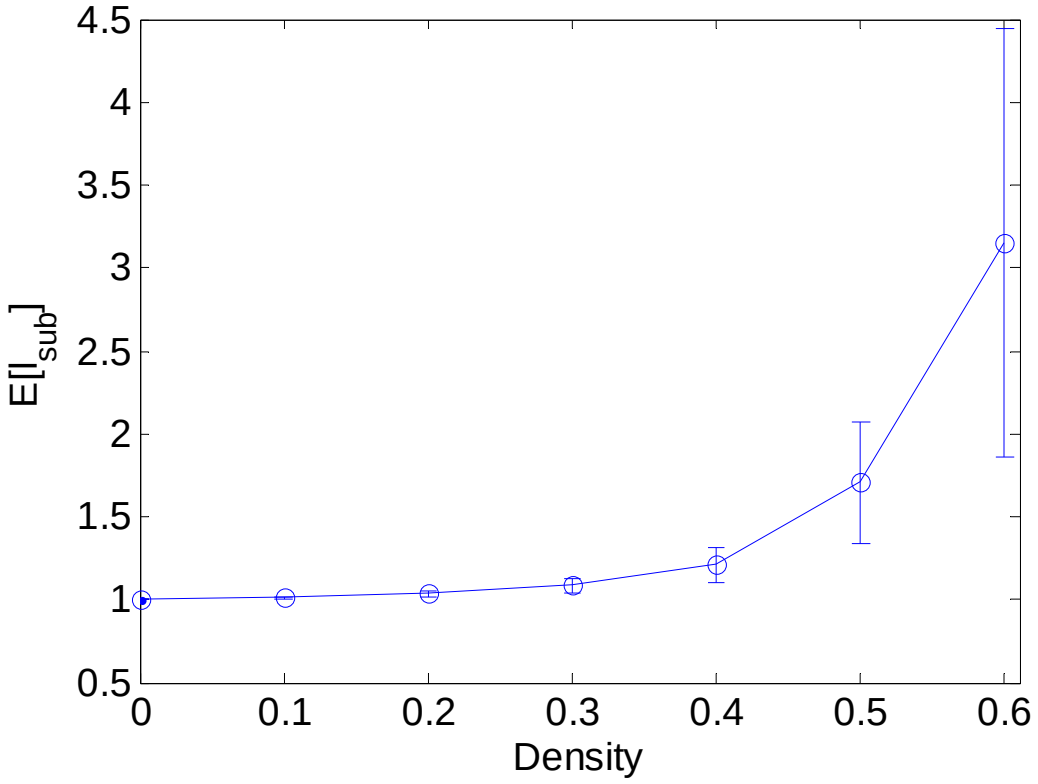


Figure 4.10: The average of the path length $E[l_{sub}]$ together with an interval of one standard deviation as a function of obstacles density, when initially the environment is unknown.

The average of the path length $E[l_{sub}]$ with respect to the obstacle density is shown in Figure 4.10. Since the robot does not have map at the initial point, $E[l_{sub}]$ increases when obstacle density increases; this behavior is also reported in [150]. This result will be used in the next subsections.

Navigation under Known Space Assumption with false obstacles

We study the success index s_{ind} and the suboptimality l_{sub} for Known Space Assumption in the function of the map errors (i.e., the dissimilarity of map (4.3) first with no method to

deal with false obstacles and later comparing the two methods describe in **section 4.3)**.

We also generate 500 random environments with uniform distribution for each obstacle density in the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$. For each density and dissimilarity value, we copy the 500 environments and use them as maps, but the maps contain a certain number of errors. These errors are also uniformly random according to the value of dissimilarity $[0, 0.6]$. Again if $d_s > 0.6$, no path exists between s_I and s_G. The planner is a conventional A* search that replans if the path is blocked.

Figure 4.11 shows the success index s_{ind} (4.3) vs. the dissimilarity d_s (4.4) with respect to obstacle density d_{obs} (4.2). It is reasonable that the probability of reaching the goal is reduced as the map dissimilarity increases. When the obstacle density is large, the robot may not find a path to the goal due to errors. Indeed, if more than 60% of states on the map are wrong, it is impossible to reach the goal for any obstacle density.

We compare Known Space and Free Space Assumptions in terms of suboptimality. Figure 4.12 shows the average of the path length $E[l_{sub}]$ (4.1) and the dissimilarity of map d_s (4.4) with respect to different obstacle densities d_{obs} (4.2). For environments with low density (≤ 0.3), the profit of knowing the environment with lower dissimilarity is not much (≤ 0.25), then the use of a map is a waste of time. Therefore, if the environment were dynamic, with a constant density ≤ 0.3, it is better to use the FSA. For environments with higher density (> 0.3), having a map with a 0.15 dissimilarity or less is a real advantage. But if the map has more than $\sim 15\%$ errors, it is better to use the FSA. **In this case, a wrong belief (corrupted map) is worse than ignorance (the FSA).**

Now we turn our attention to the two methods of **section 4.3** to deal with false obstacles. Remember that we want to know which one generates shorter paths. We use the random environments and maps of the previous experiment only for high density $\{0.4, 0.5, 0.6\}$. The low density can be ignored because we know that the FSA is better than the KSA. Note that $s_{ind} = 1$ for both methods at any density and dissimilarity of static environments. So we only focus on studying the path length $E[l_{sub}]$.

Figure 4.13 shows the results at different scales: the first row shows them for the whole range of dissimilarity, and the second row, at low dissimilarity only. The constant values

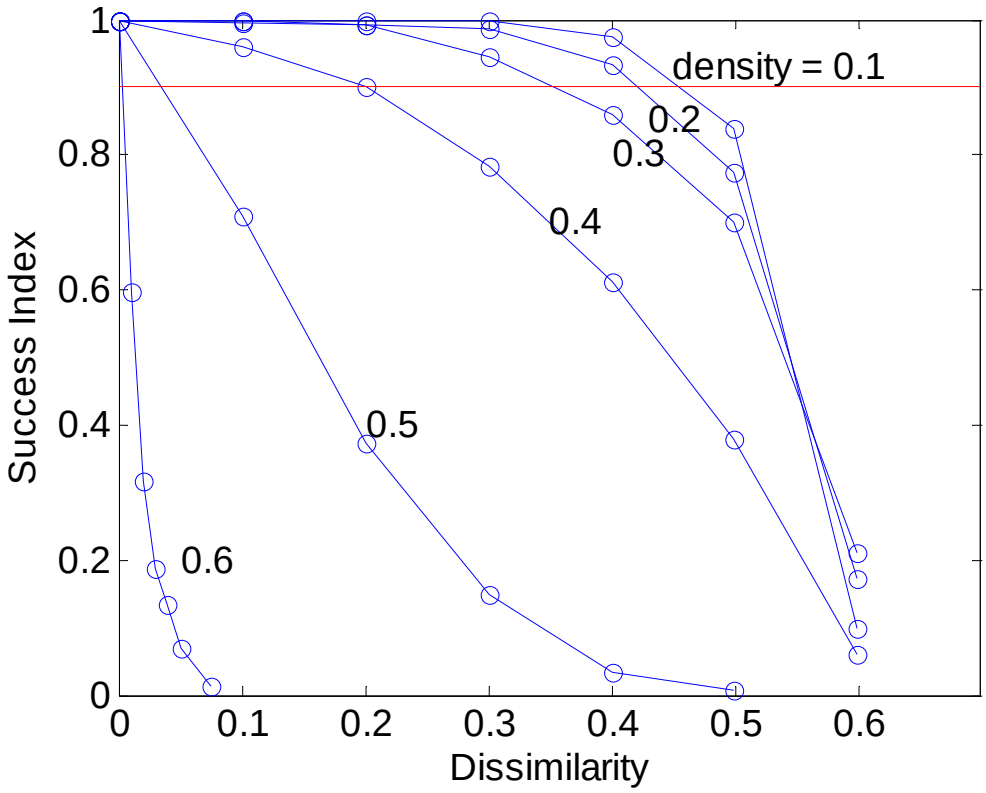


Figure 4.11: This figure shows the probability of achieving the goal (success index) when the map has errors (dissimilarity of map), considering different densities of obstacles and knowing a map initially (KSA).

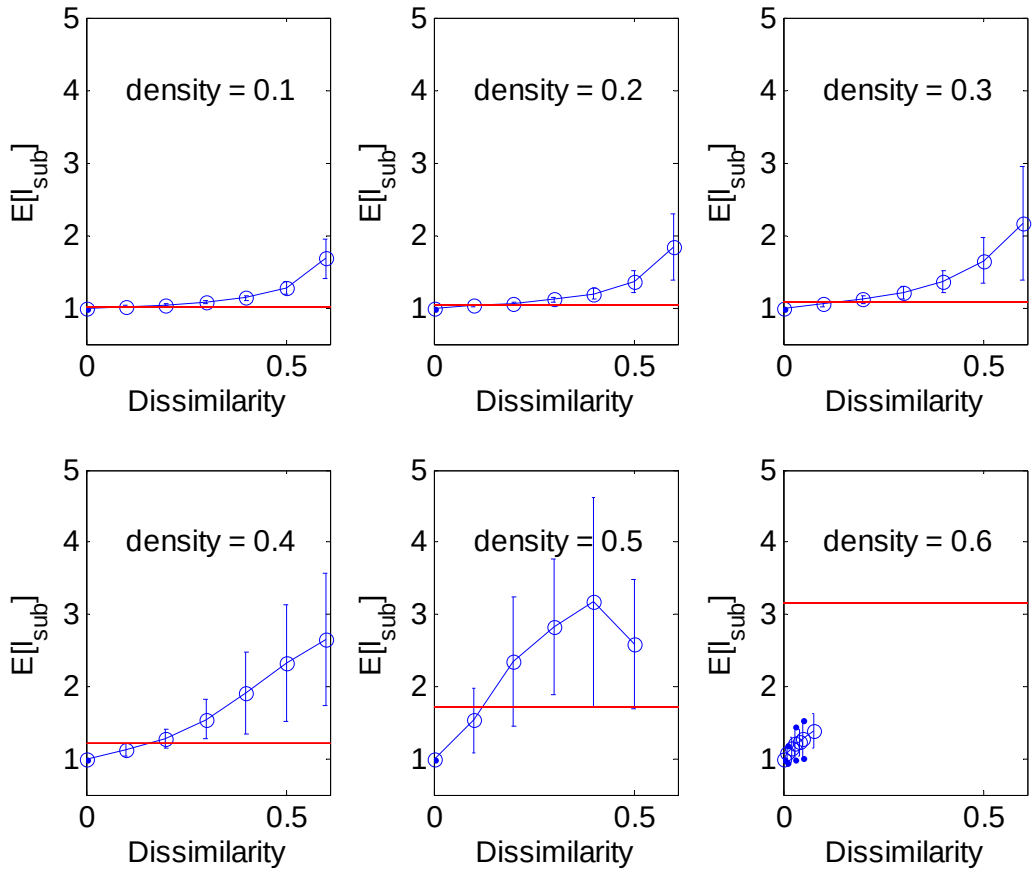


Figure 4.12: This figure compares KSA (blue line) and FSA (red line) in terms of suboptimality, considering different densities of obstacles.

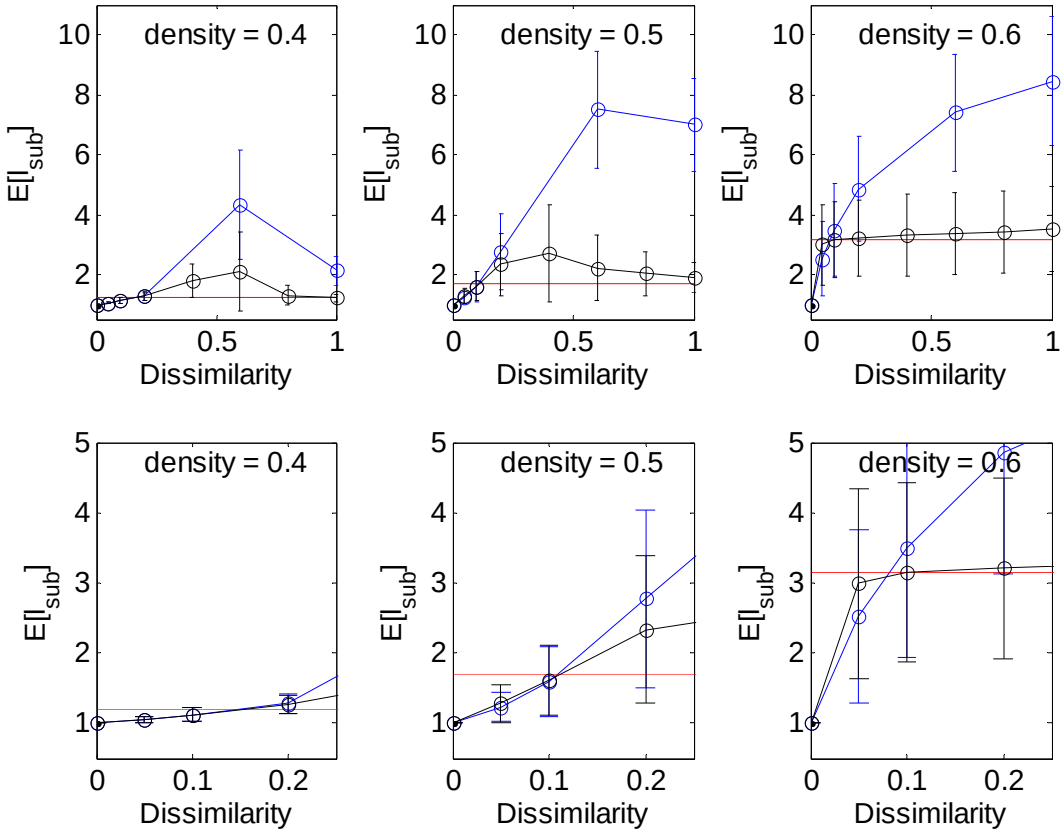


Figure 4.13: When the map has few errors (dissimilarity $< 10\%$), is better to verify the barrier (blue lines); and when the map has many errors (dissimilarity $> 10\%$), it is better to delete the unobserved region of the map (black lines). The red lines are obtained by using FSA from the beginning, which has the best suboptimality at high dissimilarity ($> 10\%$).

represented by the red lines are obtained by using the FSA from the beginning, the black lines by using the method of deleting the unobserved region, and the blue lines by using the method of verification of barrier.

We can see that the method of deleting the unobserved region (black lines) is better when there is high dissimilarity ($> 10\%$), while the method of verification of barrier (blue lines) is better when there is low dissimilarity. The latter is more obvious when the obstacle density is bigger than 0.6. In other words, when the map has few errors (dissimilarity $< 10\%$), it is better to verify the barrier, and when the map has many errors (dissimilarity $> 10\%$), it is better to delete the unobserved region of the map. These results suggest that if the robot keeps track of an estimation of dissimilarity of the map, the robot could decide which method to use and could benefit from both methods.

4.4 Navigation in unknown environments

We studied the effect of partial observability in dynamic environments; now we will study what happens in unknown environments. Free Space Assumption is the common approach to navigate in unknown parts of an environment. For example, in an initially unknown environment, the robot makes a plan assuming the space is free; it tries to execute the plan, but it finds an obstacle and needs to replan a new path. Replanning is done every time the robot cannot execute the plan, until it reaches the goal. Navigation with the FSA is illustrated in Figure 4.14-(a).

As we see in **section 4.2**, there is another way to solve this problem, using the Unknown Space Assumption instead. In an initially unknown environment, the robot observes the near space, detects the frontier S_F (between S_{UO} and S_O), chooses a promising state s_g in the frontier, and then plans and executes a path to s_g. As consequence, the robot expands the observed region, making progress to the main goal s_G. The whole process is repeated until eventually s_G is in the observed region and it can be reached. Figure 4.14-(b) illustrates this navigation process using the USA.

We could prefer this second approach because the FSA requires planning even within

regions where the robot has no information about the environment and there is a high probability that the path should be changed. The USA only plans within the observed region, and this path is highly probably correct and short. In principle, we could save computational resources.

4.4.1 Algorithms

Algorithm 4.3 *Main()*

$s_r := s_I$; $s_g := s_G$; $S_O := \varnothing$; $S_F := \varnothing$;
while $s_r \neq s_G$
  S_O, G^{map} :=*mapupdate()*
  $S_F :=$ *frontierupdate()*
  if $s_g \notin S_F$
    s_g :=*goalplanner()*
    if $s_g = \varnothing$, *return* $false$
  if IsCloseToGoal(s_r)
    s_r :=*reactiveexecution*(s_g)
    continue
  else
    path := *ComputeShortestPath*(G^{map},s_r,s_g)
    if path $= \varnothing$, *return* $false$
  s_r :=*planexecution(path)*
return true

Algorithm 4.3 implements the navigation using the USA. The initial goal is s_G. When s_G is in the unobserved region, the goal planner chooses a partial goal s_g in the frontier S_F. If s_g disappears from the frontier during the execution, the algorithm chooses another partial goal in the current frontier. One good property of Algorithm 4.3 is that it can avoid planning when the goal is close to the robot. In this case, no plan is made, reducing computational processing and the robot simply moves toward the partial goal.

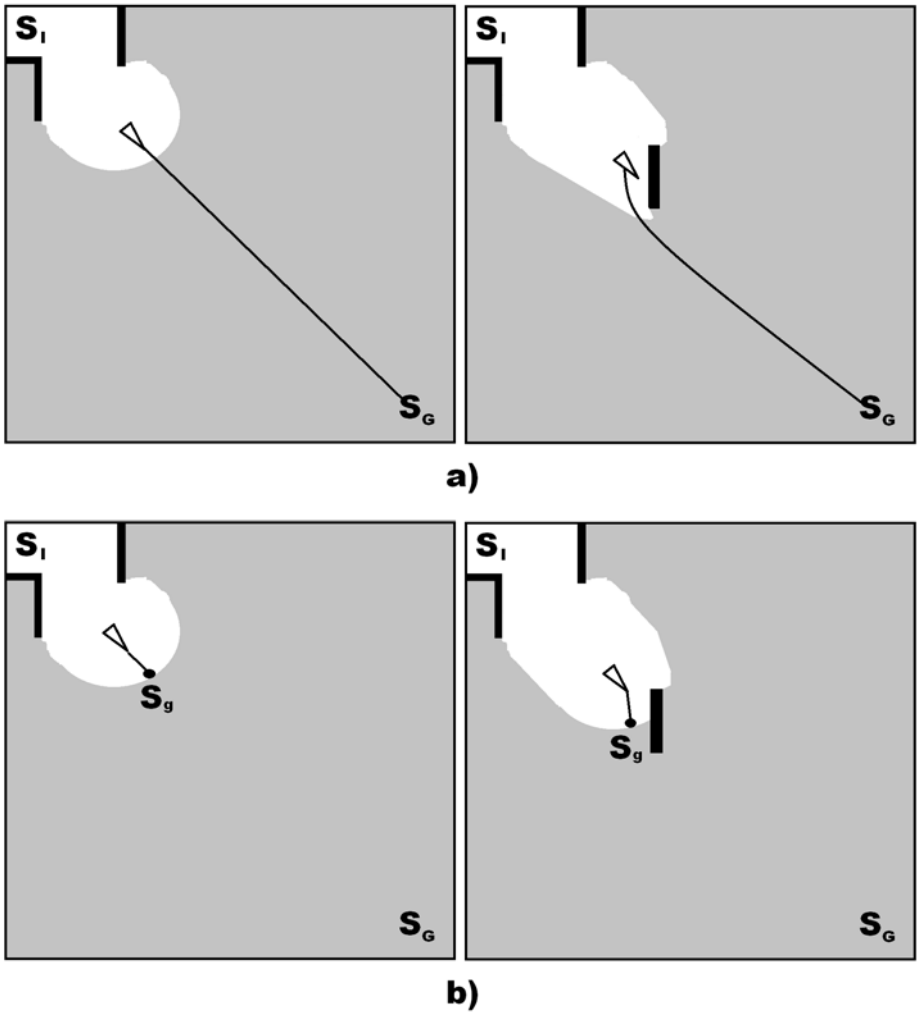
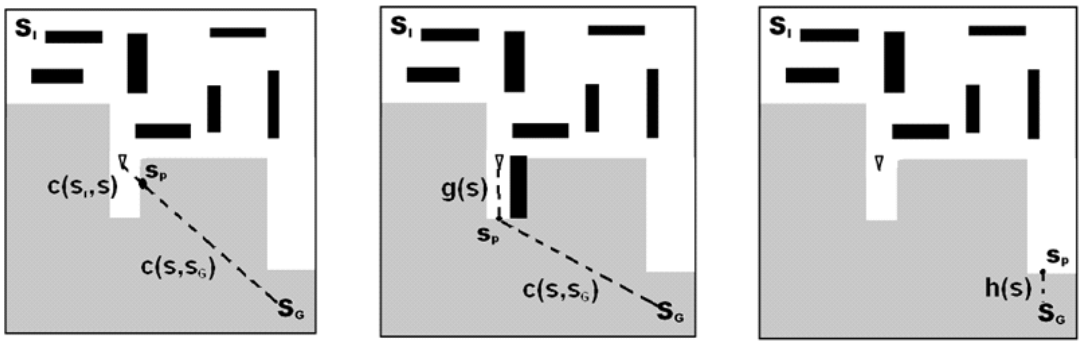


Figure 4.14: This illustrates the navigation in unknown environments using (a) FSA and (b) USA. The former generates paths across the whole map, even in the unobserved region, while the latter only makes plans within the observed region. This could be a computational advantage.



$f_1(s) = c(s_r, s) + c(s, s_G)$ $f_2(s) = g(s) + c(s, s_G)$ $f_3(s) = h(s)$

Figure 4.15: The figures show the different functions to be minimized to select a partial goal at the frontier.

Algorithm 4.4 *Goal planner*

if $s_G \in S_O$
  return s_G
else
  if $S_F \neq \varnothing$
    return $\arg\min_{s \in S_F}\{f(s)\}$
  else
    return $\varnothing$

However, the big question is how to select a good state in the frontier to make the major progress to the goal. Algorithm 4.4 shows the general way to choose a goal in the frontier. If the main goal s_G is not observed, the algorithm chooses a goal in the frontier to visit and expand the observed region. Otherwise, the goal planner would choose the main goal.

The goal planners presented choose the goal by minimizing a cost function $f(s)$. There are at least three candidates for cost function using the geometrical information from the map (see Figure 4.15).

1. $f_1(s) = c(s_r, s) + c(s, s_G)$. It measures the Euclidean distance $c()$ from the robot's position to the frontier state, and from the frontier state to the main goal. Note that real distance should consider the obstacles configuration, so $c()$ is just an optimistic approximation of the distance.

2. $f_2(s) = g(s) + c(s, s_G)$. The first term is exchanged with the real distance from the robot's position to the frontier state, considering the obstacles in the observed region. This cost function is similar to the A* search. $g(s)$ is computed by a forward search based on BFS to the frontier. This represents more computational processing.

3. $f_3(s) = h(s)$. The goal planner selects the frontier state closest to the main goal. $h(s)$ is obtained by a backward A* search from the main goal to the observed region. The search is stopped when the closest frontier state is found.

4.4.2 Experiments

The classical approach to facing unknown environments is using Free Space Assumption. We compare the method we proposed using the three cost functions with this classical approach. We want to know which one generates the shortest paths and consumes less time to plan. To answer this question, we make some Monte Carlo simulations. The environments for these experiments are the same as in **section 4.3**. Initially the map is set as an unknown space for our method. We use a conventional A* search as the planner *ComputeShortestPath().* For cost functions $f_1(s)$ and $f_3(s)$, the frontier detection method *frontierupdate()* continuously computes the frontier in the same philosophy of Fast Frontier Detector [155]. In contrast, for cost function $f_2(s)$, the algorithm detects the frontier states from scratch, similar to what the Wave Frontier Detector [155] does (this is implemented without using *frontierupdate()*).

Figure 4.16 and 4.17 show the suboptimality $E[l_{sub}]$ and the average time to complete the random navigation tasks in a function of density d_{obs}.

The FSA presents the best l_{sub} at high densities (> 0.4). Below this value, any method produces the same suboptimality l_{sub}. However, we must note that the function $f_1(s)$ has

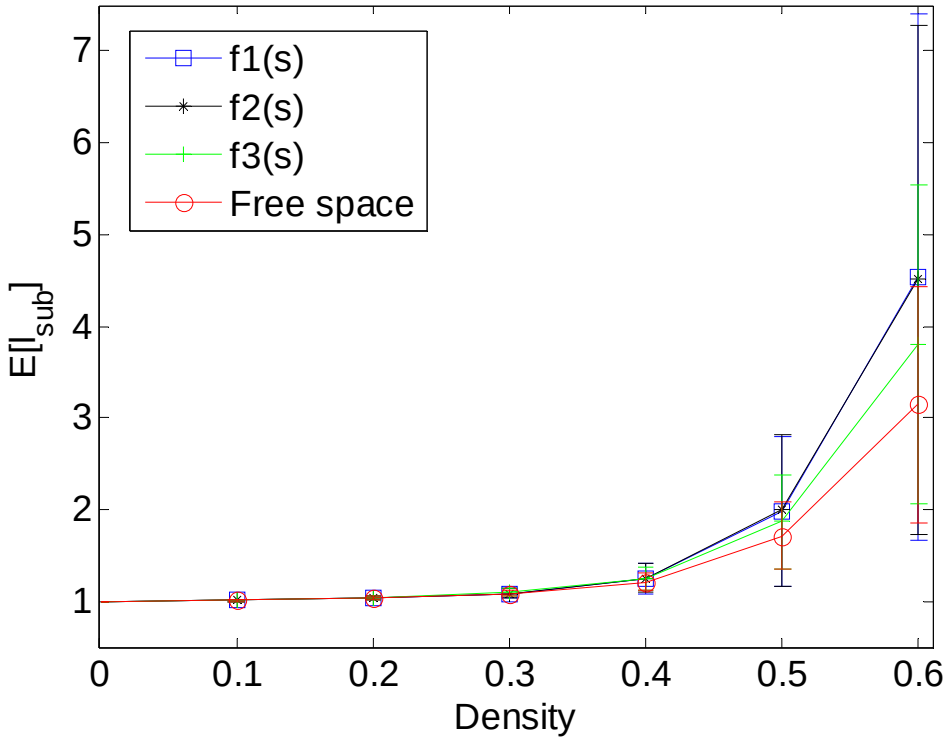


Figure 4.16: Path length $E[l_{sub}]$ to complete navigation tasks in unknown environments using FSA and USA with different cost functions.

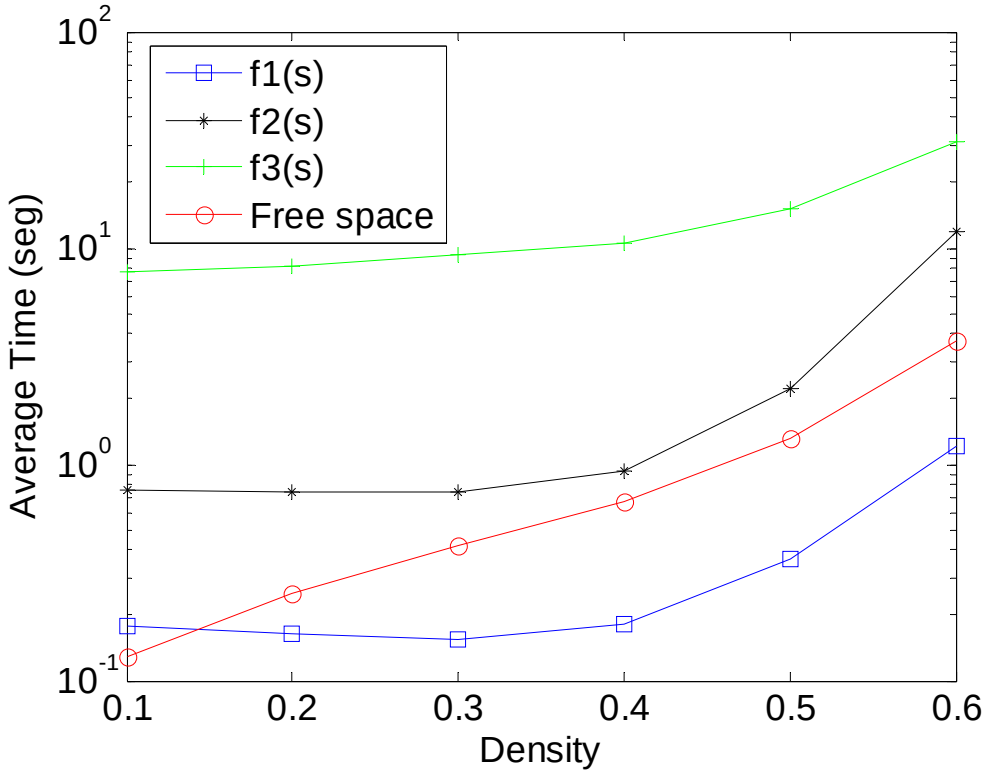


Figure 4.17: Average computation time to complete navigation tasks in unknown environments using FSA and USA with different cost functions.

the best computational efficiency (see Figure 4.17), even better than the FSA (at densities > 0.15). This is explained by two reasons:

1. Our method plans only in the observed region, instead of planning in the entire map.
2. Our method avoids planning when the partial goal s_g is too close to the robot and this could happen frequently.

Based on these results, we can recommend using the USA with the function $f_1(s)$ at low densities (< 0.4) and use the FSA at high densities (> 0.4). For real applications, this implies that we need a way to estimate the obstacle density, which is another problem outside the scope of this thesis.

4.5 Conclusion

Three assumptions are analyzed to deal with the problem of partial observability of the environment. We give the conditions sufficient to ensure the navigation is complete under Known, Free, and Unknown Space Assumptions for static environments. This analysis shows the importance of dealing with false obstacles in order to complete the navigation task.

Although it is impossible to guarantee that a navigation system is complete in dynamic environments, it is possible to create methods for recovering from false obstacles. We propose and evaluate two methods to recover from false obstacles on the map. Both must be used depending on the magnitude of the errors on the map. The method of deleting the unobserved region is better when there is high dissimilarity between the map and the environment ($>$ 10%), while the method of verification of the barrier is better when there is low dissimilarity ($< 10\%$). We think this work opens the research for methods that give robots the ability to recover from errors on the map. More work should be made in this direction because SLAM methods are imperfect and the environments are dynamic.

On the other hand, we also study the effect of partial observability on unknown environments. We propose a new method using Unknown Space Assumption to navigate more

efficiently in unknown environments. We compare with the classical approach using Free Space Assumption. Our method is more efficient, but can generate longer paths. We recommend using our method with the function $f_1(s)$ at low obstacle density (< 0.4) and when planning time is important, and using the classical method of the FSA at high obstacle density (> 0.4) and when optimality of path length is more important.

Chapter 5

Conclusion

"Don't learn everything... if you want to reach your goal, just learn what is needed." —Erik Zamora

5.1 Conclusions and outlook

Here we summarize the results and the conclusions of this thesis. Autonomous navigation is a complex problem which is divided into sub-problems such as perception, map building, robot state estimation, planning and control. In this thesis, we focus on developing some algorithms related to map building, and planning, obtaining the following results:

For **map-building:**

- Unlike the other methods based on sets, the Ellipsoidal SLAM method solves large-scale problems. Furthermore, it has similar performance to EKF-SLAM, and in some simulated cases, it can be better. Errors are modeled by ellipsoid sets without assuming Gaussianity, so we have more realistic error modeling. Other results are: (1) the convergence of the ellipsoid algorithm is proven, (2) we also propose a data association method similar to the individual compatibility nearest neighbor test in EKF-SLAM, (3) simulations showed that the Ellipsoidal SLAM could be more robust to motion model errors with bias, and (4) the experimental results show it is capable of building

maps both indoors and outdoors. However, there are technical problems remain, such as how to calibrate the Ellipsoidal SLAM parameters and how to improve accuracy.

For **planning:**

- We introduce and study the fundamental concepts of Known Space, Free Space and Unknown Space Assumptions. These assumptions are needed to navigate in dynamic and unknown environments. We give the conditions sufficient to ensure a navigation system is complete for static environments, showing the importance of dealing with false obstacles to complete the navigation task.

- **For dynamic environments**, we propose two methods to deal with false obstacles on the map. Both must be used depending on the magnitude of map errors, i.e. the dissimilarity of the map with respect to the environment. The method of deleting the unobserved region is better when there is high dissimilarity ($> 10\%$), while the method of verification of barrier is better when there is low dissimilarity ($< 10\%$). We think this work opens up research for methods that give robots the ability to correct errors on the map. More work is needed in this direction, because SLAM methods are imperfect and the environments are dynamic.

- **For unknown environments**, we proposed a method to navigate more efficiently, showing that this method can be faster than the classical approach using Free Space Assumption. We recommend using our method with the function $f_1(s)$ at low obstacle density (< 0.4) and when planning time is important, and using the classical method of the FSA at high obstacle density (> 0.4) and when optimality of path length is more important.

Finally, the following open problems should be solved in the future:

- The main hurdle for improving the accuracy of a SLAM method is to have reliable landmark detectors. Based on our experience, we can tell that if we can create a 100% reliable landmark detector, we could solve the SLAM problem almost without error.

We think that research efforts should be conducted toward this direction instead of toward finding better estimation techniques.

- We need to search for methods that give robots the ability to recover from errors on the map because SLAM methods are imperfect and environments are dynamic. One way is to ensure that robots recognize mobile objects, like doors and furniture in order to ignore them in the mapping and planning processes to avoid errors.

5.2 Final Message

Ladies and gentlemen, the future is bright. Robotics is to the 21st century and is going to catalyze the development of our civilization. Enough technology and effective methods exist to implement autonomous robots today. It is strategic to use these resources to generate wealth for the benefit of Mexican society. If we add forces from entrepreneurs, investors, scientists and government, **robotics will transform the country to improve our quality of life**.

5.3 About the author

Erik Zamora is Titular A Professor in UPIITA-IPN. In 2007-2008, I developed the first commercial Mexican myoelectric system to control a prosthesis with three degrees of freedom, which is produced by the company PRO/BIONICS. Between 2008 and 2011, I taught undergraduate courses: logic and fuzzy control, neural networks, and modern and classical control, among others. Since 2011, I have worked on autonomous navigation of robots in the Department of Automatic Control, CINVESTAV-DF. My work has been implemented on a real robot: motion planning methods, localization and map building methods and control laws in unstructured environments. In 2014, I participated in a research stay at the University of Bristol, U.K. There I developed an autonomous navigation system for a robot that is able to follow emergency signs to find emergency exits using a Kinect-like sensor.

Figure 5.1: Erik Zamora Gómez, author of this thesis.

My research focuses on **Robotic Autonomy**. Automatic robots are limited to controlled environments. We need autonomous robots in natural environments to increment the productivity and efficiency of several human activities: package delivery, cleaning, agriculture, surveillance, search & rescue, construction, transportation, and guiding people. The autonomy implies a number of problems: perception, localization, environment modeling, motion planning, control, navigation, etc. My research focuses on finding and solving new problems to extend the skills of autonomous robots.

E-mail: ezamora70@hotmail.com, ezamora1981@gmail.com, ezamorag@ipn.mx

Cellphone Number: 55-6099-1485

Chapter 6

Appendices

6.1 A basic autonomous navigation system

"More gold has been mined from the brains of men than has ever been taken from the earth." —Napoleon Hill

We go into details about autonomous navigation. We will describe a navigation system for a differential mobile robot (Koala robot), which is available in the Laboratory of Department of Automatic Control . All parts of the system will be explained: localization, mapping, planning, path smoothing and control.

6.1.1 Localization

The navigation system uses odometry to track the robot's state. The kinematic model of a differential-drive robot (Koala robot) can be described by the unicycle model,

$$\begin{aligned} x'(t) &= v(t)\cos\theta(t) \\ y'(t) &= v(t)\sin\theta(t) \\ \theta'(t) &= \varpi(t) \end{aligned} \tag{6.1}$$

where (x, y, θ) is the robot's state, $v(t)$ is the linear velocity of the robot, and $\varpi(t)$ is the angular velocity of the robot. These velocities are determined by the velocity of right

$v_r(t) = \frac{v_r(t)+v_l(t)}{2}$ and left $v_l(t) = \frac{v_r(t)-v_l(t)}{b}$ wheels, where b is the wheelbase of the vehicle (see Figure 6.1). The unicycle model must be discretized to estimate (x, y, θ). We use the model of Borenstein [89],

$$\begin{aligned} \Delta U_L &= c_L N_{L,k} \\ \Delta U_R &= c_R N_{R,k} \\ \Delta U_k &= (\Delta U_R + \Delta U_L)/2 \\ \Delta\theta_k &= (\Delta U_R - \Delta U_L)/b \\ \theta_k &= \theta_{k-1} + \Delta\theta_k \\ x_k &= x_{k-1} + \Delta U_k \cos\theta_k \\ y_k &= y_{k-1} + \Delta U_k \sin\theta_k \end{aligned} \tag{6.2}$$

where $N_{L,k}$ and $N_{R,k}$ are the left and right pulse increment of wheel encoders in a sample time k; ΔU_L and ΔU_R the incremental distance for the left and right wheel; ΔU_k is the incremental linear displacement of the center point c; $\Delta\theta_k$ is the incremental angular displacement; (x_k, y_k, θ_k) are the kinematic states; c_L and c_R are the conversion factors that translates encoder pulses into linear wheel displacement for left and right wheels. Figure 6.1 shows a graphical representation of these variables.

In order to improve accuracy of odometry, we calibrate the parameters (b, c_L, c_R) through an optimization method. The method consist of:

1. Selecting an initial position $(0, 0, 0)$ and a final position $\left(x_f^*, y_f^*, \theta_f^*\right)$ in a floor plane and measure the Cartesian coordinates of final point with respect to initial point.

2. Moving the robot from the initial to the final positions N times, following arbitrary trajectories and recollecting encoders data.

3. Applying an optimization method to identify $(b^*, c_L^*, c_R^*) = \arg\min F(b, c_L, c_R)$.We quantify percentage errors for linear displacement E_L and angular displacement E_A with respect to the total displacement given a trajectory. The objective function is

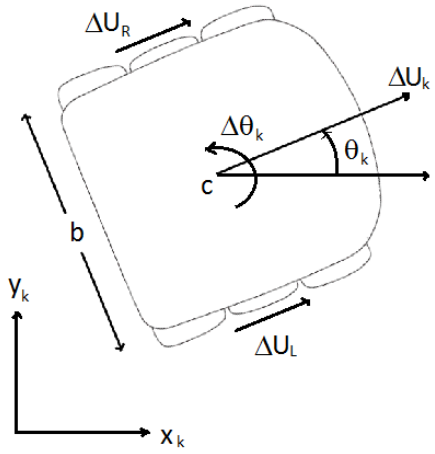


Figure 6.1: We show some parameters in the model of Borenstein for odometry.

$F\left(b, c_L, c_R\right) = \sum_{i=1}^{N} E_{L,i} + \sum_{i=1}^{N} E_{A,i}$ and

$$F\left(b, c_L, c_R\right) = \sum_{i=1}^{N} \frac{1}{d_i} \sqrt{\left(x_f^* - x_{f,i}\right)^2 + \left(y_f^* - y_{f,i}\right)^2} + \sum_{i=1}^{N} \frac{1}{d_{\theta,i}} \left|\theta_f^* - \theta_{f,i}\right| \tag{6.3}$$

where i is the path, $(x_{f,i}, y_{f,i}, \theta_{f,i})$ is the final position calculated by the odometry, $d_i = \sum_{j=1}^{M_i - 1} \sqrt{(x_{j+1,i} - x_{j,i})^2 + (y_{j+1,i} - y_{j,i})^2}$is the total linear displacement and $d_{\theta,i} = \sum_{j=1}^{M_i - 1} |\theta_{j+1,i} - \theta_{j,i}|$ is the total angular displacement (note that $\theta_f^*, \theta_{f,i} \in [-\pi, \pi]$ and M_i is number of positions).

We apply previous method with $\left(x_f^* = 6.826, y_f^* = 8.933, \theta_f^* = \pi/2\right)$. We move the robot $N = 5$ times through different trajectories. Figure 6.2 shows the trajectories obtained using nominal values $\left(b = 0.31, c_L = \frac{1}{22} \times 10^{-3}, c_R = \frac{1}{22} \times 10^{-3}\right)$, the average errors are $E_{L,avg} = 8.19\%$ and $E_{A,avg} = 3.57\%$. In contrast, Figure 6.3 shows the trajectories obtained using calibrated values $(b^* = 0.3128m, c_L^* = 4.3490 \times 10^{-5} m/seg, c_R^* = 4.3372 \times 10^{-5} m/seg)$, the average errors are $E_{L,avg} = 1.59\%$ and $E_{A,avg} = 0.56\%$. The odometry accuracy is improved. We use the Nelder-Mead simplex method [90] (in Matlab(R), *fminsearch*) to optimize the objective function 6.3.

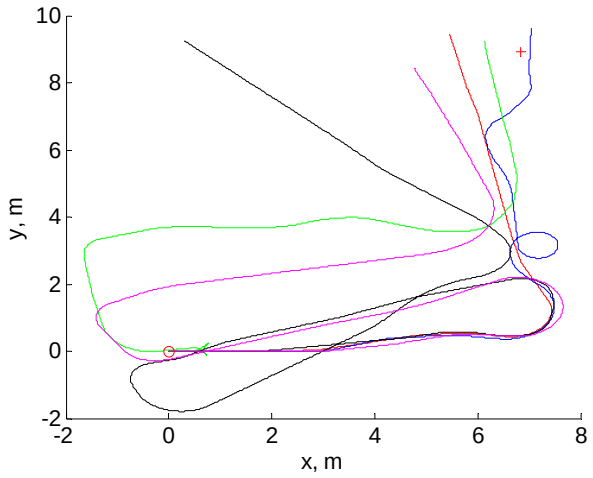


Figure 6.2: Initial and final poses are marked by a circle and a cross, respectively. Note that nominal values do not obtain a good odometry.

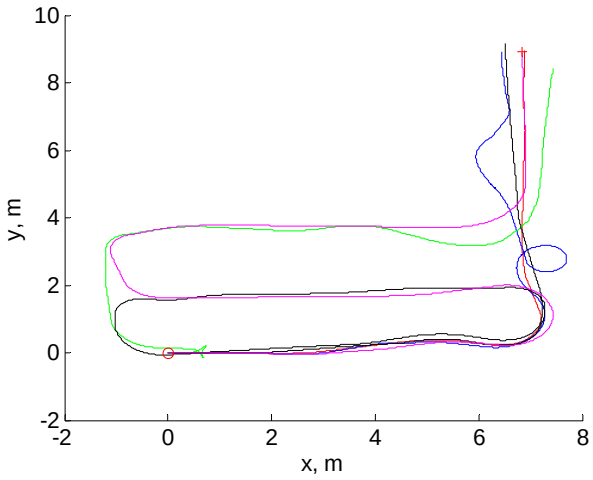


Figure 6.3: Initial and final poses are marked by a circle and a cross, respectively. The calibrated parameters improve the accuracy of odometry.

6.1.2 Mapping

Mapping is the task to model the environment. Maps are useful to locate the robot and to make motion plans. There are three types of maps: grid maps [91], features maps [58] and topological maps [92]. We use a 2-D occupancy grid map. This map divides the environment in a grid, which each cell contains the probability of this area is occupied by an obstacle. So the map is a matrix which entries are probabilities of occupation. Every time that the robot takes a range measurement from laser, some of the cells in the grid are updated by a binary Bayes filter (see [30] for mathematical details).

The procedure is as follows.

1. Initially, it is created a zero matrix with $n \times m$ dimensions; zero entries are equivalent to 0.5 probability of occupation.

2. The laser takes a set of range measurements, which are distances from sensor to obstacles.

3. Given the robot's state (x, y, θ) by odometry, the mapping algorithm updates the probabilities according to the measurements. We use a simple sensor model to update the map. The laser beam is modeled as a straight line. Every cell where the laser beam pass is updated (see Figure 6.4). The measurements out of range $[0, r_{\max}]$ are ignored.

We test this method building a couple of maps at the Department of Automatic Control (see Figure 6.5). There are inconsistencies with respect to the real environment because of the odometry error accumulation.

6.1.3 Planning

The planning problem can be stated as follows: given a map, an initial state $\mathbf{x}_I$ and a goal state $\mathbf{x}_G$, the planning algorithm must calculate a sequence of states $\mathbf{x}_1 = \mathbf{x}_I, \mathbf{x}_2, ..., \mathbf{x}_N = \mathbf{x}_G$, which conforms a free-collision trajectory. We use dynamic programming for planning in 2-D space $\mathbf{x} = (i, j)$, where i is the row and j is the column of a cell in the grid map. State

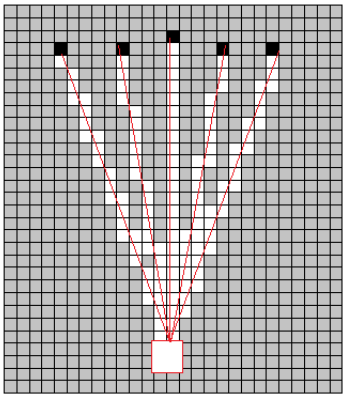


Figure 6.4: Red lines represent the laser beams and only the cells that passed the laser beam are updated with free or occupy probability (grey = 0.5, black = 1.0 and white = 0.0).

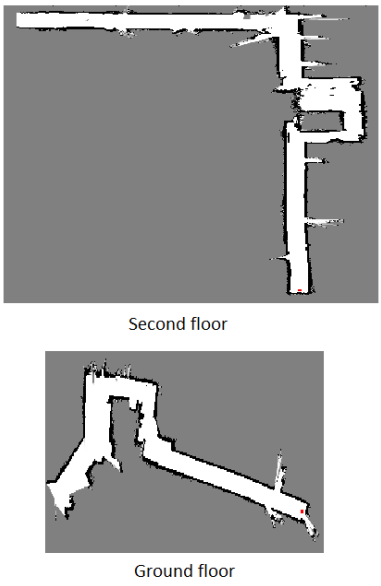


Figure 6.5: The grid maps of second and ground floors at Automatic Control Department, Zacatenco, CINVESTAV.

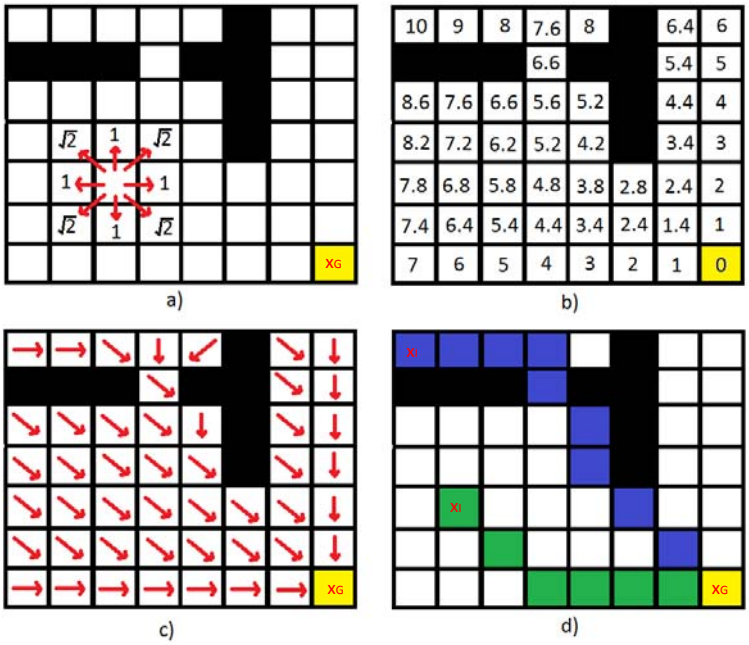


Figure 6.6: This is a small grid map as an example. (a) The robot can only move 8 different ways. The cost of action is assigned by Euclidean distance. (b) The value function is calculated according to the Algorithm 6.1. (c) The best action the robot can take is represented by arrows. (d) It shows two paths (blue and green) from different initial states.

space X is a discretized 2-D space $map(\mathbf{x})$. We restrict the robot actions $u \in U$ to $N_u = 8$ movements, as you see in Figure 6.6-(a). In order to minimize the distance, the cost $C(u)$ of each action is equal to Euclidean distance, assuming that the distance between horizontal or vertical adjacent cells is the unit.

We use Dynamic programming for plannning. This method uses the Bellman's principle of optimality to choose the best action. It begins to explore from goal location $\mathbf{x}_G$, updating the value function $V(\mathbf{x})$ of each free cell (i.e. $\mathbf{x} \in X_{free} = \{\forall \mathbf{x} | map(\mathbf{x}) \leq 0.1\}$); the occupied or uncertain cells (i.e. $\mathbf{x} \notin X_{free}$) are ignored. The value function $V(\mathbf{x})$ is the minimal cost to reach the goal from any state $\mathbf{x}$. Algorithm 6.1 calculates $V(\mathbf{x})$ to determine the best action for every free cell to reach the goal (see Figure 6.6-(b) and (c)); where $\mathbf{f}(\mathbf{x}, u)$

represents the transition action model. On the other hand, Algorithm 6.2 uses the value function to generate the path $\mathbf{P} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x}_N\}$ which contains a series of adjacent states from initial state $\mathbf{x}_1$ to the goal state $\mathbf{x}_N$ (see Figure 6.6-(d)).

Algorithm 6.1 *ValueFunction(map,*$\mathbf{x}_G$*)*

$V(\mathbf{x}) := \infty$ *for* $\forall \mathbf{x} \in X$
$V(\mathbf{x}_G) := 0$
open_states $:= \{\mathbf{x}_G\}$
while **open_states** *is not empty*
  $\mathbf{x}_{\min} := \arg\min_{\mathbf{x}} V(\mathbf{open\_states})$
  eliminate $\mathbf{x}_{\min}$ *from* **open_states**
  for $n = 1$ *to* N_u *do*
    $\mathbf{x}_{new} := \mathbf{f}(\mathbf{x}_{\min}, u(n))$
    if $\mathbf{x}_{new} \in X_{free}$
      if $V(\mathbf{x}_{\min}) + C(u(n)) < V(\mathbf{x}_{new})$
        open_states $:= \{\mathbf{open\_states}, \mathbf{x}_{new}\}$
        $V(\mathbf{x}_{new}) := V(\mathbf{x}_{\min}) + C(u(n))$
return V

Algorithm 6.2 *PathGeneration(*V*,*$\mathbf{x}_I$*)*

$\mathbf{P} := \{\mathbf{x}_I\}$
$\mathbf{x}_{next} := \mathbf{x}_I$
while $V(\mathbf{x}_{next}) \neq 0$
  $\mathbf{x}_{next} := \arg\min_f V(\mathbf{f}(\mathbf{x}_{next}, u))$ *for* $\forall u \in U$ *and* $\forall \mathbf{f}(\mathbf{x}_{next}, u) \in X$
  $\mathbf{P} := \{\mathbf{P}, x_{next}\}$
return $\mathbf{P}$

In order to prevent the generated paths are too close to the walls, we artificially widen the obstacles on the map according with the size of robot. Figure 6.7 shows a path generated by this approach with normal map and thick map. We can see that the path tends to be close to obstacles, but the robot size is considered by the greater thickness of obstacles.

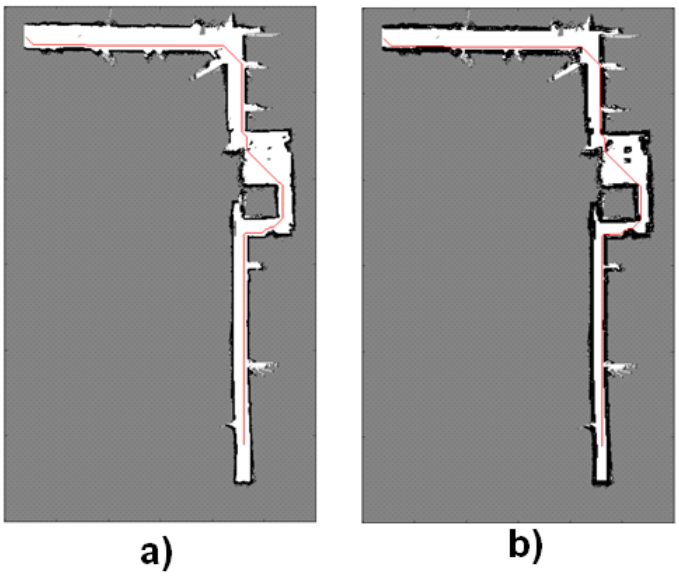


Figure 6.7: In order to prevent the generated paths are very close to the real obstacles (a), we artificially widen the obstacles in the map according with the size of robot (b).

6.1.4 Path smoothing

After obtaining a path in terms of cells, we must transform this path to calculate a metric path. The resulted path must be smoothed before executing the plan because non-smooth motions can cause slippage and over-actuation. Particularly, we should limit the curvature of paths to guarantee a smooth motion. The Koala robot can perform paths with maximum curvature about $k_{\max} = \frac{2}{b} \simeq 6.39m^{-1}$. We implement a simple algorithm to smooth the paths based on the Thrun's approach [142]. The idea is to optimize the objective function

$$F(\mathbf{P}) = \alpha_1 \sum_{i=2}^{N-1} \left\|\mathbf{P}_i^o - \mathbf{P}_i\right\|^2 + \alpha_2 \left[\sum_{i=1}^{N-1} \left\|\mathbf{P}_{i+1} - \mathbf{P}_i\right\|^2 + \sum_{i=2}^{N} \left\|\mathbf{P}_i - \mathbf{P}_{i-1}\right\|^2\right] \tag{6.4}$$

where $\mathbf{P}^o$ is the original path, $\mathbf{P}$ is the smooth path, $\mathbf{P}_i$ is a particular state in the path, α_1 and α_2 are weights. The first term penalizes the difference between original and smooth path. The second term is the smooth condition, which penalizes the distance among the adjacent states in smooth path. Considering $\mathbf{P} = \mathbf{P}^o$, we apply gradient-descent method to optimize $F(\mathbf{P})$. The derivatives of objective function are

$$\frac{\partial F}{\partial \mathbf{P}_i} = -2\alpha_1 \left(\mathbf{P}_i^o - \mathbf{P}_i\right) - 2\alpha_2 \left(\mathbf{P}_{i+1} - 2\mathbf{P}_i + \mathbf{P}_{i-1}\right) \quad \text{for } \forall i \in \{2, ..., N-1\} \tag{6.5}$$

where the initial and the goal state must be fixed.

Figures 6.8,6.9, and 6.10 show the result of smoothness (with $\alpha_1 = 2.0$, $\alpha_2 = 0.48$). The orientation of robot $\theta_i := \tan^{-1}\left(\frac{\Delta y_i}{\Delta x_i}\right)$ for original path (red lines) presents discontinuities, while the smooth path (blue lines) shows a better continuity. The curvature $k := \frac{\Delta\theta_i}{\sqrt{\Delta x_i^2 + \Delta y_i^2}}$ of original path has a maximum about $55m^{-1}$, while the maximum curvature in smooth path is $< 4m^{-1}$; below what is allowed.

6.1.5 Control

The smoothed path will be used as reference for a controller. We use the following tracking controller for Koala motion [116][117]:

$$\begin{aligned} v &= -k_1(v_r, \varpi_r)e_1 \\ \varpi &= -k_4 v_r \tfrac{\sin e_3}{e_3} e_2 - k_3(v_r, w_r)e_3 \end{aligned} \tag{6.6}$$

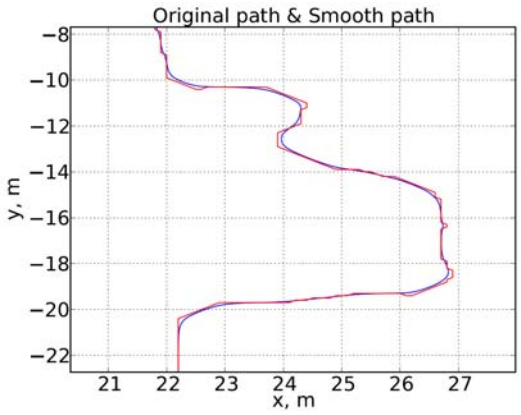


Figure 6.8: We show the original and smooth path for comparison.

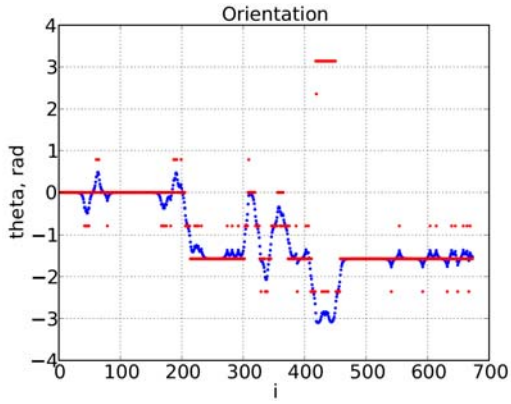


Figure 6.9: After the smooth process, the orientation has smooth transitions.

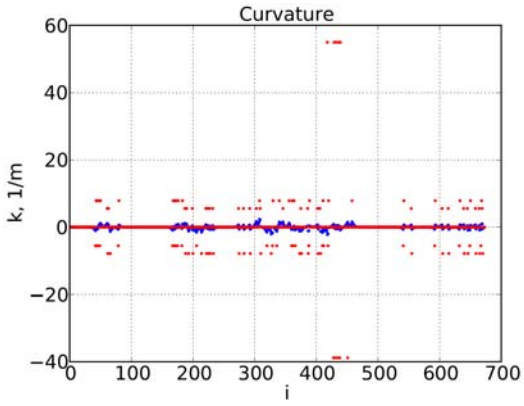


Figure 6.10: After the smooth process, the curvature is limited $k < 4m^{-1}$.

where control gains are

$$\begin{aligned} k_1(v_r, \varpi_r) = k_3(v_r, \varpi_r) = 2\varsigma\sqrt{\varpi_r^2 + g v_r^2} \\ k_4 = g \end{aligned} \tag{6.7}$$

ς and g are positive real constants. The errors expressed in the body frame of mobile robot are

$$\begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} e_x \\ e_y \\ e_\theta \end{pmatrix} \tag{6.8}$$

where $e_x = x - x_r$, $e_y = y - y_r$, $e_\theta = \theta - \theta_r$ This controller requires $(x_r, y_r, \theta_r, v_r, \varpi_r)$, but we only have a path $\mathbf{P} = \{\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, ..., \begin{bmatrix} x_N \\ y_N \end{bmatrix}\}$ with no orientation nor velocities. So we assign directly $x_r := \{x_i\}_{i=1}^N$ and $y_r := \{y_i\}_{i=1}^N$; and the rest reference signals (the orientation $\theta_r := \{\theta_i\}_{i=1}^N$, the linear velocity $v_r := \{v_i\}_{i=1}^N$, the angular velocity $\varpi_r := \{\varpi_i\}$) can be calculated as follows,

$$\theta_i = \left\{ \begin{array}{ll} arctan2(\Delta y_i, \Delta x_i) & \text{for } \forall i \in \{1, ..., N-1\} \\ arctan2(\Delta y_{N-1}, \Delta x_{N-1}) & \text{for } i = N \end{array} \right\} \tag{6.9}$$

$$v_i = \left\{ \begin{array}{ll} \frac{1}{T_r}\sqrt{\Delta x_i^2 + \Delta y_i^2} & \text{for } \forall i \in \{1, ..., N-1\} \\ \frac{1}{T_r}\sqrt{\Delta x_{N-1}^2 + \Delta y_{N-1}^2} & \text{for } i = N \end{array} \right\} \tag{6.10}$$

$$\varpi_i = \left\{ \begin{array}{cc} \frac{\Delta\theta_i}{T_r} & \text{for } \forall i \in \{1, ..., N-1\} \\ \frac{\Delta\theta_{N-1}}{T_r} & \text{for } i = N \end{array} \right\} \tag{6.11}$$

where $\Delta y_i = y_{i+1} - y_i$, $\Delta x_i = x_{i+1} - x_i$, $\Delta\theta_i = \theta_{i+1} - \theta_i$ and T_r is the virtual sample time of reference trajectory. This parameter regulates the velocity profile of real robot.

In order to evaluate the performance of controller, we make some simulations in Simulink(R) and Python. Figure 6.11 shows the discrete-time controller with sample time $T = 0.1seg$ and the kinematic model of Koala robot with saturation in velocities. The reference trajectories under test are three trajectories (a straightline path, a cycle and an "eight path"). The initial state is $(-0.1m, -0.1m, \pi)$ for all simulations. We choose these initial values because it is the worst case where the angular error e_θ and error position (e_x, e_y) are initially maximum (the resolution of grid maps is $0.1m$). The saturations $\pm 0.55m/seg$ are the maximum velocities of wheels for Koala robot. The parameterized equations for each test trajectory are the following:

For a straight line

$$\begin{array}{c} x_r(t) = v_o t \\ y_r(t) = 0 \end{array} \tag{6.12}$$

For a circular path

$$\begin{array}{c} x_r(t) = r_o \sin(\frac{2\pi}{p_o} t) \\ y_r(t) = r_o \left(1 - \cos\left(\frac{2\pi}{p_o} t\right)\right) \end{array} \tag{6.13}$$

For an "eight path"

$$\begin{array}{c} x_r(t) = r_o \sin(\frac{4\pi}{p_o} t) \\ y_r(t) = 2r_o \left(1 - \cos\left(\frac{2\pi}{p_o} t\right)\right) \end{array} \tag{6.14}$$

where $t \in [0, 60]seg$, $r_o = 1m$, $p_o = 60seg$, $v_o = 0.1m/seg$.

Figure 6.12 shows the results of these tests with $(\varsigma = 2, g = 10)$. The controller compensates well the initial error and we compensate the final error maintaining the final state of reference trajectory $(x_N, y_N, \theta_N, v_N, \varpi_N)$, until the robot state reaches it under certain tolerance $\sqrt{e_x^2 + e_y^2} < \varepsilon_N$ (in simulation, $\varepsilon_N = 0.01m$).

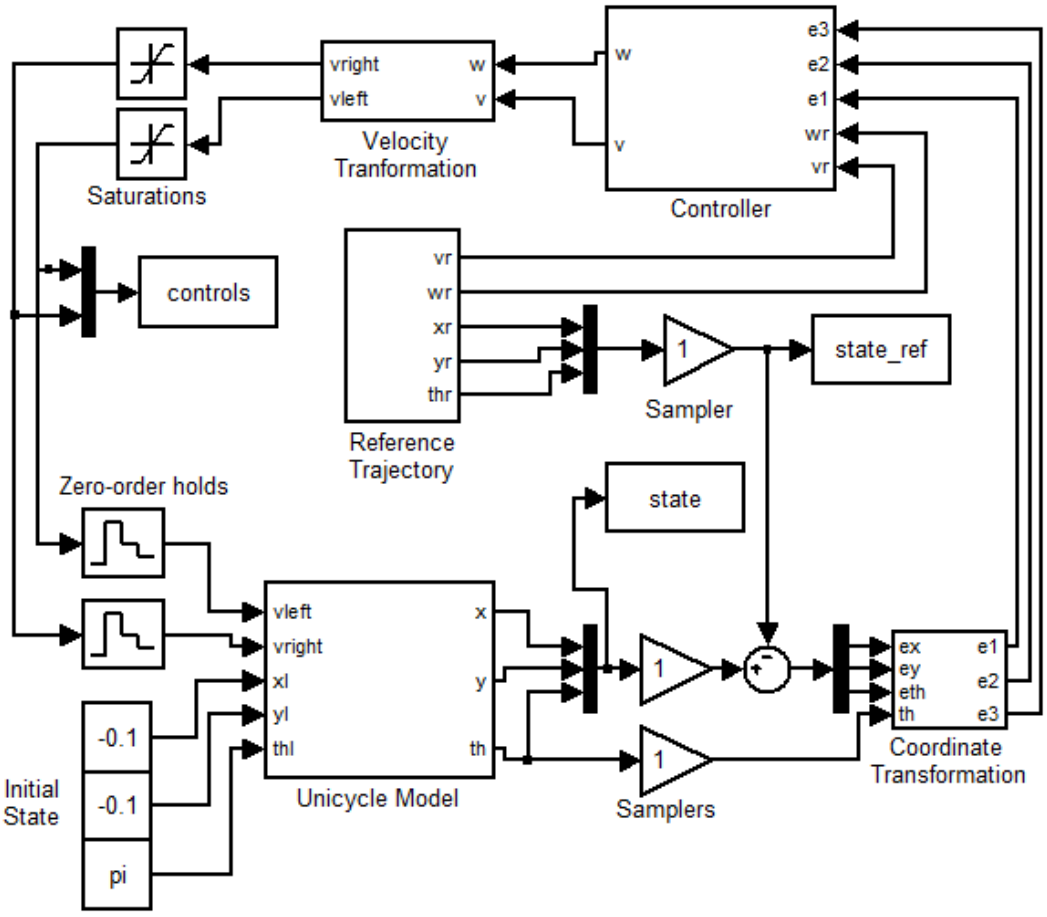


Figure 6.11: The discrete-time controller is applied to an continuous-time unicycle model that represent the kinematics of Koala robot. The simulation considers the velocity saturation effect in left and right wheel.

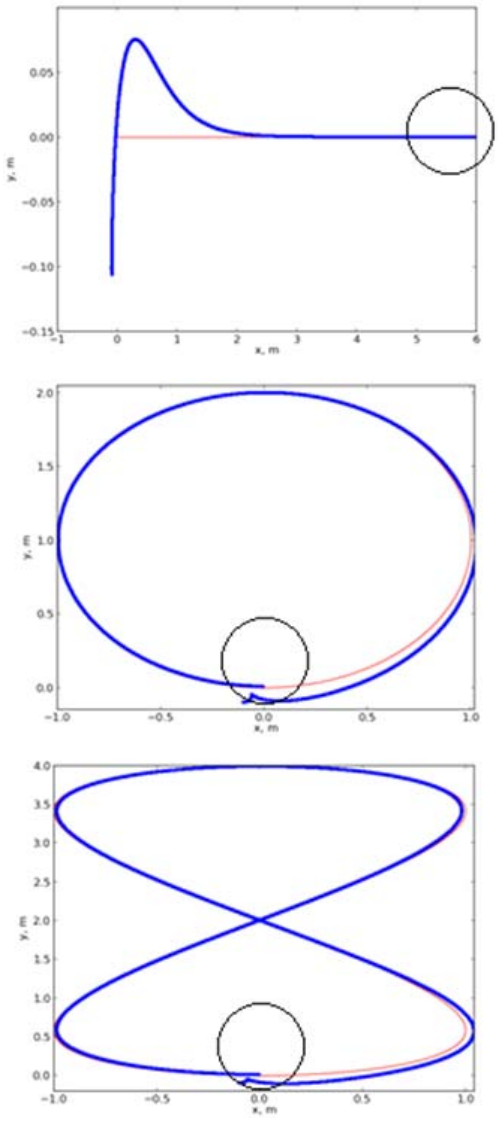


Figure 6.12: We show how the controller follows the reference trajectories (red lines). Blue lines are robot trajectories.

6.1.6 Experiment

The Koala robot was tested in an office environment at Department of Automatic Control, performing a small navigation task showed in Figure 6.13. Previously, the robot was guided manually through the environment to acquire laser and encoders measurements. The navigation system builds an occupancy grid map Figure 6.13-(a). The planner builds a value function $V(\mathbf{x})$ and generates the optimal free-collision path, Figure 6.13-(b). Finally, the controller tracks the robot state to the reference trajectory; some snapshots are shown in Figure 6.13-(c).

This experiment shows that:

- The odometry is quite problematic because the map and the navigation task are corrupted by odometry error, which is accumulated over time. It is indispensable to implement a SLAM method to avoid these problems.
- The controller can guarantee that the error in the final position of the robot can be less than $0.05m$. This is because the robot is always behind the reference state trajectory.

6.2 Autonomous navigation in unknown environments guided by emergency signs

> *"Lucky is the society that enjoys the gradual acquisition of a perfect balance between imitation and curiosity, between slavish, unthinking copying and progressive, rational experimentation." —Desmond Morris*

In this section, we present an autonomous navigation system able to find the exit, guided by emergency signs in unknown environments. Initially, the robot does not have a map and does not know where the emergency exit is. But it has a SLAM system that creates a map as the robot moves in the environment. And it has a perception system that observes the environment and interprets emergency signs (see Figure 6.14) to plan how to move toward

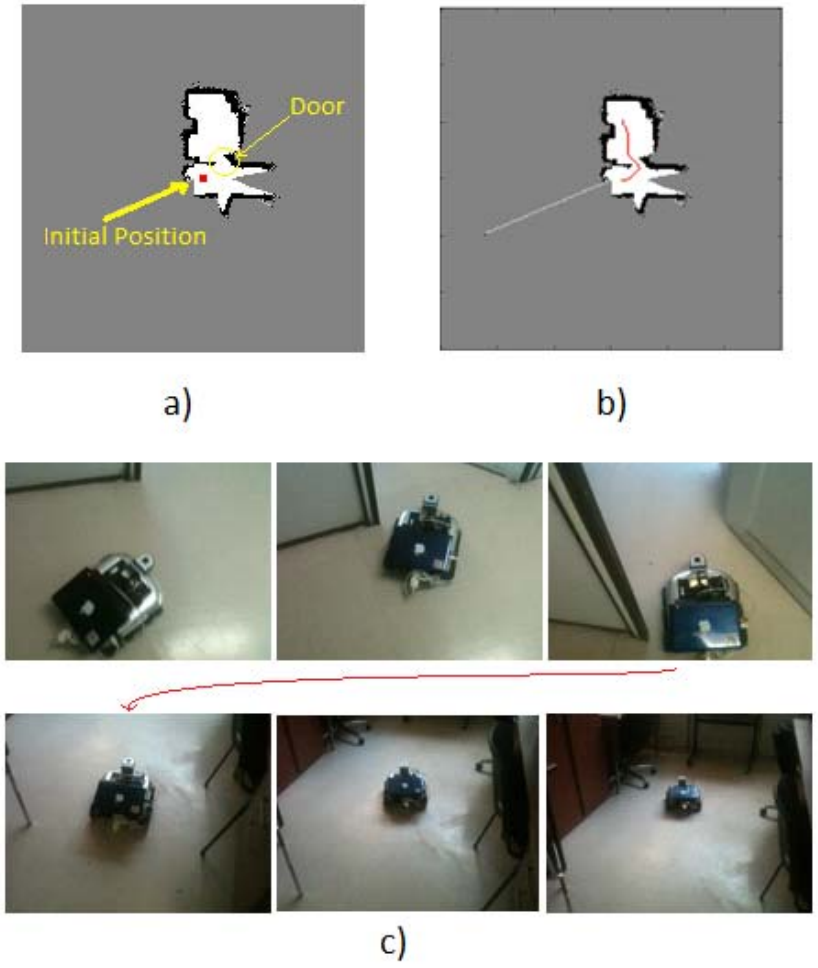


Figure 6.13: An small autonomous navigation experiment.

the possible exit. The map is necessary for preventing that the robot visits the same place more than once during exploration. Note that emergency signs are signs that were already in the building and used by humans for the same purpose. The environment was not modified in any way for facilitating navigation. This project was done as part of a research stay at the University of Bristol in England led by Walterio Mayol-Cuevas. A video showing the operation of this navigation system is online at

https://www.youtube.com/watch?v=RAj70AGsXls&feature=youtu.be**??**

The robot used for this project was an iRobot Create [156]. It is a differential robot that has encoders for odometry and bumpers to avoid obstacles. We mounted a RGBD camera [157] on the robot; which is used to build the map and observe emergency signs. The software was developed on ROS (Robot Operating System) [158] that facilitates the construction of a modular software system. ROS already have some software that were reused for this project: the driver for iRobot Create [159], the driver for the Kinect sensor [160] and Gmapping [161], which is a SLAM method.

The navigation system diagram is shown in Figure 6.15. The perception is in charge of: (1) detecting and segmenting the signs appearing in RGB images, (2) recognizing the arrow direction on the segmented image of the signs, (3) interpreting this direction in geometrical terms of the environment, (4) creating a map of the observed signs to avoid recognizing and interpreting signs already detected, and (5) choosing the nearest sign to be followed. The mapping is comprised of the module "Kinect to Laser" and the module "SLAM". The first takes only a horizontal region of the depth image and transforms it into a laser format. The second builds a 2-D occupancy grid map and locates the robot. The planning takes the goal, the map and the robot's position to generate a sequence of waypoints that the robot must follow to reach the exit. The controller follows these waypoints to execute the plan and incorporates a module to avoid obstacles. Note that the plan changes every time that the map changes or the planned path is invalid. This gives flexibility to the navigation system to explore the environment. In the following sections, we will present the details of each part

Figure 6.14: We show the signals that were used to guide the robot to the emergency exit. And we show some pictures of the environment in which the robot navigates and how the signals were set.

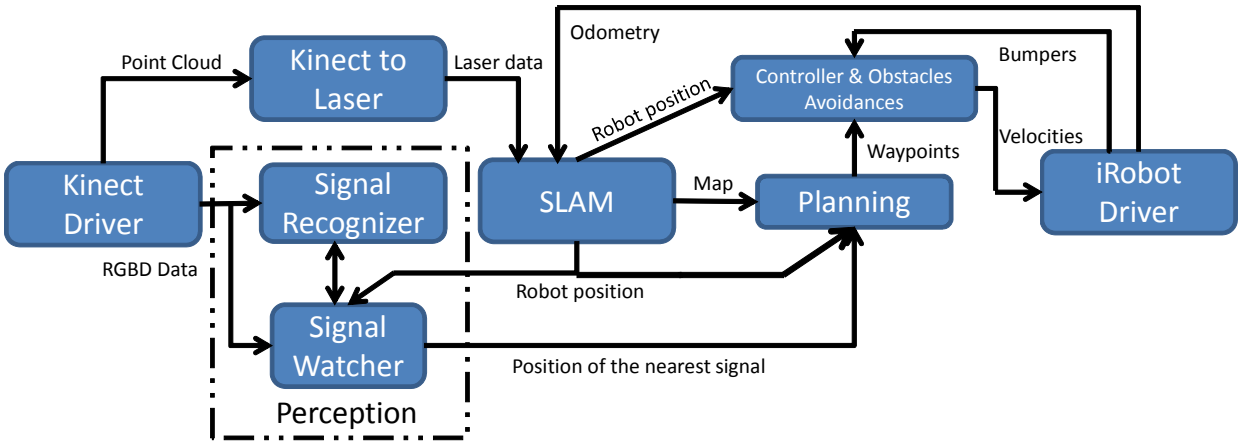


Figure 6.15: This is the diagram of the navigation system.

of this navigation system.

6.2.1 Perception

The objective of perception system is to detect, recognize and interpret emergency signs seen from RGBD camera, and provide to the planning system the position of the nearest sign and the direction of the emergency exit given by this sign. This perception system is divided into two modules: Signal Watcher and Signal Recognizer.

Signal Watcher functions are the following:

- Monitoring when one or more emergency signs appear in RGB images.
- Building a map of the positions of emergency signs. This map is used to avoid recognizing a sign more than once.
- Determining the closest signs to the robot for following this signs.
- Calling Signal Recognizer to process emergency signs that have been seen for the first time (i.e. they are not found on map previously).

- Stopping the robot during recognition process to reduce errors due to the blur images.

The Signal Recognizer recognizes the direction of arrow on emergency signs. First, the region where an emergency sign appears in RGB image is segmented using an Adaboost classifier [162][163][164]. A centroid (x_i, y_i, z_i) and a plane (a_3, a_2, a_1, a_0) of emergency sign is determined using the point cloud corresponding to the region in RGB image (where a_3, a_2, a_1, a_0 are coefficients of plane equation). The plane is adjusted to point cloud by RANSAC method [165]. This process of detecting the centroid and plane is repeated with all images taken during 10 seconds. The centroids are spatially clustered to identify the most reliable detections and discard false detections. A centroid and a plane are assigned to each group as the mean of the centroids and planes of the group, respectively. Groups with more number of detections (> 10) are processed to recognize the direction of the arrow by the correlation between templates of arrows and the last image of detected sign (see Figure 6.16). Finally, the position and direction of the emergency sign are projected on the floor plane. And the coordinates are transformed from sensor framework to robot framework. These data of the detected signs are sent to Signal Watcher to add them to the map of the sign positions.

6.2.2 Map building

Since the environment is unknown at the beginning, the robot must be able to build a map as it explores the environment to avoid visiting the same place more than once. Thus the search for emergency exit is more efficient, and we can ensure that the robot explores the whole environment, if necessary, until finding the exit. The mapping is comprised of two modules: "Kinect to Laser" and "SLAM".

The "Kinect to Laser" module takes a horizontal region ($[-0.20, 0.80]m$ of height) of the depth image from a RGBD camera and transforms these data into format of a laser. It determines the shortest depth for each angle of that horizontal region. Due to the limitations of RGBD camera, this virtual laser only covers about $60°$ and only sees obstacles between $[0.6, 6.0]m$ (see Figure 6.17). This is a major limitation because the robot cannot see nearby obstacles ($< 60cm$), so it may collide with objects that come suddenly into this blind region.

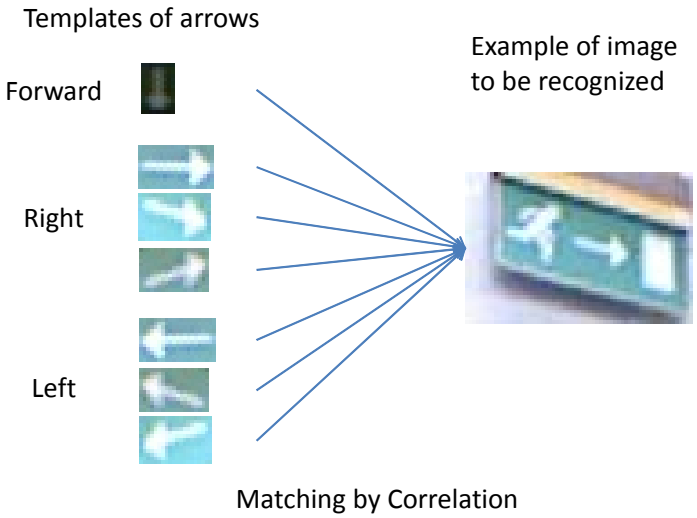


Figure 6.16: This illustrates the matching process between templates of arrows and the image of emergency signal to recognize the direction to the emergency signal.

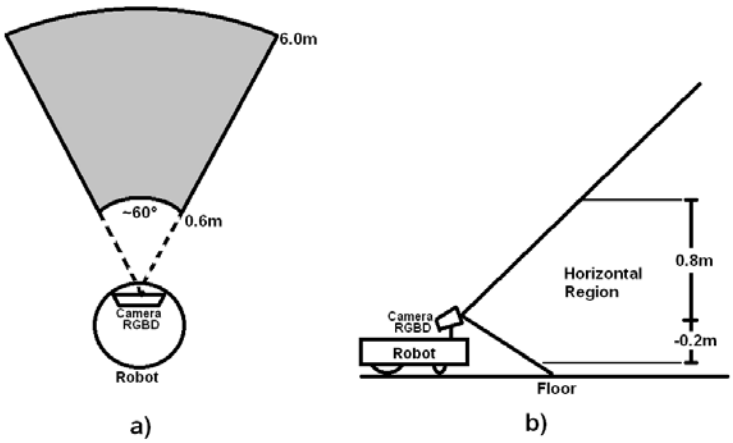


Figure 6.17: This shows (a) the region that the camera RGBD can detect obstacles and (b) the horizontal region where the shortest depth is determined.

This module also takes into account the slope of RGBD camera, reading its angular position and transforming point cloud from sensor framework to robot framework. This allows the camera can increase its angular position for seeing the emergency signs that are located at ceiling.

The module "SLAM" builds the map from laser data and odometry provided by the robot's encoders. This is a vital module for the proper functioning of the navigation system because if the map is incorrect or location fails then there is no guarantee that the robot can find the emergency exit. We use the method Gmapping to perform this task. GMapping is an efficient Rao-Blackwellized particle filer [35] in which each particle carries an individual map of the environment. The method can be summarized by the following four steps:

1. Sampling: The next generation of particles is obtained from the past generation by sampling from a proposal probability distribution.

2. Importance Weighting: An individual importance weight is assigned to each particle according to the importance sampling principle.

3. Resampling: Particles are drawn with replacement proportional to their importance weight. After resampling, all the particles have the same weight.

4. Map Estimation: For each particle, the corresponding map estimate is computed based on the trajectory of that sample and the history of observations.

This method is already implemented as a standard module in ROS [161]. We reuse it in this project.

6.2.3 Planning & Execution

The planning system decides to where the robot will move to find the emergency exit. We use a conventional algorithm in 2-D: A* search [101][102]. This algorithm requires three inputs: the map, the current location of the robot and the goal. Commonly, the goal is a point (x_G, y_G), but the problem is to find the emergency exit, so in principle we have no goal

defined in terms of coordinates. What we have is the information (x_s, y_s, θ_s) of the nearest emergency sign to robot; where (x_s, y_s) is the location of the sign and θ_s is the direction of the arrow in the sign. With this information we can project a straight line in the direction of θ_s and located on the sign (x_s, y_s), and take the intersection (x_G, y_G) between this line and the border of the map as the goal (see Figure 6.18). The planning generates a path toward this goal assuming that the unobserved region of the map is free of obstacles. The path is a sequence of coordinates (x, y) spaced every 1m, called waypoints.

A P controller follows these waypoints using the following control law to assign linear v and angular w velocities:

$$\begin{aligned} v &= V_{\max} sat(0, 1, \tfrac{c-|d\theta|}{c}) \\ w &= W_{\max} sat(-1, 1, \tfrac{d\theta}{c}) \end{aligned} \tag{6.15}$$

where $V_{\max}$ y $W_{\max}$ are maximum linear and angular velocities, c is a parameter to adjust the velocity of turns, $sat()$ is the saturation function, and $d\theta = \arctan 2 \left(\frac{y-y_R}{x-x_R}\right) - \theta_R$ is the angular difference between direction to next waypoint (x, y) and robot orientation θ_R. Note that if the robot collides with some unexpected obstacle, then a preprogrammed routine is activated to avoid the obstacle.

6.2.4 Experiments

We made seven experiments to evaluate the effectiveness of this navigation system. One of which was completely successful reaching the emergency exit, 3 were partially successful because the robot just lost in the environment near to the exit, and 3 were completely failed since it could not leave the initial region. The cause of all these failures was due to SLAM system that lost the location of robot. Since the angular range of virtual laser is very short (˜60 °), the SLAM system does not have enough information to keep robustly the location of robot. Note the importance of SLAM system in the Figure 6.15. The systems of planning, controlling, and perception depend on robot localization. If this fails, then the other systems also fail. Possibly changing the virtual laser for a laser with over 180° angular range, the SLAM system will be more reliable. Figure 6.19 shows some snapshots of the successful

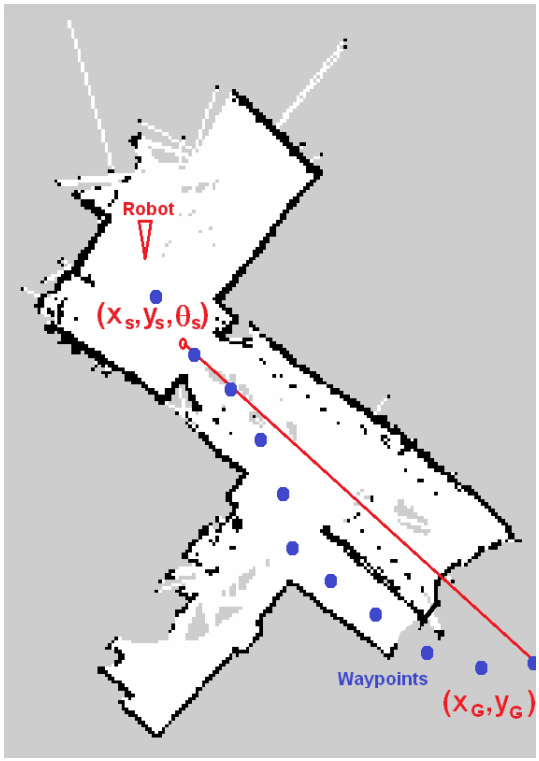


Figure 6.18: This illustrates how to calculate a goal (x_G, y_G) through information provided by emergency signal (x_s, y_s, θ_s). The goal is used by A* search to compute the path, which is a sequence of points (in blue).

experiment. A video showing this experiment is online at

https://www.youtube.com/watch?v=RAj70AGsXls&feature=youtu.be**??**

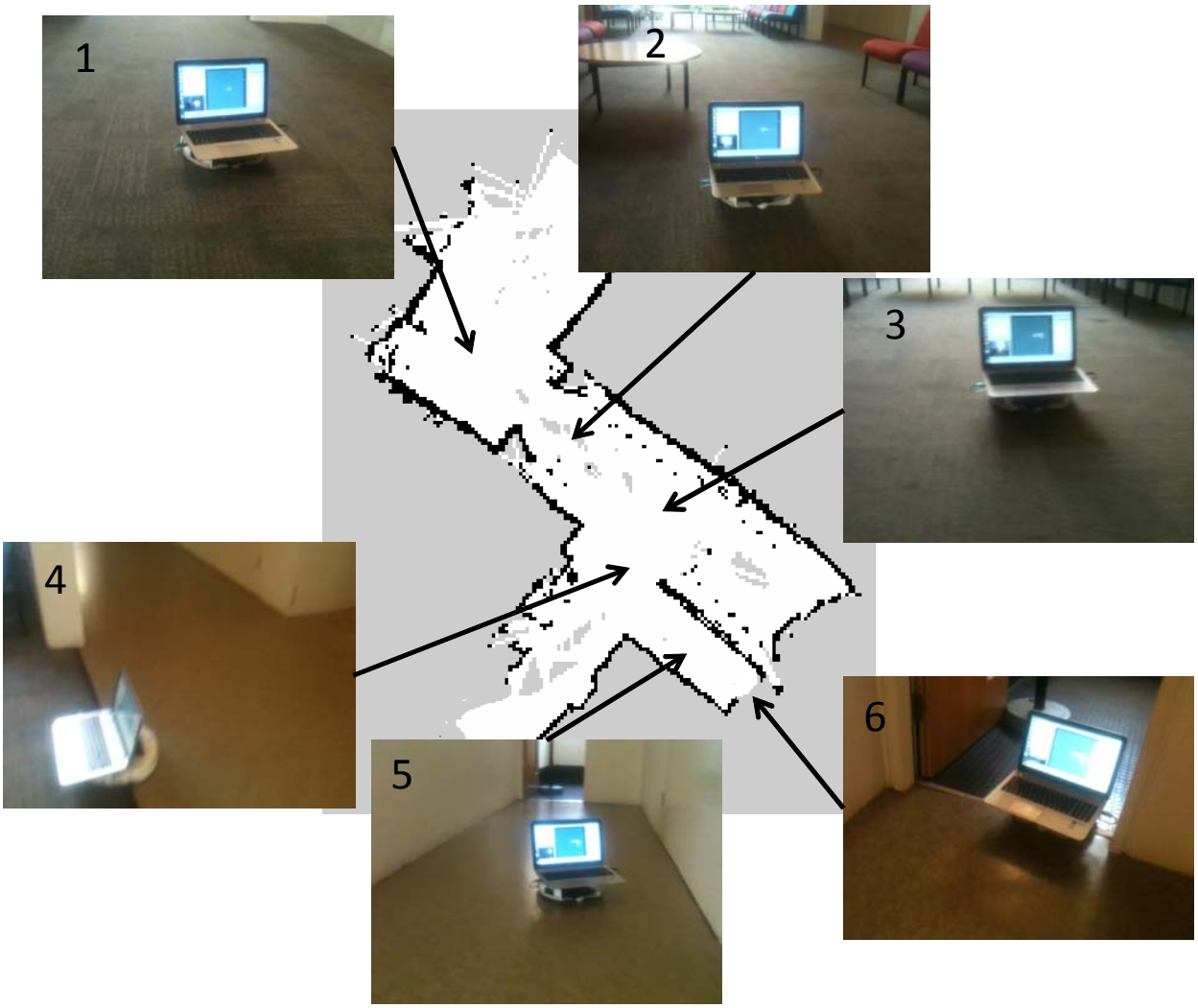


Figure 6.19: We show some snapshots during the travel of robot to the emergency exit.

Bibliography

[1] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian andPamela Mahoney. "Stanley: The robot that won the DARPA Grand Challenge", Journal of Robotic Systems - Special Issue on the DARPA Grand Challenge, 23(9): 661-692, 2006.

[2] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker , Robert E. Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M. Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M. Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M. Snider, Joshua C. Struble, Anthony (Tony) Stentz, Michael Taylor , William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar, "Autonomous driving in urban environments: Boss and the Urban Challenge," Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I, Vol. 25, No. 8, June, 2008, pp. 425-466.

[3] RobotCar UK project. Consulted in 23/07/2014 http://mrg.robots.ox.ac.uk/robotcar/

[4] Perez A., Karaman S., Shkolnik A., Frazzoli E., Teller S., Walter M.R., "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms", IEEE/RSJ International Conference Intelligent Robots and Systems, pp.4307-4313, 2011.

[5] Stollenga M., Pape L., Frank M., Leitner J., Forster A, Schmidhuber J., "Task-Relevant Roadmaps: A Framework for Humanoid Motion Planning", 2013 (to be published).

[6] DARPA (2012) "Darpa Robotics Challenge". Recuperado el 01/09/2013 de http://www.theroboticschallenge.org/

[7] Savage J., Negrete M., Matamoros M., Cruz J., Contreras L., Pacheco A., Figueroa I., Márquez J., "Pumas@Home 2013 Team Description Paper", RoboCup, 2013.

[8] Pineda L. A., Meza I. V., Fuentes G., Rascón C., Peña M., Ortega H., Reyes-Castillo M., Salinas L., Ortega J. D., Rodríguez-García A., Estrada V., "The Golem Team, RoboCup@Home 2013", RoboCup, 2013.

[9] Sucar L. E., Morales E., Heyer P., Vasquez I., Palacios-Alonso M. A., Escalante H. J., Oropeza J. M., Avila S., Rocha A., Herrera J., Ocana A., Reyes A., Vázquez H., Zorilla P., Hayet J., Rivera M., Esquivel J., Rodriguez D., Estevez C., "Markovito's Team Description RoboCup@Home", RoboCup, 2012.

[10] Vargas H., Olmedo E., Martínez D., Poisot V., Perroni A., Rodriguez A., Granillo O., Merino C., Antonio W., Carbonel C., Portillo A., "Donaxi@HOME Project", RoboCup, 2013.

[11] Amazon Prime Air. Consulted in 23/07/2014 http://www.amazon.com/b?node=8037720011

[12] Matternet Inc., (2011) "Matternet". Consulted in 23/07/2014 http://matternet.us/

[13] Swisslog Inc. (2013) "Automated Materials Transport Systems". Consulted in 23/07/2014 http://www.swisslog.com/index/hcs-index/hcs-systems.htm

[14] iRobot Corp. (2011) "Aspiradora robótica". Consulted in 23/07/2014 http://www.irobot.com.mx/

[15] Lely Group (2013) "Mobile barn cleaner". Consulted in 23/07/2014 de http://www.lely.com/en/housing/mobile-barn-cleaner/discovery

[16] Zucchetti Centro Sistemi S.P.A. (2011) "Ambrogio Robot". Consulted in 23/07/2014 http://www.ambrogiorobot.com/es/

[17] Eaton RP., Katupitiya J., Siew KW, Howarth B., "Autonomous farming: modelling and control of agricultural machinery in a unified framework", International Journal of Intelligent Systems Technologies and Applications, vol. 8, no.1-4, pp. 444 - 457, 2010.

[18] University of New South Wales (2013) "Associate Professor Jayantha Katupitiya". Consulted in 23/07/2014 http://www.mech.unsw.edu.au/info-about/contact-us/staff-database/katupitiya

[19] FroboMind Project (2013) "FroboMind is a robot control system software platform designed for field robotics research". Consulted in 23/07/2014 http://www.frobomind.org/index.php/Main_Page

[20] Evan Ackerman (2012) "Japanese Security Firm to Start Renting Surveillance Drones". Consulted in 23/07/2014 http://spectrum.ieee.org/automaton/robotics/military-robots/japanese-security-firm-to-start-renting-surveillance-drones

[21] Siciliano B., Khatib O., (Eds.), "Handbook of Robotics", Springer, 2008.

[22] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, Oskar von Stryk, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots", RoboCup, 2013.

[23] Stroupe A., Huntsberger T., Okon A., Aghazarian H., Robinson M., "Behavior-based multi-robot collaboration for autonomous construction tasks", IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.1495-1500, 2005.

[24] Durrant-Whyte H., Roy N., Abbeel P., "Construction of Cubic Structures with Quadrotor Teams", Book: Robotics:Science and Systems VII, MIT Press, pp. 177-184, 2012.

[25] Knepper R. A., Layton T., Romanishin J., Rus D, "IkeaBot: An Autonomous Multi-Robot Coordinated Furniture Assembly System", IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 2013.

[26] KIVA Systems (2013) "Solutions". Consulted in 23/07/2014 http://www.kivasystems.com/solutions/

[27] Burgard. W, Cremers A.B., Fox D., Hanel D., Lakemeyer G., Schulz D., Steiner W., Thrun S., "The interactive museum tour-guide robot", MI-98, 1998.

[28] Thrun S., Bennewitz M., Burgard W., Cremers A.B., Dellaert F., Fox D., Hahnel D., Rosenberg C., Roy N., Schulte J., Schulz D., "MINERVA: a second-generation museum tour-guide robot", IEEE International Conference on Robotics & Automation, vol. 3, pp.1999-2005, 1999.

[29] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, "Introduction to Autonomous Mobile robots", 2nd edition, The MIT Press, 2011.

[30] S. Thrun, W. Burgard, D. Fox. "Probabilistic Robotics", MIT Press, 2005.

[31] G. Dissanayake, S. B. Williams, H. Durrant-Whyte, T. Bailey. "Map Management for Efficient Simultaneous Localization and Mapping (SLAM)", Journal of Autonomous Robots,12(3): 267-286, 2002.

[32] S. Thrun, M. Montemerlo, "The GraphSLAM algorithm with applications to large-scale mapping of urban structures", International Journal on Robotics Research, 25(5/6):403-429, 2006.

[33] J. Folkesson, H. Christensen, "Closing the Loop With Graphical SLAM", IEEE Transactions on Robotics, 23(4): 731-741, 2007.

[34] R. M. Eustice, H. Singh, J. J. Leonard, "Exactly Sparse Delayed-State Filters for View-Based SLAM", IEEE Transactions on Robotics, 22(6): 1100-1114, 2006.

[35] G. Grisetti, C. Stachniss, W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", IEEE Transactions on Robotics, 23(1):34-46, 2007.

[36] K. L. Ho, P. Newman, "Loop closure detection in SLAM by combining visual and spatial appearance", Robotics and Autonomous Systems, 54(9), 740-749, 2006.

[37] H. El Mokni, F. Govaers, "Coupled laser inertial navigation system for pedestrian tracking", 8th Workshop on Positioning Navigation and Communication, 176-179, 2011.

[38] J. Nieto, J. Guivant, E. Nebot, "DenseSLAM: Simultaneous Localization and Dense Mapping", The International Journal of Robotics Research, 25: 711-744, 2006.

[39] T. Kollar, N. Roy, "Trajectory Optimization using Reinforcement Learning for Map Exploration",The International Journal of Robotics Research, 27(2):175–196, 2008.

[40] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, B. Stewart, "Distributed Multirobot Exploration and Mapping", Proceedings of the IEEE, 94(7):1325-1339, 2006.

[41] H. Durrant-Whyte, T. Bailey, "Simultaneous localization and Mappping (SLAM): Part I The essential algorithms", IEEE Robotics & Automation Magazine, 13(2):99-110, 2006.

[42] T. Bailey, H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II", IEEE Robotics & Automation Magazine, 13(3):108-117, 2006.

[43] S. Huang, G. Dissanayake, "Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM", IEEE Transactions on Robotics, 23(5):1036-1049, 2007.

[44] D. Rodriguez-Losada, F. Matia, A. Jimenez, R. Galan, "Consistency improvement for SLAM - EKF for indoor environments", IEEE International Conference on Robotics and Automation, 418-423, 2006.

[45] E. D Nerurkar, S. I Roumeliotis, "Power-SLAM: a linear-complexity, anytime algorithm for SLAM", The International Journal of Robotics Research, 30: 772-788, 2011.

[46] S. Julier, J. Uhlmann, "Using covariance intersection for SLAM", Robotics and Autonomous Systems, 55(1):3-20, 2007.

[47] S. Thrun, Y. Liu,, D. Koller, A. Y. Ng, Z. Ghahramani, H. Durrant-Whyte, "Simultaneous Localization and Mapping with Sparse Extended Information Filters", The International Journal of Robotics Research, 23: 693-716, 2004.

[48] M. Walter, R. Eustice, J. Leonard, "A provably consistent method for imposing sparsity in feature-based SLAM information filters", International Symposium of Robotics Research, 2005.

[49] Z. Wang, S. Huang, G. Dissanayake, "D-SLAM: A Decoupled Solution to Simultaneous Localization and Mapping", International. Journal of Robotics Research, 26(2):187-204, 2007.

[50] F. Dellaert, M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing", The International Journal of Robotics Research, 25: 1181-1203, 2006.

[51] M.Kaess, H. Johannsson, R. Roberts, V. Ila, J Leonard, F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree", The International Journal of Robotics Research, 31: 216-235, 2012.

[52] L. M. Paz, J. D. Tardós, J. Neira, "Divide and Conquer: EKF SLAM in $O(n)$", IEEE Transactions on Robotics, 24(5):1107-1120, 2008.

[53] S. Huang, Z. Wang, G. Dissanayake, "Sparse Local Submap Joining Filter for Building Large-Scale Maps", IEEE Transactions on Robotics, 24(5):1121-1130,2008.

[54] C. Cadena, J. Neira, "SLAM in O(log n) with the Combined Kalman - Information filter", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2069-2076, 2009.

[55] M. Bosse, P. Newman, J. Leonard, S. Teller, "Simultaneous Localization and Map Building in Large-Scale Cyclic Environments Using the Atlas Framework", The International Journal of Robotics,23: 1113-1139, 2004.

[56] B. Lisien, D. Morales, D. Silver, G. A. Kantor, I. Rekleitis, H. Choset, "The hierarchical atlas", IEEE Transactions on Robotics, 21(3):473-481, 2005.

[57] U. Frese, "Efficient 6-DOF SLAM with Treemap as a Generic Backend", IEEE International Conference on Robotics and Automation, 4814-4819, 2007.

[58] M. Montemerlo, S. Thrun, "FastSLAM, A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics", Springer Tracts in Advanced Robotics, Vol. 27, 2007.

[59] C. Kim, R. Sakthivel, W.K. Chung, "Unscented FastSLAM: A Robust and Efficient Solution to the SLAM Problem", IEEE Transactions on Robotics, 24(4):808-820, 2008.

[60] J. Mullane, Ba-Ngu Vo, M. Adams, Ba-Tuong Vo, "A Random-Finite-Set Approach to Bayesian SLAM", IEEE Transactions on Robotics, 27(2):268-282, 2011.

[61] G. Sibley, L. Matthies, G. Sukhatme, “Sliding window filter with application to planetary landing”, Journal of Field Robotics, 27(5):587–608, 2010.

[62] K. Konolige, “Large-scale map-making,” the 19th national conference on Artifical Intelligence, 457–463, 2004.

[63] E. Olson, J. Leonard, S. Teller, “Fast iterative optimization of pose graphs with poor initial estimates”, IEEE ICRA, 2262–2269, 2006.

[64] G. Grisetti, R. Kümmerle, C. Stachniss, W. Burgard, "A Tutorial on Graph-Based SLAM", IEEE Intelligent Transportation Systems Magazine, 2(4):31-43, 2010.

[65] K. Konolige,G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, R. Vincent, "Efficient Sparse Pose Adjustment for 2D mapping", IEEE/RSJ International Conference on Intelligent Robots and Systems, 22-29, 2010.

[66] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, C. Hertzberg, "Hierarchical optimization on manifolds for online 2D and 3D mapping", IEEE International Conference on Robotics and Automation, 273-278, 2010.

[67] H. Kretzschmar, C. Stachniss, G. Grisetti, "Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders", IEEE/RSJ International Conference on Intelligent Robots and Systems, 865-871, 2011.

[68] F. Dellaert, J. Carlson, V. Ila, Kai Ni, C. Thorpe, "Subgraph-preconditioned conjugate gradients for large scale SLAM", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2566-2571, 2010.

[69] V. Ila, J.M. Porta, J. Andrade-Cetto, "Information-Based Compact Pose SLAM", IEEE Transactions on Robotics, 26(1): 78-93, 2010.

[70] A. Eliazar, R. Parr, "DP-SLAM 2.0", IEEE International Conference on Robotics and Automation, 2:1314-1320, 2004.

[71] Sharma K, Moon I, Kim SG, "Extraction of Visual Landmarks Using Improved Feature Matching Technique for Stereo Vision Applications", IETE Tech Rev 2012;29:473-81

[72] F. Fraundorfer, C. Engels, D. Nister, "Topological mapping, localization and navigation using image collections", IEEE/RSJ International Conference on Intelligent Robots and Systems, 3872-3877, 2007.

[73] G. Schindler, M. Brown, R. Szeliski, "City-Scale Location Recognition", IEEE Conference on Computer Vision and Pattern Recognition, 1-7, 2007.

[74] M. Milford, G. Wyeth, "Persistent Navigation and Mapping using a Biologically Inspired SLAM System", International Journal of Robotics. Research, 29(9):1131-1153, 2010.

[75] M. Cummins, P. Newman, "Accelerating FAB-MAP with concentration inequalities", IEEE Transactions on Robotics, 26(6):1042-1050, 2010.

[76] A. Angeli, D. Filliat, S. Doncieux, J.A. Meyer, "Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words", IEEE Transactions on Robotics, 24(5):1027-1037, 2008.

[77] T. Nicosevici, R. Garcia, "Automatic Visual Bag-of-Words for Online Robot Navigation and Mapping", IEEE Transactions on Robotics, 28(4):886-898, 2012.

[78] W. Maddern, M. Milford, G. Wyeth, "Towards persistent indoor appearance-based localization, mapping and navigation using CAT-Graph", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4224-4230, 2012.

[79] K. Granström, J. Callmer, F. Ramos, J. Nieto, "Learning to detect loop closure from range data", IEEE Int. Conf. on Robotics & Automation, 2009.

[80] M. Bosse, R. Zlot, "Place recognition using regional point descriptors for 3D mapping", O Bousquet and G Ratsch, eds Field and Service Robotics, Cambridge, MA, 195–204, 2010.

[81] S. Grzonka, A. Karwath, F. Dijoux, W. Burgard, "Activity-Based Estimation of Human Trajectories", IEEE Transactions on Robotics, 28(1):234-245, 2012.

[82] M. Angermann, P. Robertson, "FootSLAM: Pedestrian Simultaneous Localization and Mapping Without Exteroceptive Sensors—Hitchhiking on Human Perception and Cognition", Proceedings of the IEEE, 100(13):1840-1848, 2012.

[83] Shin, H.; Chon, Y.; Cha, H., "Unsupervised Construction of an Indoor Floor Plan Using a Smartphone", IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, PP(99):1-10, 2011.

[84] D.F. Wolf, G.S. Sukhatme, "Semantic Mapping Using Mobile Robots", IEEE Transactions on Robotics, 24(2):245-258, 2008.

[85] J.-S. Lee, C. Kim, W. K. Chung, "Robust RBPF-SLAM using sonar sensors in non-static environments", IEEE International Conference on Robotics and Automation, 250-256, 2010.

[86] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, H. Durrant-Whyte, "Simultaneous Localization, Mapping and Moving Object Tracking", The International Journal of Robotics Research, 26: 889-916, 2007.

[87] Guan D, Ma T, Yuan W, Lee YK, Jehad Sarkar A M., "Review of Sensor-based Activity Recognition Systems", IETE Tech Rev 2011, 28:418-33.

[88] J.M. Porta, "CuikSLAM: A Kinematics-based Approach to SLAM", ICRA, 2425-2431, 2005.

[89] J. Borenstein, H. R. Everett, L. Feng, "Where I am? Sensors and Methods for Mobile Robot Positioning", University of Michigan, 1996.

[90] J. C. Lagarias, J. A. Reeds, M. H. Wright, P. E. Wright, "Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions", SIAM J. Optim., 9(1):112–147, 1998.

[91] T. Gindele, S. Brechtel, J. Schröder, R. Dillmann, "Bayesian Occupancy grid Filter for dynamic environments using prior map knowledge", IEEE Intelligent Vehicles Symposium, 669- 676, 2009.

[92] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure", ACM Computing Surveys, 23(3):345-405, 1991.

[93] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", The International Journal of Robotics Research, 5: 90-98, 1986.

[94] H.J.S. Feder, J.J.E. Slotine, "Real-time path planning using harmonic potentials in dynamic environments", IEEE ICRA 1:874-881, 1997.

[95] R. Bellman, "Dynamic Programming", Princeton University Press, 1957.

[96] J. Cesarone, K. F. Eman, "Efficient manipulator collision avoidance by dynamic programming", Robotics and Computer-Integrated Manufacturing, 8(1):35-44, 1991.

[97] Dimitri P. Bertsekas, "Dynamic Programming: Deterministic and Stochastic Models". Prentice-Hall, Englewood Cliffs, NJ, 1987.

[98] A. Tahirovic, G. Magnani, "General Framework for Mobile Robot Navigation Using Passivity-Based MPC", IEEE Transactions on Automatic Control, 56(1):184-190, 2011.

[99] S. M. LaValle, "Planning Algorithms", Cambridge University Press, 2006.

[100] S. Russsell, P. Norvig, "Artificial Intelligence: a modern approach", 3rd Edition, Prentice-Hall, 2010.

[101] P. E. Hart, N.J. Nilsson, B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", IEEE Transactions on systems science and cybernetics, 4(2):100-107,1968.

[102] P. E. Hart, N.J. Nilsson, B. Raphael, Correction to "A formal basis for the heuristic determination of minimum cost paths", SIGART Newsletter, 37:28-29, 1972.

[103] E. W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, 1:279-271, 1959.

[104] R.E. Korf, "Linear-space best-first search", AIJ, 62(1): 41-78, 1993.

[105] P.P. Chakrabarti, S. Ghose, A. Acharya, S.C. de Sarkar, "Heuristic search in restricted memory", AIJ, 41(2),197-222, 1989.

[106] A. Stentz, "Optimal and efficient path planning for partially-known environments", IEEE International Conference on Robotics and Automation, 4:3310-3317,1994.

[107] S. Koenig, M. Likhachev, "Fast replanning for navigation in unknown terrain", IEEE Transactions on Robotics, 21(3):354-363, 2005.

[108] M. Likhachev, G. Gordon, S. Thrun, " ARA*: Anytime A* with Provable Bounds on Sub-Optimality", Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge, MA, 2004.

[109] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun, " Anytime Dynamic A*: An Anytime, Replanning Algorithm", Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2005.

[110] A.R. Willms, S.X. Yang, "An efficient dynamic system for real-time robot-path planning", IEEE Transactions on Systems, Man, and Cybernetics, 36(4):755-766, 2006.

[111] A.R. Willms, S.X. Yang, "Real-Time Robot Path Planning via a Distance-Propagating Dynamic System with Obstacle Clearance", IEEE Transactions on Systems, Man, and Cybernetics, 38(3):884-893, 2008.

[112] G.E. Jan, Ki Yin Chang, I. Parberry, "Optimal Path Planning for Mobile Robot Navigation", IEEE/ASME Transactions on Mechatronics, 13(4):451-460, 2008.

[113] C. Y. Lee, “An algorithm for path connection and its applications,” IRE Trans. Electron. Comput., vol. EC-10, pp. 346–365, Sep. 1961.

[114] D. Hsu, Randomized single-query motion planning in expansive spaces, PhD thesis, Department of Computer Science, Stanford University, 2000.

[115] S.M. LaValle, "Rapidly-Exploring Random Trees: Progress and Prospects", Algorithmic and computational robotics : new directions, 293-308, 2000.

[116] C. Canudas de Wit, H. Khennouf, C. Samson, O. Sordalen, "Nonlinear Control Design for Mobile Robots", Recent Developments in Mobile Robots, World Scientific,1993.

[117] C. Samson, K. Ait-Abderrahim, "Mobile robot control, part 1: Feedback control of a nonholonomic wheeled cart in cartesian space", Technical Report 1288, INRIA Sophia-Antipolis, 1990.

[118] G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem", IEEE Transactions on Robotics and Automation, 17(3):229-241, 2001.

[119] Erik Zamora, Wen Yu, Recent Advances on Simultaneous Localization and Mapping (SLAM) for Mobile Robots, IETE Technical Review, 30(6): 490-497, 2013.

[120] E. Scholte, M. E. Campbell, "A nonlinear set-membership filter for online applications", Int. J. Robust Nonlinear Control, 13:1337–1358, 2003.

[121] A.Garulli and A.Vicino, Set membership localization of mobile robots via angle measurements, IEEE Transactions on Robotics and Automation, Vol.17, No.4, 450-463, 2001.

[122] L. Jaulin, "A Nonlinear Set Membership Approach for the Localization and Map Building of Underwater Robots", IEEE Transactions on Robotics, 25(1): 88-98, 2009.

[123] L. Jaulin, "Range-Only SLAM With Occupancy Maps: A Set-Membership Approach", IEEE Transactions on Robotics, 27(5):1004-1010, 2011.

[124] M. Di Marco, A. Garulli, A. Giannitrapani, A. Vicino, "A Set Theoretic Approach to Dynamic Robot Localization and Mapping", Autonomous Robots, 16(1): 23-47, 2004.

[125] F.C. Schweppe, "Recursive state estimation: Unknown but bounded errors and system inputs", IEEE Transactions on Automatic Control, 13(1):22-28, 1968.

[126] F. C. Schweppe, Uncertain Dynamic Systems, Englewood Cliffs, NJ: Prentice-Hall, 1973.

[127] E. Fogel and Y. F. Huang, On the value of information in system identification: Bounded noise case, Automatica, Vol. 18, No.2, 229–238, 1982.

[128] E.Weyer, M.C. Campi, Non-asymptotic confidence ellipsoids for the least squares estimate, 39rd IEEE Conference on Decision and Control, Sydney, Australia, 2688-2693, 2000.

[129] L.Ros, A.Sabater and F.Thomas, An ellipsoid calculus based on propagation and fusion, IEEE Transactions on Systems, Man and Cybernetics, Vol.32, No.4, 430-442, 2002.

[130] M.V.Correa, L.A.Aguirre, and R.R.Saldanha, Using steady-state prior knowledge to constrain parameter estimates in nonlinear system identification, IEEE Transactions on Circuits and Systems, Part I, Vol.49, No.9, 1376-1381, 2002.

[131] R.G.Lorenz, and S.P.Boyd, Robust minimum variance beam-forming, IEEE Transactions on Signal Processing, Vol. 53, No.5, 1684-1696, 2005

[132] S.A.Nazin and B.T. Polyak. Limiting behavior of bounding ellipsoid for state estimation. Proceedings of the 5th IFAC Symposium on Nonlinear Control Systems. St.Petersburg, Russia, 585-589, 2001.

[133] W. Yu, J. de Jesus Rubio. "Recurrent Neural Networks Training With Stable Bounding Ellipsoid Algorithm," IEEE Transactions on Neural Networks, 20(6):983-991, 2009.

[134] J. Neira, J.D. Tardos, "Data association in stochastic mapping using the joint compatibility test", IEEE Transactions on Robotics and Automation, 17(6):890-897, 2001.

[135] S.B. Williams, "Efficient solutions to autonomous mapping and navigation problems", PhD Thesis, The University of Sidney, 2001.

[136] D. G. Maksarov & J. P. Norton, "State bounding with ellipsoidal set description of the uncertainty", International Journal of Control, 65(5):847-866, 1996.

[137] J.E. Guivant, E.M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation", IEEE Transactions on Robotics and Automation, 17(3):242-257, 2001.

[138] K-Team Corporation, (2014) "Koala Robot". Consulted in 24/11/2014 http://www.k-team.com/

[139] Natural Point Inc., (2014) "Cameras Flex 3". Consulted in 24/11/2014 http://www.naturalpoint.com/optitrack/products/flex-3/

[140] G. C. Goodwin and K. Sang Sin, Adaptive Filtering Prediction and Control, Prentice-Hall, Englewood Cliffs, NJ:07632, 1984.

[141] I. Chabini, S. Lan, Adaptation of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks, IEEE Transactions onIntelligent Transportation Systems, Vol.3, No.1, 60 -74, 2002.

[142] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments, International Journal of Robotics Research, Vol.29, 485-501, 2010.

[143] M. Pivtoraiko, R. A. Knepper, A. Kelly, Differentially constrained mobile robot motion planning in state lattices, Journal of Field Robotics, Special Issue: Special Issue on Space Robotics Part I, 26(3):308–333, 2009.

[144] J. Ren, K.A. McIsaac, R.V. Patel, Modified Newton's method applied to potential field-based navigation for mobile robots, IEEE Transactions on Robotics, 22(2):384-391, 2006.

[145] L. E. Kavraki, P. Švestka, J.-C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Transaction on Robotics and Automation, 12(4):566–580, 1996.

[146] S. M. LaValle, J. J. Kuffner, Randomized kinodynamic planning, The International Journal of Robotics Research, 20(5):378–400, 2001.

[147] S.Grzonka, G.Grisetti, W.Burgard, A Fully Autonomous Indoor Quadrotor, IEEE Transactions on Robotics, Vol.28, No.1, 90 -100, 2012.

[148] E.Marder-Eppstein, E.Berger, T.Foote, B.Gerkey, K.Konolige, The Office Marathon: Robust navigation in an indoor office environment, 2010 IEEE International Conference on Robotics and Automation (ICRA12), 300 -307, Anchorage, Alaska USA, 2010.

[149] C.Hernandez, X.Sun, S.Koenig and P. Meseguer, Tree Adaptive A*, International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS11), 123-130, Taipei, Taiwan, 2011.

[150] S.Koenig, C.Tovey, Y.Smirnov, Performance bounds for planning in unknown terrain, Artificial Intelligence, Vol.147, No.1-2, 253 - 279, 2003.

[151] S. Shen, N. Michael, V. Kumar, "Stochastic differential equation-based exploration algorithm for autonomous indoor 3D exploration with a micro-aerial vehicle", International Journal of Robotics Research, Vol.31, 1431-1444, 2012.

[152] D. Lang, etc., RoboCupRescue 2011 - Robot League Team, Technical Report, Universitat Koblenz-Landau, 2011.

[153] S.Koenig, Y.Smirnov, Sensor-Based Planning with the Freespace Assumption, International Conference on Robotics and Automation (ICRA97), 3540 - 3545, Albuquerque, New Mexico, USA, 1997.

[154] A. Naima, Long-term robot mapping in dynamic environments, PhD Thesis, MIT, 2011.

[155] M.Keidar, G.A.Kaminka, Efficient frontier detection for robot exploration, International Journal of Robotics Research, 2013.

[156] iRobot Corp. (2011) "iRobot Create". Consulted in 18/12/2014 http://store.irobot.com/education-research-robots/shop.jsp?categoryId=3311368

[157] Khoshelham K, Elberink SO, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications". Sensors. 2012; 12(2):1437-1454.

[158] ROS (2014) "Robot Operating System". Consulted in 18/12/2014 http://wiki.ros.org/

[159] ROS (2014) "Driver for iRobot Create". Consulted in 18/12/2014 http://wiki.ros.org/irobot_create_2_1

[160] ROS (2014) "Driver for Kinect". Consulted in 18/12/2014 http://wiki.ros.org/openni_launch

[161] ROS (2014) "Gmapping". Consulted in 18/12/2014 http://wiki.ros.org/gmapping

[162] Paul Viola, Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. Conference on Computer Vision and Pattern Recognition (CVPR), 2001: 511-518.

[163] Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. Submitted to ICIP 2002.

[164] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang and Stan Z. Li. Learning Multi-scale Block Local Binary Patterns for Face Recognition. International Conference on Biometrics (ICB), 2007: 828-837.

[165] Fischler, M A., Bolles, R., "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," in Proc. 1980 Image Understandtng Workshop (College Park, Md., Apr 1980), L. S. Baurnann, Ed, Science Applications, McLean, Va., 1980:71-8.