UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

GRAPHSLAM ALGORITHM IMPLEMENTATION FOR SOLVING SIMULTANEOUS LOCALIZATION AND MAPPING

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

FRANCO ANDREAS CUROTTO MOLINA

PROFESOR GUÍA:
MARTIN ADAMS

MIEMBROS DE LA COMISIÓN:
MARCOS EDUARDO ORCHARD CONCHA
JORGE FELIPE SILVA SÁNCHEZ

SANTIAGO DE CHILE
MARZO 2016

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FRANCO ANDREAS CUROTTO MOLINA
FECHA: MARZO 2016
PROF. GUÍA: SR. MARTIN ADAMS

GRAPHSLAM ALGORITHM IMPLEMENTATION FOR SOLVING SIMULTANEOUS LOCALIZATION AND MAPPING

SLAM (Simultaneous Localization and Mapping) es el problema de estimar la posición de un robot (u otro agente), y simultáneamente, generar un mapa de su entorno. Es considerado un concepto clave en la robótica móvil, y fundamental para alcanzar sistemas verdaderamente autónomos.

Entre las muchas soluciones que se han propuesto para resolver SLAM, los métodos basados en grafos han adquirido gran interés por parte de los investigadores en los últimos años. Estas soluciones presentan varias ventajas, como la habilidad de manejar grandes cantidades de datos, y conseguir la trayectoria completa del robot, en vez de solo la última posición. Una implementación particular de este método es el algoritmo GraphSLAM, presentado por primera vez por Thrun y Montemerlo en 2006.

En esta memoria, el algoritmo GraphSLAM es implementado para resolver el problema de SLAM en el caso de dos dimensiones. En objetivo principal de esta memoria es proveer de una solución de SLAM ampliamente aceptada para la realización de pruebas comparativas con nuevos algoritmos de SLAM. La implementación usa el framework g^{2}o como herramienta para la optimización de mínimos cuadrados no lineales.

La implementación de GraphSLAM es capaz de resolver SLAM con asociación de datos conocida y desconocida. Esto significa que, incluso cuando el robot no tiene conocimiento del origen de las mediciones, éste puede asociar las mediciones a los estados correspondientes, mediante el uso de estimación probabilística. El algoritmo también usa un método basado en kernel para la estimación robusta ante *outliers*. Para mejorar el tiempo de cómputo del algoritmo, varias estrategias fueron diseñadas para verificar las asociaciones y ejecutar el algoritmo de manera eficiente.

La implementación final se probó con datos simulados y reales, en el caso de asociación conocida y desconocida. El algoritmo fue exitoso en todas las pruebas, siendo capaz de estimar la trayectoria del robot y el mapa del entorno con un error pequeño. Las principales ventajas del algoritmo son su alta precisión, y su alto grado de configuración dado por la selección de parámetros. Las mayores desventajas son el tiempo de cómputo del algoritmo cuando la cantidad de datos es alta, y su incapacidad de eliminar falsos positivos.

Finalmente, como trabajo futuro, se sugieren modificaciones para aumentar la velocidad de convergencia, y para eliminar falsos positivos.

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FRANCO ANDREAS CUROTTO MOLINA
FECHA: MARZO 2016
PROF. GUÍA: SR. MARTIN ADAMS

GRAPHSLAM ALGORITHM IMPLEMENTATION FOR SOLVING SIMULTANEOUS LOCALIZATION AND MAPPING

SLAM (Simultaneous Localization and Mapping) is the problem of estimating a robot's (or other agent's) position and simultaneously generate a map of its environment. It is considered to be a core concept in mobile robotics, and a fundamental one to achieve truly autonomous systems.

Among the many solutions that have been proposed for solving SLAM, graph-based approaches have gained significant interest from researchers in the recent years. These solutions present various advantages, such as the capability to handle large amounts of data, and to retrieve the complete robot trajectory, rather than just the last position. A particular implementation of this approach is the GraphSLAM algorithm, first presented by Thrun and Montemerlo in 2006.

In this thesis, the GraphSLAM algorithm is implemented for solving the SLAM problem in a two dimensional scenario. The main objective of this work is to provide a widely accepted SLAM solution for making benchmark comparisons to newer SLAM algorithms. The implementation uses the g^{2}o framework as a tool for nonlinear least squares optimization.

The GraphSLAM implementation is able to solve SLAM with known and unknown data association. This means that even when the robot has no knowledge of the origin of measurements, it can associate measurements to corresponding states, by means of probabilistic estimation. The algorithm also uses a kernel-based method for robust estimation against outliers. In order to improve the algorithm's computation time, several strategies were designed to efficiently test the association between landmarks and run the optimizations.

The final implementation was tested with simulated and real data, in the case of known and unknown data association. It worked successfully in all the test cases, being able to estimate the robot path and the environment map with small error. The main advantages of the algorithm are the high accuracy and the high level of customization given by its parameters selection. The major drawbacks are the algorithm's computation time for large datasets, and the inability to remove false alarms.

Finally, as future work, modifications are suggested to increase convergence speed, and for dealing with false positives.

A mi familia

Acknowledgments

I would like to thank professor Martin Adams for giving me the opportunity to work in the thrilling and challenging field of robotics. This project has given me a very useful experience about the scientific investigation and work in general.

I would also like to thank my co-workers in the AMTC laboratory, especially Keith Leung and Felipe Inostroza. Your constant help, support, and review of my work allowed me to make a much better project than I could have achieved on my own. Also, thank you for providing me with sample codes and datasets for the project. It made my work much easier.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The study of robotic systems is fundamental to achieve the increasingly demanding goal of automating the processes that occur in every aspect of our lives. Autonomous systems are becoming more ubiquitous by the day. While historically they were first used in manufacturing companies and industrial processes, now they have found applications in areas such as farming, mining, transportation, security, medicine, household maintenance, space exploration, military uses, and much more.

In particular, mobile robotics is the study of a mechanical agent that can move in an environment. A robot's ability to change its pose makes mobile robots capable of doing a much wider range of tasks than stationary robots. However, their motion capability comes with an essential problem: as the robot moves, it must compute its new position in order to continue operating properly. In robotics, the problem of estimating the robot's current position is called *localization*. Sensors are used to gather information about the robot location, however, any kind of sensor is contaminated with noise, so the robot position cannot be retrieved with absolute certainty, but it must be estimated by means of probabilistic methods.

If the environment in which the robot is immersed is also unknown, it must be estimated alongside with the robot pose. The problem of estimating the robot's environment is called *mapping*. When both localization and mapping must be solved concurrently, the problem is called *Simultaneous Localization and Mapping* (SLAM). SLAM is considered to be the "Holy Grail" of mobile robotics, as knowing both, robot location and the environment, are crucial for every robot to work properly [2]. The subfield of robotics that studies the probabilistic method and algorithms to solve problems such as SLAM is usually called Probabilistic Robotics.

Currently, one of the most widely used algorithms to solve SLAM is GraphSLAM. GraphSLAM was developed by Thrun and Montemerlo [11], and is considered to perform better and have lower complexity than most filtering methods, such as the Extended Kalman Filter. GraphSLAM represents the necessary information regarding the robot and the map as nodes of a graph. The graph can then be converted to a special kind of matrix called a sparse matrix. The advantage of using this type of matrix is that there exist specialized algorithms that operate upon it, that are many times more efficient that the ones used on regular, dense, matrices.

1.1 General Objectives

The main objective of this work is to implement an offline version of the GraphSLAM algorithm for solving 2D SLAM[1]. The implementation should be able to handle known and unknown data association, and be robust to non-Gaussian noise and outliers. The g^2o (general graph optimization) framework[2] will be used as a non-linear least squares solver for the algorithm.

The contribution of this work is to provide a fully functional SLAM algorithm, which could be used for the navigation of robots in real world scenarios, and as a benchmark comparison for newer SLAM algorithms.

1.2 Specific Objectives

The objectives of this thesis are to:

1. Learn how to use the g^2o framework.
2. Implement a data association algorithm to handle landmarks of unknown correspondence.
3. Test the implementation with simulated and real data.

1.3 Document Structure

The remainder of the document is organized as follows. In Chapter 2, the basic concepts of Probabilistic Robotics and SLAM, as well as the theoretical framework of the GraphSLAM algorithm, are presented. In Chapter 3, the implemented GraphSLAM algorithm is explained in detail. In Chapter 4, the results of the implementations are shown for various scenarios, and a parameter analysis is made. Finally, in Chapter 5, the results are discussed, and possible future work is suggested.

[1] The repository of this work can be found in: `https://github.com/francocurotto/GraphSLAM`.
[2] `https://github.com/RainerKuemmerle/g2o`

Chapter 2

Background

This chapter presents the theoretical framework upon which this work was developed. It has the purpose of introducing readers unfamiliar with the topic of robotics and give a theoretical foundation for the work done.

Section 2.1 describes the basic concepts of stochastic state estimation in mobile robotics, needed to understand the rest of the work. Section 2.2 defines the problem of SLAM in a mathematical way. The GraphSLAM algorithm is presented and discussed in detail in Section 2.3. Finally, a brief review of the state of the art in SLAM is given in Section 2.4.

2.1 Basic Concepts in Probabilistic Robotics

2.1.1 Environment and State

Robotics is the science of perceiving and manipulating the physical world through an electro-mechanical device, which is called a *robot*. The robot is provided with actuators to interact with its soundings (such as wheels, or mechanical arms), and sensors to measure its environment (such as cameras, or laser sensors). In this context, the *environment* refers to the dynamical system with which the robot can interact (this includes the robot itself), and that is characterized by its *state*. The state is a mathematical description that can be represented as a collection of *state variables* that summarizes all the information of interest. State variables can contain information about the robot itself (for example, the robot position or velocity), or the robot operating environment (e.g., position of nearby objects). Furthermore, state variables can be *static* or *dynamic*. Dynamic state variables can change over time (like the position of people) while static variables remain constant (such as walls or trees). In this work, time is treated as a discrete variable, indexed by k.

2.1.2 Controls and Measurements

A robot can interact with the environment in two ways, it can influence the environment using its actuators, and it can gather information of the state through its sensors.

Usually, a robot's actuators are activated through a *control input.* These controls inputs could be given by a human using a controller device, or an algorithm implemented in a computer. It is useful to keep a record of all the control inputs that has been applied over time. A single control input at time k is denoted as $\boldsymbol{u}_k$, where the control $\boldsymbol{u}_k$ changes the state from time $k-1$ to k. The sequence of control data from time k_1 to k_2 is denoted as $\boldsymbol{u}_{k_1:k_2} = \boldsymbol{u}_{k_1}, \boldsymbol{u}_{k_1+1} \ldots, \boldsymbol{u}_{k_2}$.

Sensors are used to take measurements of nearby objects. As the robot acquire information of its surroundings, it can generate a *map* of the environment. Formally, a map $\boldsymbol{m}$ is a list of objects: $\boldsymbol{m} = [\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_j, \ldots, \boldsymbol{m}_N]$. Here N is the total number of objects in the environment, and each $\boldsymbol{m}_j$ is a vector of properties. In this work, it's assumed that the map is static, i.e., it doesn't change over time, hence $\boldsymbol{m}$ doesn't have a time index.

A *feature-based* map is used in this work. In feature-based maps, each element of the map correspond to a distinct object, called *landmark* or *feature*, with an unique set of properties. Typically, these properties are the position of the landmark, plus a distinct signature, such as the color of the landmark. Hence, in a 2 dimensional scenario, the vector $\boldsymbol{m}_j$ would look like:

$$\boldsymbol{m}_j = \begin{bmatrix} m_{j,x} \\ m_{j,y} \\ \boldsymbol{m}_{j,s} \end{bmatrix} \tag{2.1}$$

Being $m_{j,x}$ and $m_{j,y}$, the horizontal and vertical position of the landmark j respectively, and $\boldsymbol{m}_{j,s}$ its signature vector.

It is assumed that each sensor produces at most one measurement per landmark at every instance of time. The *set of measurements* produced at time k is denoted as $\boldsymbol{z}_k$. To distinguish each landmark measured, $\boldsymbol{z}_k^i$ denotes the i-th landmark detected at time k. Note that the measurement $\boldsymbol{z}_k^i$ is a vector, since several properties can be sensed from a single landmark (e.g., distance and relative angle from the robot). The list of measurements taken from time k_1 to k_2 is denoted as $\boldsymbol{z}_{k_1:k_2}$.

2.1.3 Motion and Measurement Models

To tackle any problem in robotics, the mathematical models that describe the behavior of the robot must be defined. The *pose* of the robot defines the state variables that depend only on the robot (usually robot's position and orientation). The pose at time k is denoted as $\boldsymbol{x}_k$.

The robot motion model describes how the control input changes the robot pose from one timestep to the next. This model is called the forward kinematics equations of the robot.

In the core of the probabilistic robotics, it is the assumption that one cannot have a deterministic description of the world, that is, there is always an amount of uncertainty in it. Therefore, it is advantageous to use probabilistic models that take into account this uncertainty. The simplest probabilistic assumption is that models are contaminated with zero-mean white Gaussian noise. Using this probabilistic approach, a generic motion model is:

$$\boldsymbol{x}_k = \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1}) + \boldsymbol{\delta}_k \tag{2.2}$$

Where $\boldsymbol{g}$ is a deterministic function that describes the robot's kinematics. $\boldsymbol{u}_k$ is the control input that change the pose form $\boldsymbol{x}_{k-1}$ to $\boldsymbol{x}_k$. And $\boldsymbol{\delta}_k$ is a multivariate random Gaussian variable, with zero mean and covariance matrix $\boldsymbol{R}_k$ ($\boldsymbol{\delta}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k)$).

Since the motion model involves the addition of a Gaussian random variable, the model itself can be seen as a Gaussian random process, which represents the probability of the robot to end up in pose $\boldsymbol{x}_k$, given previous pose $\boldsymbol{x}_{k-1}$ and control input $\boldsymbol{u}_k$:

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = \det(2\pi\boldsymbol{R}_k)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1}))^T \boldsymbol{R}_k^{-1}(\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1}))\right\} \tag{2.3}$$

Most robots incorporate an internal sensor to measure the change of position between two timesteps. This way it can be verified if the control input actually produced the desired result. An example of this kind of sensors are the rotary encoders found in robot's wheels to count the number of turns it have made. The method of estimating the robot motion using these sensors is called *odometry*.

Similarly, the measurement model depicts how the robot obtains information of the environment. In other words, it describes mathematically the acquisition of measurements $\boldsymbol{z}_k$. Just as the motion model, independent Gaussian noise is assumed for the measurements. A generic measurement model is:

$$\boldsymbol{z}_k^i = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_j, i) + \boldsymbol{\varepsilon}_k^i \tag{2.4}$$

Where $\boldsymbol{m}_j$ represents the i-th measured landmark in measurement $\boldsymbol{z}_k^i$, and $\boldsymbol{x}_k$ the pose in which the measurements were made. $\boldsymbol{\varepsilon}$ is an Gaussian random variable, with zero mean and $\boldsymbol{Q}_k$ covariance matrix ($\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k)$).

Again, this model can be represented as a random process and is given by:

$$p(\boldsymbol{z}_k^i|\boldsymbol{x}_k, \boldsymbol{m}) = \det(2\pi\boldsymbol{Q}_k)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{z}_k^i - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_j, i))^T \boldsymbol{Q}_k^{-1}(\boldsymbol{z}_k^i - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_j, i))\right\} \tag{2.5}$$

In principle, there is no error-free method to associate the detection number i with the landmark number j. A possible solution is to assume that the association is automatically given by a special function $j = c_k^i$, called the *correspondence function*. The function c_k^i indicates deterministically which feature j correspond to each detection i, at every time k.

2.1.4 Localization and Mapping

Two of the main problems of interest in the field of probabilistic robotics are *localization* and *mapping*. In essence, these two problems correspond to the estimation of a particular subset of the state of the environment, given another subset of the same state. In this scenario, the state variables can be partitioned into the robot's internal state, given by the pose $\boldsymbol{x}_k$, and external states that can be measured by its sensor, given by the map $\boldsymbol{m}$.

In the localization problem, it is assumed that the map is known with absolute certainty, while the robot pose is unknown. The problem is then to estimate the robot pose over time as it moves around the environment and takes measurements from the map. Aside from the map, control inputs and measurements are available. Mathematically, the problem is to calculate the Probability Density Function (PDF):

$$p(\boldsymbol{x}_k|\boldsymbol{m}, \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) \tag{2.6}$$

Conversely, in the mapping problem, it is assumed that the location of the robot is known, and it is necessary to estimate the map (location and signature of landmarks). Again, all robot measurement are available. Note that robot control inputs are not needed, as they only affect the robot pose, which is already known. Mathematically, it is necessary to determine the PDF:

$$p(\boldsymbol{m}|\boldsymbol{x}_{0:k}, \boldsymbol{z}_{1:k}) \tag{2.7}$$

Where $\boldsymbol{x}_{0:k}$ denote the sequence of robot poses from time 0 to k, also known as the robot's *trajectory* or *path*. These two problems are just a particular case of an estimation problem, and classical filtering techniques, such as Extended Kalman Filter (EKF), have been applied successfully to solve them [10].

2.2 Description of the SLAM Problem

Both localization and mapping problem have an important limitation, they both assume a complete knowledge about a subset of the state of the environment: the robot pose and the map respectively. In many practical problems, there is no absolute knowledge of any of the state variables, and all the information regarding the state must be derived only from the control input and the measurements. In this case, localization and mapping have to be solved concurrently. In robotics, this problem is called Simultaneous Localization and Mapping (SLAM).

Mathematically SLAM can be stated as the determination of the PDF:

$$p(\boldsymbol{x}_k, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) \tag{2.8}$$

Note that both, robot's pose and the map are estimated given the control input and the measurements. Also, note that according to (2.8), only the current pose of the robot is being estimated. This is called the *online SLAM* problem. In some applications is useful to estimate the whole robot trajectory. In this case the SLAM problem is stated sightly differently:

$$p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) \tag{2.9}$$

This case is called the *full SLAM* problem. In (2.9) all the robot poses up to time k are being estimated.

It can be proven that the online SLAM is equivalent to the full SLAM after "marginalizing" the previous pose variables:

$$p(\boldsymbol{x}_k, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \int \int \dots \int p(\boldsymbol{x}_{1:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) d\boldsymbol{x}_1 d\boldsymbol{x}_2 \dots d\boldsymbol{x}_{k-1} \tag{2.10}$$

2.2.1 Correspondence Problem in SLAM

Until now is being assumed that when the robot takes a set measurement $\boldsymbol{z}_k$, it can successfully associate the i-th measurement with the corresponding feature detected j, by means of the correspondence function c_k^i. This is called the correspondence problem, and in reality is a nontrivial one. This is because, in the presence of noise, measurements may be incorrectly associated.

The correspondence problem is of great importance in SLAM. A single wrong association of landmarks could lead to divergences in the estimation. Also, once the wrong association is done, it may be impossible to recover from the mistake if the algorithm is not robust enough, or if the necessary information is no longer available.

In special cases, like simulations and when features can be distinguished correctly by measurements, it can be assumed that data association is known, i.e., one has access to c_k^i.

Most commonly, data association is not given and must be estimated by the algorithm. There exist different techniques to deal with this problem, one of the most popular is *maximum likelihood correspondence*.

This work deals with both cases, in which the correspondence is known, and when it is unknown.

2.3 The GraphSLAM Algorithm

GraphSLAM is an algorithm for solving SLAM. It was first presented in [11]. It transforms the SLAM posterior (2.9) in a graphical network that represents the likelihood of the data. It then transforms the graph into a least square minimization problem, that can be solved with conventional optimization techniques.

SLAM Representation in Graphs

A graph is a mathematical concept that is composed of two types of entities, *nodes* and *edges*. Nodes are abstract entities that are uniquely identified by some symbol (for example, a letter or a number), they are usually represented as a circle in a diagram. An edge is a pair of two different nodes that correspond to a connection between those nodes, in a diagram is represented as a line linking the nodes. The set of nodes in a graph is denoted as $\mathcal{V} = \{i : i \text{ is a node}\}$, and the set of edges as $\mathcal{E} = \{\langle i, j\rangle : \text{node } i \text{ is connected to node } j\}$. Figure 2.1 shows an example of a graph.

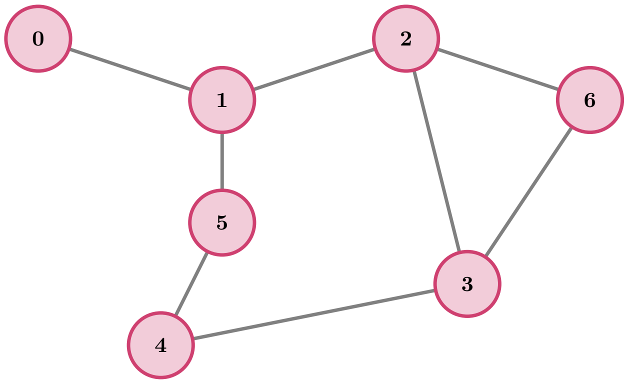


Figure 2.1: An example of a graph, with 6 nodes and 8 edges.

In the GraphSLAM context, nodes represent specifics state variables of the environment, and edges represent information. Nodes could represent two types of state variables: it could be either the pose of the robot $\boldsymbol{x}_k$ at a certain time k, or the position of a landmark $\boldsymbol{m}_j$. There is also two types of edges in the graph, the first ones are edges connecting two consecutive robot poses $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$, and these correspond to the translation that the robot realizes between k and $k+1$ produced by the control input $\boldsymbol{u}_{k+1}$. The second ones are edges connecting the robot pose $\boldsymbol{x}_k$ and the landmark $\boldsymbol{m}_j$ sensed in the measurement $\boldsymbol{z}_k^i$. Figure 2.2 illustrates the graph generated by a robot, as it moves on a map and take measurements of landmarks.

In the graph, the set of all nodes actually constitute all the state variables, and the edges accumulate all the information generated by the robot actions (motions and measurements).

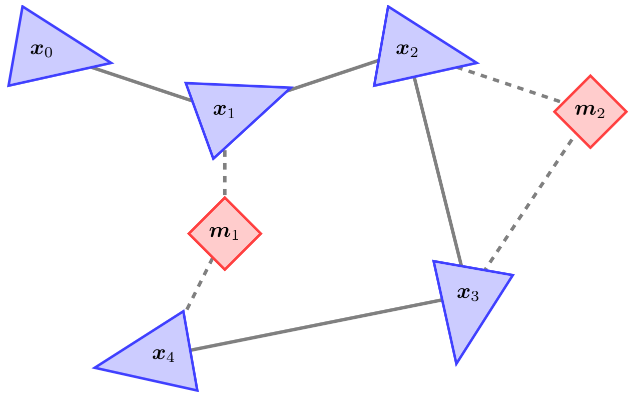


Figure 2.2: GraphSLAM illustration in 2D equivalent to Figure 2.1. The blue triangles are robot poses, and the red diamonds are landmarks positions. The solid lines represents the robot motion and the dashed lines the robot measurements.

Quadratic Form of SLAM Posterior

In SLAM, one usually wants to find a mathematical expression for the posterior probability (2.9), and then find the state that is more consistent with the data.

In order to make the problem tractable, several assumption must be made about the random behavior of the state variables. The must important of these assumptions is that the states correspond to a *Markov process*. In Markov processes, future states are conditionally independent of past states, given the current state.

Using Bayes theorem and the Markov assumption, the posterior (2.9) can be rewritten as:

$$\begin{aligned} p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) &= \eta \, p(\boldsymbol{z}_k | \boldsymbol{x}_{0:k}, \boldsymbol{m}, \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \; p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \qquad (2.11) \\ &= \eta \, p(\boldsymbol{z}_k | \boldsymbol{x}_k, \boldsymbol{m}) \; p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \qquad (2.12) \end{aligned}$$

Where in (2.11) the Bayes theorem is applied for the measurements $\boldsymbol{z}_k$, η is the normalizing factor. In (2.12) it is assumed that current measurements $\boldsymbol{z}_k$ are conditionally independent of past measurements, poses and control inputs, given the current pose $\boldsymbol{x}_k$ and map $\boldsymbol{m}$.

The last term of (2.12) can also be expanded using the definition of conditional probability:

$$\begin{aligned} p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) &= p(\boldsymbol{x}_k | \boldsymbol{x}_{0:k-1}, \boldsymbol{m}, \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \; p(\boldsymbol{x}_{0:k-1}, \boldsymbol{m} | \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \qquad (2.13) \\ &= p(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \; p(\boldsymbol{x}_{0:k-1}, \boldsymbol{m} | \boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \qquad (2.14) \end{aligned}$$

Where again the Markov assumption is applied, this time, to pose $\boldsymbol{x}_k$.

Substituting (2.14) into (2.12) gives:

$$p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \eta\, p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m})\; p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\; p(\boldsymbol{x}_{0:k-1}, \boldsymbol{m}|\boldsymbol{z}_{1:k-1}, \boldsymbol{u}_{1:k}) \tag{2.15}$$

Note that equation (2.15) simply states that the likelihood of the state at time k is proportional to the same likelihood at time $k-1$, multiplied by the motion and measurements probabilities. Applying (2.15) recursively yields:

$$p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \eta\, p(\boldsymbol{x}_0) \prod_k p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)\; p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) \tag{2.16}$$

Where $p(\boldsymbol{x}_0)$ is the initial knowledge of the robot pose. The final assumption to be made is that every individual measurement $\boldsymbol{z}_k^i$ is conditionally independent between each other, given the pose and map, i.e., $p(\boldsymbol{z}_k^i, \boldsymbol{z}_k^j|\boldsymbol{x}_k, \boldsymbol{m}) = p(\boldsymbol{z}_k^i|\boldsymbol{x}_k, \boldsymbol{m})\; p(\boldsymbol{z}_k^j|\boldsymbol{x}_k, \boldsymbol{m})$. Then, the final form of the posterior is:

$$p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \eta\, p(\boldsymbol{x}_0) \prod_k \left[p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \prod_i p(\boldsymbol{z}_k^i|\boldsymbol{x}_k, \boldsymbol{m}) \right] \tag{2.17}$$

For mathematical proposes, it is convenient to work with the negative log-likelihood of the posterior:

$$\begin{aligned} &-\log(p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \\ &\qquad -c - \log(p(\boldsymbol{x}_0)) - \sum_k \left[\log(p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)) \sum_i \log(p(\boldsymbol{z}_k^i|\boldsymbol{x}_k, \boldsymbol{m})) \right] \end{aligned} \tag{2.18}$$

Where $c = \log(\eta)$. An expression is given for $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$ in (2.3), and for $p(\boldsymbol{z}_k^i|\boldsymbol{x}_k, \boldsymbol{m})$ in (2.5). For the initial belief, as usual, it is assumed a zero-mean Gaussian distribution with covariance $\boldsymbol{\Omega}_0^{-1}$ ($p(\boldsymbol{x}_0) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Omega}_0^{-1})$). In virtue of the assumption of independent Gaussian noise, replacing all these expression into (2.18), gives a quadratic form for the negative log-SLAM posterior:

$$\begin{aligned} &-\log(p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \\ &\qquad c + \boldsymbol{x}_0^T \boldsymbol{\Omega}_0 \boldsymbol{x}_0 + \\ &\qquad \sum_k (\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1}))^T \boldsymbol{R}_k^{-1} (\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1})) + \\ &\qquad \sum_k \sum_i (\boldsymbol{z}_k^i - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_i))^T \boldsymbol{Q}_k^{-1} (\boldsymbol{z}_k^i - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_i)) \end{aligned} \tag{2.19}$$

Notice that every term of the sum in (2.19) has an associated edge in the graph representation (see Figure 2.2).

Notation Simplification

For notation simplification, the state vector $\boldsymbol{y}$ that contains the variables of all the poses over time, and all the landmarks, is defined:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_k \\ \boldsymbol{m} \end{bmatrix} \tag{2.20}$$

Furthermore every term in (2.19) can be encapsulated into a single notation called the *error function*: $\boldsymbol{e}_{ij}(\boldsymbol{y})$. Every index i, j in the error function corresponds to a node in the graph, that is, a robot pose or a landmark position. Then $\boldsymbol{e}_{ij}(\boldsymbol{y})$ is given by either by $\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1})$, if both indexes correspond to consecutive poses, by $\boldsymbol{z}_k^i - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_i)$, if indexes correspond to one pose and one landmark, or $\boldsymbol{x}_0$ if $i = j = 0$. The error function can be seen as difference between the expected and actual odometry or measurement.

Similarly, the information matrix $\boldsymbol{\Omega}_{ij}$ between nodes i and j can be defined as $\boldsymbol{R}_k^{-1}$, $\boldsymbol{Q}_k^{-1}$, or $\boldsymbol{\Omega}_0$ given by the same conditions as above. Given this,(2.19) can be written as:

$$F(\boldsymbol{y}) := -\log(p(\boldsymbol{y}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \sum_{\langle i,j\rangle \in \mathcal{E}} \boldsymbol{e}_{ij}(\boldsymbol{y})^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij}(\boldsymbol{y}) \tag{2.21}$$

Where the constant c is removed, as it is not state dependent. Finally, the most probable state for the map and the poses is determined by solving the following minimization problem:

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} F(\boldsymbol{y}) \tag{2.22}$$

Taylor Expansion

The terms in (2.19) are quadratic in the functions $\boldsymbol{g}$ and $\boldsymbol{h}$, which are usually nonlinear functions. Having nonlinear dependency of F over $\boldsymbol{y}$ makes (2.22) difficult to solve. A way to simplify the problem is to linearize those functions using a first order Taylor approximation over $\boldsymbol{e}_{ij}$:

$$\boldsymbol{e}_{ij}(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y}) \approx \boldsymbol{e}_{ij}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{ij} \boldsymbol{\Delta y} \tag{2.23}$$

Where $\breve{\boldsymbol{y}}$ an initial estimate of the state, $\boldsymbol{J}_{ij}$ is the Jacobian of $\boldsymbol{e}_{ij}$ computed at $\breve{\boldsymbol{y}}$, and $\boldsymbol{\Delta y}$ is a small increment around $\boldsymbol{y}$. Then, a local approximation of F can be obtained:

$$\begin{aligned}
F(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y}) &= \sum_{\langle i,j\rangle \in \mathcal{E}} \boldsymbol{e}_{ij}(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y})^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij}(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y}) \\
&\approx \sum_{\langle i,j\rangle \in \mathcal{E}} (\boldsymbol{e}_{ij}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{ij}\boldsymbol{\Delta y})^T \boldsymbol{\Omega}_{ij} (\boldsymbol{e}_{ij}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{ij}\boldsymbol{\Delta y}) \\
&= \sum_{\langle i,j\rangle \in \mathcal{E}} \underbrace{\boldsymbol{e}_{ij}(\breve{\boldsymbol{y}})^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij}(\breve{\boldsymbol{y}})}_{:=k_{ij}} + 2\underbrace{\boldsymbol{e}_{ij}(\breve{\boldsymbol{y}})^T \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}}_{:=\boldsymbol{b}_{ij}} \boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \underbrace{\boldsymbol{J}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}}_{:=\boldsymbol{H}_{ij}} \boldsymbol{\Delta y} \\
&= \sum_{\langle i,j\rangle \in \mathcal{E}} k_{ij} + 2\boldsymbol{b}_{ij}\boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \boldsymbol{H}_{ij} \boldsymbol{\Delta y} \\
&= k + 2\boldsymbol{b}\boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \boldsymbol{H} \boldsymbol{\Delta y} \qquad (2.24)
\end{aligned}$$

Where $\sum k_{ij} = k$, $\sum \boldsymbol{b}_{ij} = \boldsymbol{b}$ and $\sum \boldsymbol{H}_{ij} = \boldsymbol{H}$. $\boldsymbol{H}$ and $\boldsymbol{b}$ are called the information matrix and the information vector of the linearized system, respectively. The expression (2.24) can be minimized in $\boldsymbol{\Delta y}$ by solving the linear system:

$$\boldsymbol{H}\boldsymbol{\Delta y}^* = -\boldsymbol{b} \qquad (2.25)$$

Then the solution of the original problem is obtained by adding the increment obtained in the linear system with the initial guess:

$$\boldsymbol{y}^* = \breve{\boldsymbol{y}} + \boldsymbol{\Delta y}^* \qquad (2.26)$$

The popular Gauss-Newton algorithm iterates several times between the steps of linearizing the system in (2.24), solving the linear system in (2.25), and updating the state in (2.26).

Structure of the Linearized System

Of all steps of the Gauss-Newton algorithm, the linear system solving in (2.25) is the most computationally expensive, because it involves a matrix inversion (in practice more efficient methods are used, such as QR decomposition), and it's also the more prone to numerical errors. In some cases, the number of landmarks and the path of the robot is so large, that the inversion of $\boldsymbol{H}$ becomes intractable. It'll be shown actually, that the information matrix $\boldsymbol{H}$ has and underlying structure that makes the solving of the system (2.25), more efficient and precise.

Looking at $\boldsymbol{e}_{ij}(\boldsymbol{y})$ it can be seen that the only variables that are present in the function are those involves in nodes i and j. This means that the Jacobian $\boldsymbol{J}_{ij}$ of $\boldsymbol{e}_{ij}$ has an special structure formed by 2 blocks, and the rest is zero:

$$\boldsymbol{J}_{ij} = \left[\underbrace{\mathbf{0} \ldots \mathbf{0} \underbrace{\boldsymbol{A}_{ij}}_{i} \mathbf{0} \ldots \mathbf{0} \underbrace{\boldsymbol{B}_{ij}}_{j} \mathbf{0} \ldots \mathbf{0}}_{\boldsymbol{y}} \right] \tag{2.27}$$

Where $\boldsymbol{A}_{ij}$ and $\boldsymbol{B}_{ij}$ are the matrices of the derivatives of the functions in $\boldsymbol{e}_{ij}$ with respect the variables in nodes i and j respectively. This makes the matrix $\boldsymbol{H}_{ij}$ to be formed of four blocks, and the rest filled with zeros:

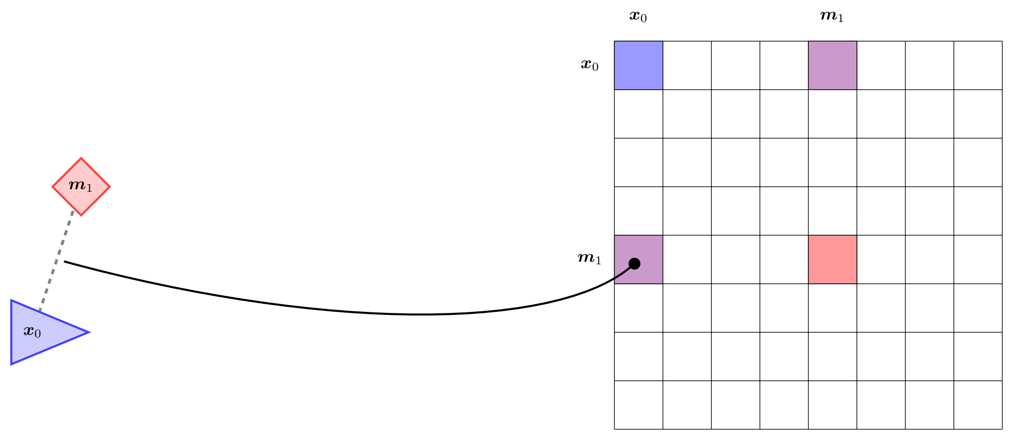
$$\boldsymbol{H}_{ij} = \begin{bmatrix} \ddots & & & & \\ & \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} & \ldots & \boldsymbol{A}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij} & \\ & \vdots & & \vdots & \\ & \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{A}_{ij} & \ldots & \boldsymbol{B}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{B}_{ij} & \\ & & & & \ddots \end{bmatrix} \tag{2.28}$$

Where all the zero entries are omitted. The off-diagonal blocks represent the relative information given by the edge $\langle i, j \rangle$. Therefore, every term $\boldsymbol{H}_{ij}$ adds 4 blocks to the information matrix $\boldsymbol{H}$.

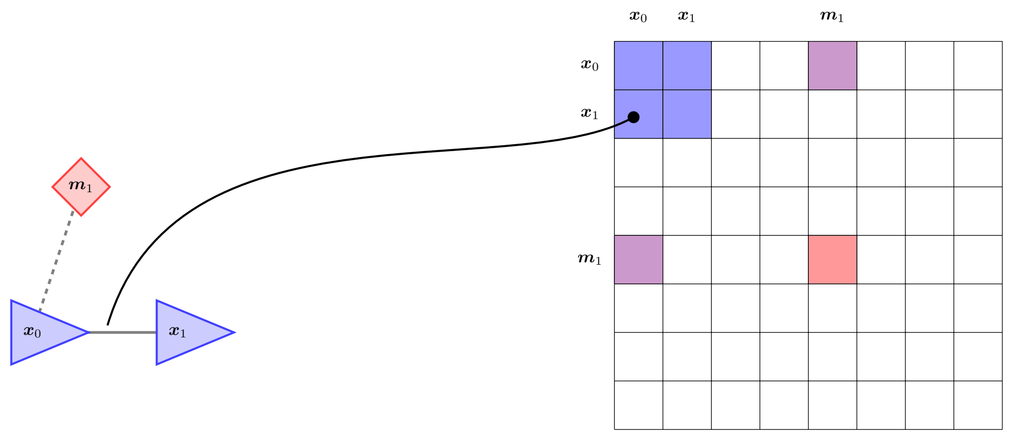
However, for the SLAM problem, not every combination of blocks (i.e. edges) is possible. This fact is represented in Figure 2.3. This figure shows how the information matrix get filled as the robot moves in the environment and takes measurements. Every colored square corresponds to a block from (2.28).

The information matrix can be divided into four submatrices. The top-left matrix (filled with blue squares) has the relative information between to pair of poses. Since the information acquired is only relative to two consecutive poses, this submatrix has a band diagonal structure, leaving all the other elements in zero. The top-right and bottom-left submatrices are equivalent and contain the relative information between a pose and a measured landmark. The number of blocks in these matrices depends on the number of landmarks measured, but is often in SLAM that only spatially close landmarks are measured at every pose, leaving these matrices, again, with much of their elements in zero. Finally, the bottom-right submatrix regards information about a pair of landmarks. Since there are not direct measurement between two landmarks, this submatrix is left as a block diagonal matrix.

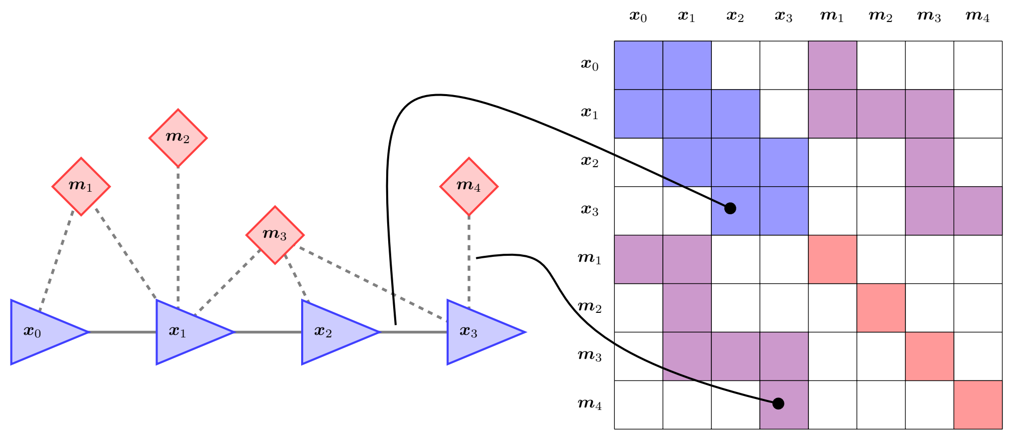
Therefore, it is shown that the number of non-zero elements in $\boldsymbol{H}$ is proportional to the number of edges in the graph, leading to a matrix with a large number of zero elements. These kinds of matrices are called *sparse matrices*. Fast and efficient algorithm exist to solve (2.25) for sparse matrices, such as Cholesky decomposition and Preconditioned Conjugate Gradient (PCG). These algorithms are already implemented in the g^{2}o framework that will be used.



(a) Block addition to the information matrix $\boldsymbol{H}$ after a landmark measurement.



(b) Block addition to the information matrix $\boldsymbol{H}$ after a robot movement.



(c) Shows the block structure of the information matrix $\boldsymbol{H}$ after 3 robot movements and 7 measurements.

Figure 2.3: Progressing filling of the information matrix $\boldsymbol{H}$ as the robot moves and takes measurements. Blue: pose-pose block. Purple: pose-landmark block. Red: landmark-landmark block.

2.4 Review of the State of the Art in SLAM

SLAM as a probabilistic problem has its beginning in the 1986 IEEE Robotics and Automation Conference held in San Francisco, California [3], [9]. One of the first papers to give a solution to SLAM was [9] using the Extended Kalman Filter technique. This solution is now known as EKF SLAM.

One of the first papers to state SLAM as an optimization over a graph is [8] by Lu and Milios. They were the first to treat motion and measurements in an equal manner as information constraints, which differed from standard EKF technique where motion information is used for prediction, and measurement as correction for the Kalman filter.

The GraphSLAM algorithm, as stated in this work, was first presented in [11] by Thrun and Montemerlo. In this paper, the SLAM negative likelihood posterior is derived for solving the full SLAM problem, and the optimization problem is stated in a similar way as in section 2.3. However, it does not specify how to solve the optimization in (2.22) from subsection Notation Simplification.

From there, several improvements have been made to the general idea of the GraphSLAM algorithm. In [1] a similar graph representation is used to solve SLAM, called $\sqrt{\text{SAM}}$ (Square Root Smoothing and Mapping). In this context, *smoothing* corresponds to estimating the entire robot trajectory up to the current time. $\sqrt{\text{SAM}}$ uses the measurement Jacobian instead of the information matrix to represent the system uncertainty. Although they are equivalent in terms of information, they have different computational properties. $\sqrt{\text{SAM}}$ can be used as a batch or an incremental algorithm for solving SLAM, and use either QR or Cholesky factorization of the linear equation, with variable ordering for complexity improvement.

An updated version of $\sqrt{\text{SAM}}$, called iSAM, is presented in [5]. It improves the performance in the incremental version, by directly updating the square root information matrix with new measurements as they arrive. It also avoids unnecessary fill-in in the information matrix generated by trajectory loop, by doing periodic variable reordering. It solves online data association using maximum likelihood and nearest neighbor matching.

Another improvement of iSAM is presented in [4], iSAM2, in which they define a new data structure, called the *Bayes Tree*, that allows them to improve the performance in the online case, by updating only the necessary variables in the linearization point after the arrival of new data.

In [7] a robust consistency-based loop-closure verification method, Realizing Reversing and Recovering (RRR) algorithm, is presented for the detection and correction of incorrect loop closures generated by the SLAM algorithm. Incorrect loop closures appear when a robot visits similar looking locations, which can severely corrupt the map estimate.

For efficiently solving graph optimization problems that appear in SLAM and other situations, a framework called g^{2}o (general graph optimization) is presented in [6]. g^{2}o is an open source optimization tool written in `C++`. It was designed to be easily extensible to a wide range of problems, and it contains typical optimization techniques used for sparse matrices, such as Cholesky decomposition and Preconditioned Conjugate Gradient.

Chapter 3

Methodology and Implementation

In this chapter, the implementation of the GraphSLAM algorithm is presented and explained. The algorithm is capable of solving the SLAM problem in an offline manner for a 2D scenario, in the cases of known and unknown data association.

The g^2o framework used in this work provides of a least squares solver for the optimization of equation (2.22), as well as a protocol to define the graph of the SLAM problem. g^2o is well optimized and it has several options for the solver, so the known data association version of the GraphSLAM algorithm is relatively straightforward to implement.

However, g^2o does not provide a way to handle unknown data association, so the main goal of this work is to implement a method for solving the correspondence problem in an efficient manner.

3.1 The g^2o Protocol

The first step in implementing the GraphSLAM algorithm is to define a protocol to store and interpret the data on a graph. g^2o already provides such a protocol. The stored data are of two types: data from nodes, and data from edges.

In the SLAM context, nodes themselves can be of two types, pose nodes and landmark nodes. Pose nodes represent the pose of the robot. In the 2D case, they consist of 3 variables: robot's x and y position, and robot's orientation θ. In g^2o a pose node is denoted with the keyword `VERTEX_SE2`[1]. Landmark nodes represent the 2D position (x and y) of a landmark. They are denoted with the keyword `VERTEX_XY`.

Edges represent either robot's odometry (data of robot's change in position), or robot's measurements of landmarks. Odometry is measured as the difference between robot's pose at two consecutive timesteps: $(\Delta x, \Delta y, \Delta \theta)$. On the other hand, robot measurements are given

[1]Vertex is synonymous of node. SE2 is the Non-Euclidean space that consists of two spatial dimensions and an angular dimension.

as the x and y distance to the landmark relative to the robot reference frame. Figure 3.1 illustrates the odometry and measurement of a robot. Keywords `EDGE_SE2` and `EDGE_SE2_XY` are use to denote odometry and measurement edges respectively.

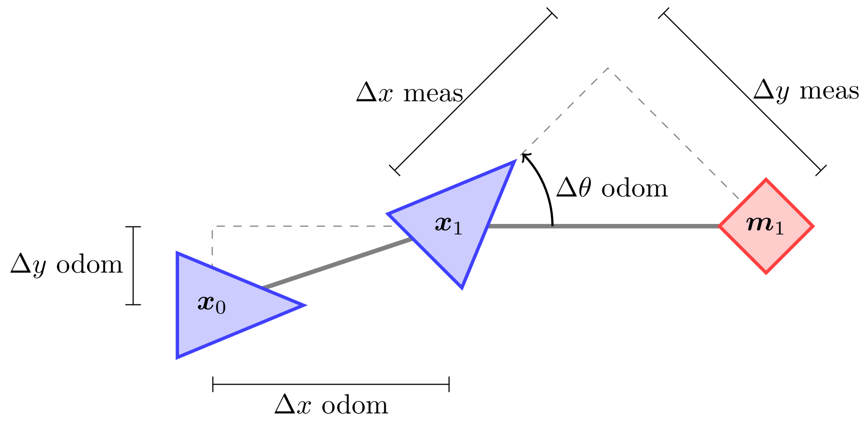


Figure 3.1: Illustration of odometry and measurement values in g^{2}o.

For GraphSLAM to work correctly, one must also provide to the algorithm the uncertainty of odometry and measurements. These correspond to the covariance matrices $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ from motion (2.2) and measurement (2.4) models respectively. g^{2}o works with the inverse of the covariance matrix, known as the information matrix. Nevertheless, these representations are equivalent in terms of the knowledge of the system. Since covariance matrices (and their inverse) are symmetric, only the upper diagonal block is needed. In this work, it is assumed that the model uncertainties are time independent, i.e., all nodes have the same values for the information matrix. The notation of each element of the matrices is given by:

$$\boldsymbol{R}_k^{-1} = \begin{pmatrix} ipxx & ipxy & ipxa \\ ipxy & ipyy & ipya \\ ipxa & ipya & ipaa \end{pmatrix} \quad \boldsymbol{Q}_k^{-1} = \begin{pmatrix} ilxx & ilxy \\ ilxy & ilyy \end{pmatrix} \tag{3.1}$$

Where each element of the information matrices can be obtained by inverting the covariance matrix of the corresponding model.

Finally, nodes must be indexed so they can be distinguishable from one another. This is indicated by an integer `id`. Ids are used in edges to indicate which two nodes the edge is connecting.

Table 3.1 summarizes g^{2}o notation to represent nodes and edges.

In this work, it is assumed that the first pose is known with absolute certainty, and is fixed to $(0, 0, 0)$. In g^{2}o, this is done by the command `FIX id`, where the id of the first pose is used. To use the data in the framework, it can be written in plain text, which is then uploaded to g^{2}o.

The code below presents a simple example of the g^{2}o protocol, its graphical equivalence is shown in Figure 3.2. In practice, when solving SLAM, only odometry and measurements

Graph element	Notation
Pose node	`VERTEX_SE2 id x y a`
Landmark node	`VERTEX_XY id x y`
Odometry edge	`EDGE_SE2 id1 id2 dx dy da ipxx ipxy ipxa ipyy ipya ipaa`
Measurement edge	`EDGE_SE2_XY id1 id2 dx dy ilxx ilxy ilyy`

Table 3.1: g^{2}o protocol for node and edge definition. `a`: angle.

are known, then only edges must be specified in the file:

```
VERTEX_SE2 0 0 0    0
FIX 0
VERTEX_SE2 1 4 0 1.57
VERTEX_SE2 2 4 4 3.14
VERTEX_SE2 3 0 4 3.14

VERTEX_XY 11 2 2
VERTEX_XY 12 6 2
VERTEX_XY 13 2 6

EDGE_SE2 0 1 4 0 1.57 1 0 0 1 0 1
EDGE_SE2 1 2 4 0 1.57 1 0 0 1 0 1
EDGE_SE2 2 3 4 0    0 1 0 0 1 0 1

EDGE_SE2_XY 0 11  2  2 1 0 1
EDGE_SE2_XY 1 11  2  2 1 0 1
EDGE_SE2_XY 1 12  2 -2 1 0 1
EDGE_SE2_XY 2 11  2  2 1 0 1
EDGE_SE2_XY 2 12 -2  2 1 0 1
EDGE_SE2_XY 2 13  2 -2 1 0 1
EDGE_SE2_XY 3 11 -2  2 1 0 1
EDGE_SE2_XY 3 13 -2 -2 1 0 1
```

Listing 3.1: g^{2}o protocol example

3.2 The Known Correspondence Case

In the known correspondence case each measurement incorporates the information of the landmark sensed from the map. It is equivalent to knowing the correct values of `id1` and `id2` for every `EDGE_SE2_XY`. In a real world scenario, data association is usually not known, however, the known correspondence case is still useful in simulations to check the correctness of the algorithm.

With g^{2}o, solving the known correspondence case is just a matter of loading the data to the framework, set the desired parameters, and running the solver.

The parameters that can be set in the solver include:

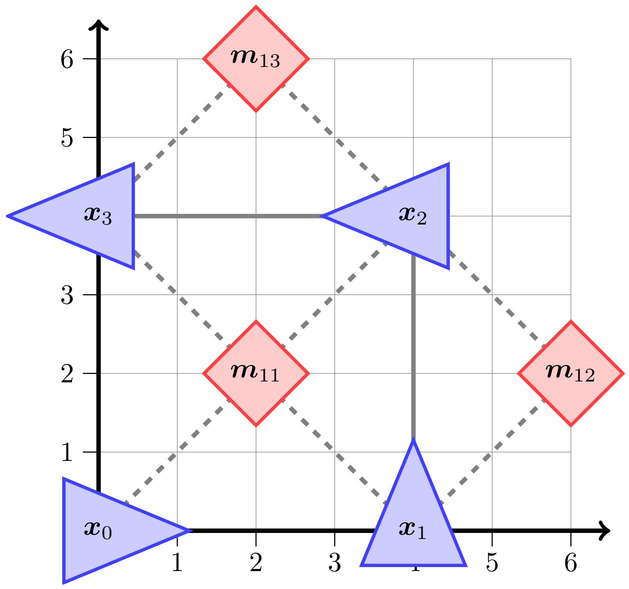


Figure 3.2: Graphical representation of g^{2}o protocol example. The lower left corner corresponds to (0,0).

1. The sparse solver for the inversion in (2.25): Cholesky solver, PCG solver.
2. The optimization algorithm for solving (2.24)-(2.26): Gauss-Newton, Levenberg-Marquardt.
3. The number of iterations for the algorithm to stop.

For the sparse solver, g^{2}o uses third-party libraries from which the user can choose: CHOLMOD, CSparse[2], Eigen[3].

This work provides of a Python script to easily set the parameters of the framework and run the solver. It also provides of a 2D simulator that generates a robot path, landmarks, and measurements. It is a modified version of the g^{2}o simulator that allows the user to set the information parameters from matrices (3.1) in the simulations. This makes it possible to test the GraphSLAM algorithm for different noise levels. The simulation also allows the user to compare the results with the *ground truth*. The ground truth is the true estimate to be achieved, and it is given by the simulator.

The Python script is also able to generate an initial guess of the estimate using robot odometry and the first measurement of each landmark. The initial guess is used as a starting point for the optimization algorithm. Figure 3.3 shows an example of the ground truth and the initial guess of a SLAM simulation.

The pseudocode for the known correspondence case is shown in Algorithm 1.

Optionally g^{2}o can use robust kernels to deal with *outliers*. An outlier is a corrupt measurement that doesn't follow the distribution assumed for the model. They are usually generated by sensors malfunctions and tend to have extreme values, far away from the expected measurement.

[2]CHOLMOD and CSparse can be found in `http://faculty.cse.tamu.edu/davis/suitesparse.html`
[3]`http://eigen.tuxfamily.org/index.php?title=Main_Page`

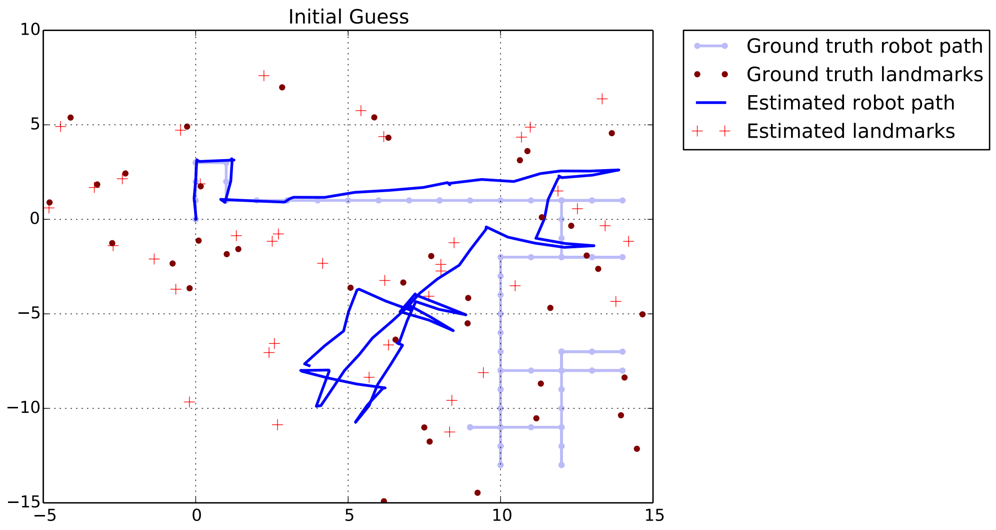


Figure 3.3: Example of an initialization of a SLAM simulation. The light blue line is the ground truth path, and dark red circles are ground truth landmarks. The blue line is odometry path, and red crosses are the initial guess for landmarks.

Algorithm 1 GraphSLAM Known Correspondence

```
Require: optimizer, data
1: optimizer.setParameters(parameters)
2: optimizer.loadData(data)
3: optimizer.genInitialGuess()
4: optimizer.solve(numberIterations)
5: optimizer.writeData()
```

From equation (2.21), it can be seen that each error function has a quadratic influence in function $F(\boldsymbol{y})$. This means that a single outlier can significantly degrade the construction of F, and thus the result of the estimation. To mitigate this problem a robust kernel function can be applied to each error term $\boldsymbol{e}_{ij}(\boldsymbol{y})$ in (2.21), so that high values of $\boldsymbol{e}_{ij}$ has reduced effect in F. Robust kernels included in g^{2}o are: Cauchy, DCS, Fair, GemanMcClure, Huber, PseudoHuber, Saturated, Tukey, Welsch. Most robust kernels must also specify the kernel width, that is the point on the function in which the kernel effect start. Figure 3.4 shows the plots of different kernels in g^{2}o.

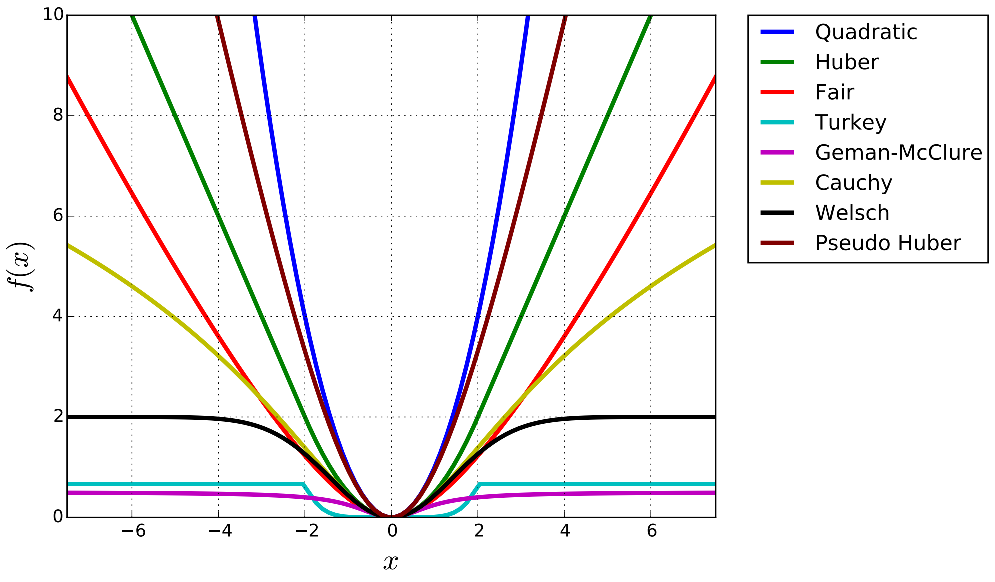


Figure 3.4: Plots of different robust kernels with equal width, compared with the quadratic function.

3.3 The Unknown Correspondence Case

In the unknown correspondence case, there is no information of which landmark generates each measurement, neither of how many landmarks are on the map. The g^{2}o framework does not provide a way to solve the correspondence problem, so a method must be developed to address the issue. In this work the method implemented is based on the one presented in [11], with some differences to take in account the speed of the algorithm.

3.3.1 The Correspondence Test

The premise of the method is as follows: a correspondence test is developed to check the likelihood of two landmarks being the same. If the likelihood is high enough, the landmarks are merged into one.

To justify mathematically this test, the variable $\boldsymbol{m}_{i,j} = (\boldsymbol{m}_i \ \boldsymbol{m}_j)^T$ is defined as the concatenation of landmarks i and j. The posterior probability of $\boldsymbol{m}_{i,j}$ over the measurements and control inputs is given by:

$$p(\boldsymbol{m}_{i,j}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \det(2\pi\boldsymbol{\Sigma}_{\boldsymbol{m}_{i,j}})^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{m}_{i,j} - \boldsymbol{\mu}_{\boldsymbol{m}_{i,j}})^T \boldsymbol{\Sigma}^{-1}_{\boldsymbol{m}_{i,j}} (\boldsymbol{m}_{i,j} - \boldsymbol{\mu}_{\boldsymbol{m}_{i,j}})\right\}, \quad (3.2)$$

Where $\boldsymbol{\mu}_{\boldsymbol{m}_{i,j}} = (\boldsymbol{\mu}_{\boldsymbol{m}_i}\ \boldsymbol{\mu}_{\boldsymbol{m}_j})^T$ is the current estimate of the landmarks i and j. Matrix $\boldsymbol{\Sigma}_{\boldsymbol{m}_{i,j}}$ is the covariance matrix marginalized over landmarks i and j. Since it has been assumed a normal distribution for the estimate, this covariance can be computed using the marginalization lemma [11]. In practice, g^{2}o provides a function to compute $\boldsymbol{\Sigma}_{\boldsymbol{m}_{i,j}}$.

Then, to compare $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ another variable is defined as the difference between the two landmarks: $\boldsymbol{\Delta}_{i,j} = \boldsymbol{m}_i - \boldsymbol{m}_j$, or alternatively, using the difference matrix:

$$D = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \quad (3.3)$$

$$\Rightarrow \boldsymbol{\Delta}_{i,j} = \boldsymbol{m}_i - \boldsymbol{m}_j = D^T \boldsymbol{m}_{i,j} \quad (3.4)$$

It can be proven that $\boldsymbol{\Delta}_{i,j}$ is distributed as $\mathcal{N}(D^T\boldsymbol{\mu}_{\boldsymbol{m}_{i,j}}, D^T\boldsymbol{\Sigma}_{\boldsymbol{m}_{i,j}}D) := \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\Delta}_{i,j}}, \boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{i,j}})$. Then its PDF is given by:

$$p(\boldsymbol{\Delta}_{i,j}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \det(2\pi\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{i,j}})^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{\Delta}_{i,j} - \boldsymbol{\mu}_{\boldsymbol{\Delta}_{i,j}})^T \boldsymbol{\Sigma}^{-1}_{\boldsymbol{\Delta}_{i,j}} (\boldsymbol{\Delta}_{i,j} - \boldsymbol{\mu}_{\boldsymbol{\Delta}_{i,j}})\right\} \quad (3.5)$$

When two landmarks are equivalent it is expected that their position is the same, hence $\boldsymbol{\Delta}_{i,j} = 0$. Evaluating this in the posterior probability (3.5) gives the likelihood of landmark equivalence:

$$\pi_{j=k} := p(\boldsymbol{\Delta}_{i,j} = 0|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \det(2\pi\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{i,j}})^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\boldsymbol{\mu}^T_{\boldsymbol{\Delta}_{i,j}} \boldsymbol{\Sigma}^{-1}_{\boldsymbol{\Delta}_{i,j}} \boldsymbol{\mu}_{\boldsymbol{\Delta}_{i,j}}\right\} \quad (3.6)$$

The correspondence test consists in assert a landmark equivalence when the likelihood $\pi_{j=k}$ is greater than a user-defined threshold χ. Intuitively a greater threshold means being more strict in considering landmark equivalences.

3.3.2 The Unknown Correspondence Algorithm

Once the correspondence test is defined, it can be used to implement a GraphSLAM algorithm with unknown data association.

The algorithm works as follows: first, all landmarks are initialized as if each measurement corresponds to an individual landmark. The correspondence test is run over every possible pair of landmarks, merging landmarks that pass the test. After the tests are finished, the estimate is updated running the solver in the same way as in the case of known correspondence. After the solver, the correspondence tests are run again, and the solver is run afterward. Correspondence test and solver are alternated until no new landmark associations are found.

Algorithm 2 presents the unknown correspondence algorithm in pseudocode.

Algorithm 2 GraphSLAM Unknown Correspondence

```
Require: optimizer, data
 1: optimizer.setParameters(parameters)
 2: optimizer.loadData(data)
 3: optimizer.genInitialGuess()
 4:
 5: while association found do
 6:     for all pair of landmark ($i$,$j$) do
 7:         if correspondenceTest($i$,$j$) $\geq \chi$ then
 8:             optimizer.merge($i$,$j$)
 9:         end if
10:     end for
11:     optimizer.solve(numberIterations)
12: end while
13:
14: optimizer.writeData()
```

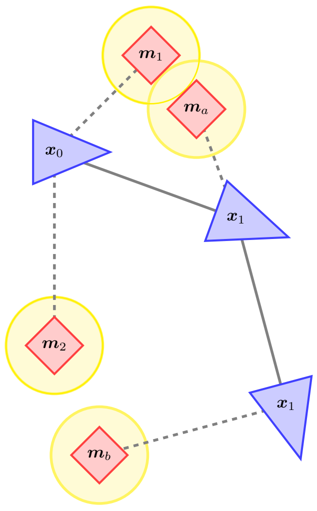
3.3.3 Speeding up the Unknown Correspondence Algorithm

Algorithm 2 is inefficient. In particular, the correspondence test is run over every pair of landmarks at each iteration. The number of pairs is quadratic in the number of landmarks, furthermore, even landmarks that are obviously not equivalent, such as landmarks greatly separated, are tested. Empirical testing has shown that the bottleneck of the algorithm is the correspondence test, in particular, the computation of the marginalized covariance $\mathbf{\Sigma}_{\mathbf{\Delta}_{i,j}}$, therefore it is necessary to call this function as less often as possible. The optimization of the algorithm is essential to run GraphSLAM in large datasets. The next subsections present the strategies adopted to improve the algorithm performance.

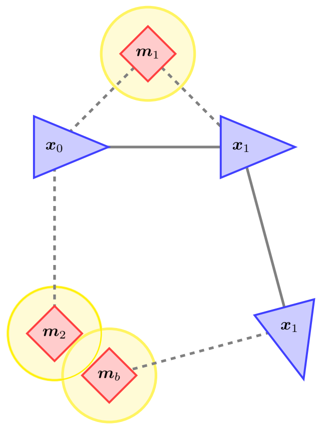
Incremental Optimization

Incremental optimization is based on the following principle: landmarks measured late in the robot's path are subject to the accumulated error of all past robot's poses. This fact is can be visualized in the simulation in Figure 3.3. Due to this, equivalent landmarks found late in the trajectory will be merged only when all the previous estimates of the pose have already been corrected by the algorithm. Testing correspondence between these landmarks earlier is pointless, simply because their error is too high to produce an association.

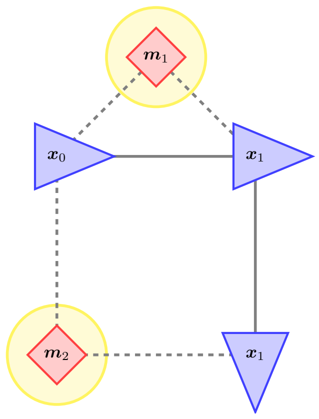
A way to deal with this problem is to test association only for landmarks measured in early poses, and then test for late poses when the path is corrected. An extreme version of this is the *incremental optimization* algorithm: at each iteration consider only the current pose. Test landmarks observed in the current pose with landmarks from all previous poses and then run the solver. In the next iteration do the same for the next pose, and repeat until the last pose. Intuitively, the algorithm is incrementally correcting the path with early measured landmarks, which have low uncertainty, until all path is corrected. Figure 3.5 illustrates the working of the incremental optimization for a simple example.



(a) Initial guess.



(b) Estimate after associating m_1 and m_a, and running the solver.



(c) Estimate after associating m_2 and m_b, and running the solver.

Figure 3.5: Example of the incremental optimization. The yellow circle represents the landmark uncertainty. Note that at first is not possible to associate landmarks $\boldsymbol{m}_2$ and $\boldsymbol{m}_b$ because $\boldsymbol{m}_b$ has the accumulated error of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. It's not until the association between $\boldsymbol{m}_1$ and $\boldsymbol{m}_a$, observed in previous poses, is made that the latter landmarks can be merged.

The incremental optimization has a subtle flaw, once a correspondence test is run between two landmarks, it's never tested again, since only current measurements are tested. If a test failed at some point, but posterior corrections make it possible for the test to pass, this association will be missed. It is expected that this situation doesn't occur often, since the incremental optimization algorithm usually gives a good estimate of the path up to the current pose. Nevertheless, this error is absolutely possible.

To mitigate this problem, *full optimizations* can be run occasionally along with the incremental optimization. Full optimization consists in testing correspondence between all possible pairs of measured landmarks up to the current pose, in a similar fashion as it is done in Algorithm 2. This way a compromise can be achieved between the algorithm speed and performance. In this implementation, the user is able to choose the frequency with which the full optimizations are run. It is done by setting the parameter *io* (inter full optimizations), which is the number of incremental optimization performed between two full optimizations.

Pose Skipping

The incremental optimization algorithm described above runs the solver and produces an optimization every time after a new pose is analyzed. In some cases, the solver optimization becomes the bottleneck rather than the data association. In these cases, it is convenient to test for association between several consecutive poses after running the solver.

This strategy is named *pose skipping*, where a parameter ps is introduced to control the number poses to be skipped after the next optimization. If $ps = n$ it indicates that the next optimization will be executed n poses after the current pose. By default $ps = 1$ (no skipping is made).

Distance Test

Even with incremental optimization, there are still a lot of landmarks that are unnecessarily tested. The distance test strategy attempts to avoid testing landmarks that are widely separated, and thus are obviously not equivalent. To do this, a distance threshold is defined, with which, every pair of landmarks separated by a greater distance automatically fails the test.

Defining a value for the threshold is nontrivial. Two methods are implemented. The first one is by user-defined input. If the user has prior knowledge of the robot or the map, he/she could make a good estimate of the maximum distance between measurements from the same landmark. Then the user can set the distance threshold to this value.

Even if the maximum distance is not known a priori, a threshold value can still be calculated. Consider the threshold χ for the correspondence test in Algorithm 2, for a given value of χ, there exists a distance for which, even in the best case scenario, landmarks separated by this distance or more with never be associated. To prove this, consider the random variable for the distance $\boldsymbol{\delta}_{i,j}$ between two landmarks. Its PDF is given by:

$$p(\boldsymbol{\delta}_{i,j}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k}) = \frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{i,j}}}} \exp\left\{-\frac{(\boldsymbol{\delta}_{i,j}-\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}})^2}{2\sigma^2_{\boldsymbol{\delta}_{i,j}}}\right\} \tag{3.7}$$

$$d_{i=j} := p(\boldsymbol{\delta}_{i,j}=0|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k}) = \frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{i,j}}}} \exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{i,j}}}{2\sigma^2_{\boldsymbol{\delta}_{i,j}}}\right\} \tag{3.8}$$

Which is equivalent to the PDF $p(\boldsymbol{\Delta}_{i,j}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k})$ in (3.5), for the one dimensional case. $\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{i,j}}$ is the Euclidean distance between the mean of the landmarks, and $\sigma^2_{\boldsymbol{\delta}_{i,j}}$ is the variance of $p(\boldsymbol{\Delta}_{i,j}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k})$ projected in the line between $\boldsymbol{\mu}^2_{\boldsymbol{\Delta}_{i,j}}$ and $(0,0)$. For this analysis, the exact value of $\sigma^2_{\boldsymbol{\delta}_{i,j}}$ is not important.

The maximum distance at which association can still be made is given by the following problem:

$$\max_{d_{i=j} \geq \chi} \quad |\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}}| \tag{3.9}$$

Considering only the positive values of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}}$, $d_{i=j}$ becomes a decreasing function. By this fact, it can be seen that maximum at (3.9) is achieved when $d_{i=j} = \chi$. A visual demonstration of this is shown in Figure 3.6. Expanding the equality gives:

$$d_{i=j} = \chi \tag{3.10}$$

$$\frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{i,j}}}} \exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{i,j}}}{2\sigma^2_{\boldsymbol{\delta}_{i,j}}}\right\} = \chi \tag{3.11}$$

$$\exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{i,j}}}{2\sigma^2_{\boldsymbol{\delta}_{i,j}}}\right\} = \sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{i,j}}}\chi \tag{3.12}$$

$$\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}} = \sqrt{-2\sigma^2_{\boldsymbol{\delta}_{i,j}} \log(\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{i,j}}}\chi)} \tag{3.13}$$

So the maximum distance for an association is given by (3.13). Unfortunately the distance is a function of $\sigma^2_{\boldsymbol{\delta}_{i,j}}$, which is computationally expensive to get. However expression (3.13) is concave in $\sigma^2_{\boldsymbol{\delta}_{i,j}}$, so a global maximum can be found (see Figure 3.7). Intuitively this means that neither a large nor a low value of $\sigma^2_{\boldsymbol{\delta}_{i,j}}$ is good for associating landmarks at great distances.

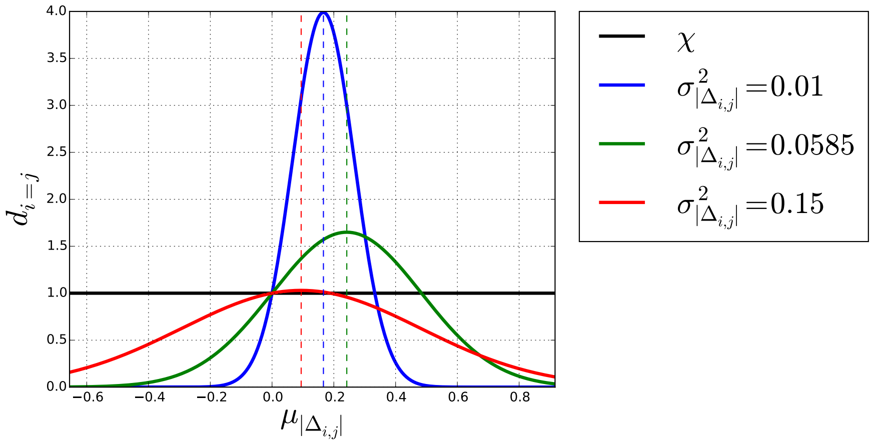


Figure 3.6: Plots of the PDF of the distance between two landmarks for different values of $\sigma^2_{\boldsymbol{\delta}_{i,j}}$. The plots shows that maximum distance from zero is achieve when, $d_{j=k} = \chi$. Note that the function that achieve more distance is neither the one with more nor with less variance.

Using differential calculus it can be found maximum of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}}$ for $\sigma^2_{\boldsymbol{\delta}_{i,j}}$ is achieved in $\sigma^2_{\boldsymbol{\delta}_{i,j}} = \frac{1}{2\pi e\chi}$, which yields a value of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}} = \frac{1}{\sqrt{2\pi e}\chi}$. This value can be used as a distance threshold that only depends on χ. The disadvantage of this method is that the threshold computed tend to be large relative to the problem, so the amount of correspondence tests avoided is limited.

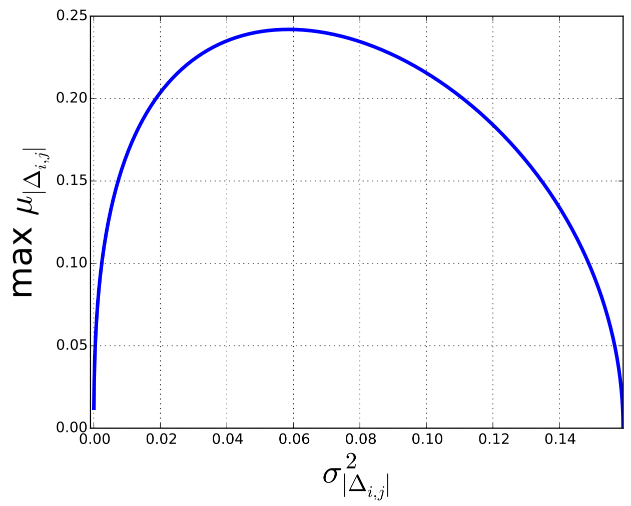


Figure 3.7: Plot of the maximum distance $\boldsymbol{\mu}_{\boldsymbol{\delta}_{i,j}}$ in function of $\sigma^2_{\boldsymbol{\delta}_{i,j}}$. It can be seen that is a concave function.

3.4 The Final Algorithm

The final version of the implemented GraphSLAM algorithm solves the SLAM problem for unknown correspondence, it incorporates the incremental optimization, pose skipping and distance test strategies to speed up the algorithm. The implementation can be downloaded from repository in `https://github.com/francocurotto/GraphSLAM`, with instructions on how to install, compile and run the program.

The algorithm of the final version of GraphSLAM is presented in Algorithm 3. *ps* and *io* correspond to the parameters for pose skipping and inter full optimization. To check if it is the right time to use either of the strategies, the module (%) operator is used between the pose number and the parameter.

The GraphSLAM algorithm calls to functions, `intrementalDataAssociation` and `full-Optiomization`. The `intrementalDataAssociation` function tries to associate the landmarks observed in the current pose, with the previous landmarks. Its pseudocode is shown in Algorithm 4. The distance test is performed in line 5, where *dt* is the maximum distance accepted for an association. `fullOptimization` function search associations between all previous poses up to the current pose, and then run the solver. It can be done by calling `intrementalDataAssociation` iteratively, as shown in 5.

Algorithm 3 GraphSLAM Final Version

```
Require: optimizer, data
 1: optimizer.setParameters(parameters)
 2: optimizer.loadData(data)
 3: optimizer.genInitialGuess()
 4:
 5: for all poses $p$ do
 6:     INCREMENTALDATAASSOCIATION(p)
 7:     if $p \% ps = 0$ then
 8:         optimizer.solve(numberIterations)
 9:     end if
10:     if $p \% io = 0$ then
11:         FULLOPTIMIZATION(p)
12:     end if
13: end for
14:
15: optimizer.writeData()
```

Algorithm 4 Incremental Data Association Function

```
 1: function INCREMENTALDATAASSOCIATION($p$)
 2:     for all landmarks $l_p$ observed in $p$ do
 3:         for all previous poses $q$ up to $p$ do
 4:             for all landmarks $l_q$ observed in $q$ do
 5:                 if distance($l_p$,$l_q$) $< dt$ then
 6:                     if correspondenceTest($i$,$j$) $\geq \chi$ then
 7:                         optimizer.merge($i$,$j$)
 8:                     end if
 9:                 end if
10:             end for
11:         end for
12:     end for
13: end function
```

Algorithm 5 Full Optimization Function

```
1: function FULLOPTIMIZATION($p$)
2:     while association found do
3:         for all previous poses $q$ up to $p$ do
4:             INCREMENTALDATAASSOCIATION($q$)
5:         end for
6:         optimizer.solve(numberIterations)
7:     end while
8: end function
```

Chapter 4

Results

In this chapter, the results of the GraphSLAM algorithm are presented for different test scenarios. In Section 4.1 the algorithm is tested for the simple case of known data association and simulated data. A parameters variation analysis is made, and their effect on the path estimation is shown. In Section 4.2 a similar analysis is performed, but for the case of unknown data association. In this case, new parameters related to the data association algorithm are tested. Finally in Section 4.3, the GraphSLAM algorithm is run in more realistic data. This data includes a robot simulated with Gazebo[1], a much more realistic robot simulator, and data obtained from real robots in outdoor environments.

Through these tests, several parameters and methods must be chosen for the algorithm to work properly. To limit the scope of this work, the sparse solver and the optimization algorithm are fixed and used in all the following tests. CSpase library and the Cholesky decomposition is used for the sparse solver, and the Levenberg-Marquardt method is used as the optimization algorithm. Also, whenever the robust kernel method is used, Huber is the chosen kernel function (see Section 3.2).

4.1 Known Data Association

The known data association case is the easiest one of the three scenarios because correspondence between landmarks is given a priori. In this case Algorithm 1 is used.

The parameters to be set in this case are the following:

- n_p: number of poses of the robot path.
- n_l: number of landmarks in the map.
- i_{op}: odometry position information (inverse of variance).
- i_{oa}: odometry angle information.
- i_{lp}: landmark position information.

[1] `http://gazebosim.org/`

- it: number of iterations of the optimization algorithm.
- k_w: width of the chosen robust kernel.

Where n_p, n_l, i_{op}, i_{oa}, and i_{lp} are parameters regarding the robot behavior in the test. They are passed to the simulator. The parameters it and k_w define de optimization strategy.

Through all the tests made in this work, it is assumed that the information matrix of odometry and measurements models (same as (3.1)) are diagonal, i.e., their variables are not correlated. Furthermore, it is assumed that these matrices have the following structure:

$$\boldsymbol{R}_k^{-1} = \begin{pmatrix} i_{op} & 0 & 0 \\ 0 & i_{op} & 0 \\ 0 & 0 & i_{oa} \end{pmatrix} \quad \boldsymbol{Q}_k^{-1} = \begin{pmatrix} i_{lp} & 0 \\ 0 & i_{lp} \end{pmatrix} \tag{4.1}$$

This means that the robot experiences same uncertainty in the x and y axis, for both, motion and measurement model.

Test I is run with the parameters of Table 4.1.

The simulation is constrained to a 2D world with $x \in [-15, 15]$, $y \in [-15, 15]$. At each step the simulator moves the robot 1 unit of distance and in turns randomly an angle of $\theta = 0°$, $90°$, or $-90°$. All simulations start from at $(0, 0)$.

A kernel width of 1 unit is chosen heuristically, looking at the distance between landmarks. Intuitively, the user should set the kernel width to a distance in which two landmarks are unlikely to be the same, and therefore, it corresponds to an outlier.

The results of the test are shown in Figure 4.1. In Figure 4.1a the initial guess is plotted on the left, and the posterior estimation after the solver on the right. The ground truth is added in both graphs for comparison. Landmarks not observed by the robot are not shown. It can be seen that the path of the initial guess rapidly diverges from the real solution. On the other hand, the solver estimation fits quite well with the ground truth, both for the path and the landmarks.

To have a more quantitative view of the error made by the solver, the cumulative path error for every step is shown in Figure 4.1b. The cumulative normalized error is given by:

$$error(i) = \frac{1}{i} \sum_i ||pos_{GT} - pos_{est}|| \tag{4.2}$$

Where i is the path's timestep. pos_{GT} and pos_{est} are the ground truth and estimated position respectively. Then the error is the sum of the Euclidean distance between both positions, normalized by the number of steps.

It can be seen that the error start increasing early in the path, but then it gets stabilized. This is the effect of aggregating the information of the measurements and running the optimizer. As long as the robot keep measuring the same landmarks, it can maintain its error low.

In the next tests, all the information parameters of the robot are decreased simultaneously. Is intended to show the effect in the estimation when adding uncertainty to the robot.

Test II, Test III, and Test IV, shows the results when $i_{op} = i_{oa} = i_{lp} = 100$, $i_{op} = i_{oa} = i_{lp} = 10$, and $i_{op} = i_{oa} = i_{lp} = 1$ respectively.

It can be seen that the estimated path gradually degrades as the information decrease. Finally, when the information is 1, the algorithm breaks.

Test V shows the effect running the algorithm with a low number of iterations. In this case $it = 1$. It can be seen that the algorithm was not able to converge properly, in contrast to Test I.

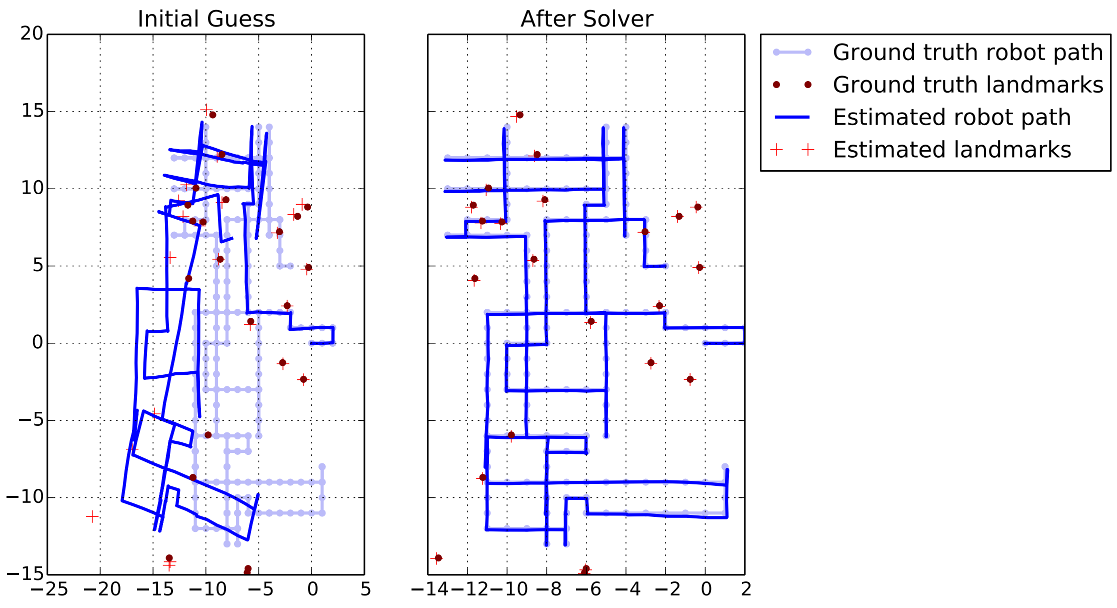
In Test VI the kernel width was reduced to $k_w = 0.1$, meaning the errors are suppressed at a shorter distance. It can be seen that the new width actually improves the results, getting a normalized error for the full path of around 0.08.

In terms of speed the algorithm performs quite fast, taking around $1[s]$ for all the tests.

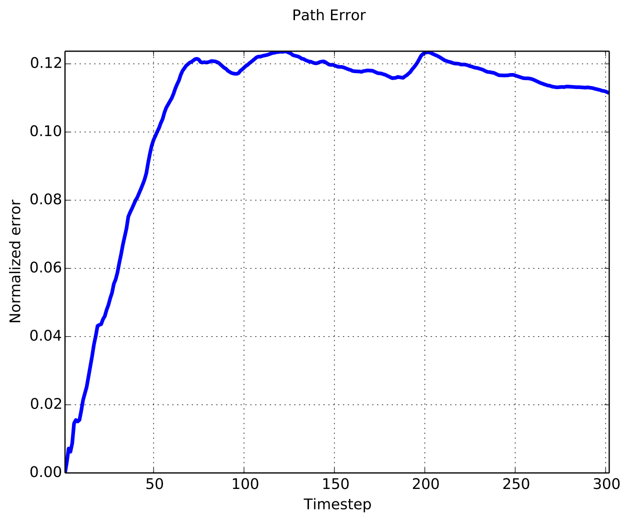
Test I.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	1000	1000	1000	20	1

Table 4.1: Parameters for Test I.



(a) Initial guess and solver estimation.



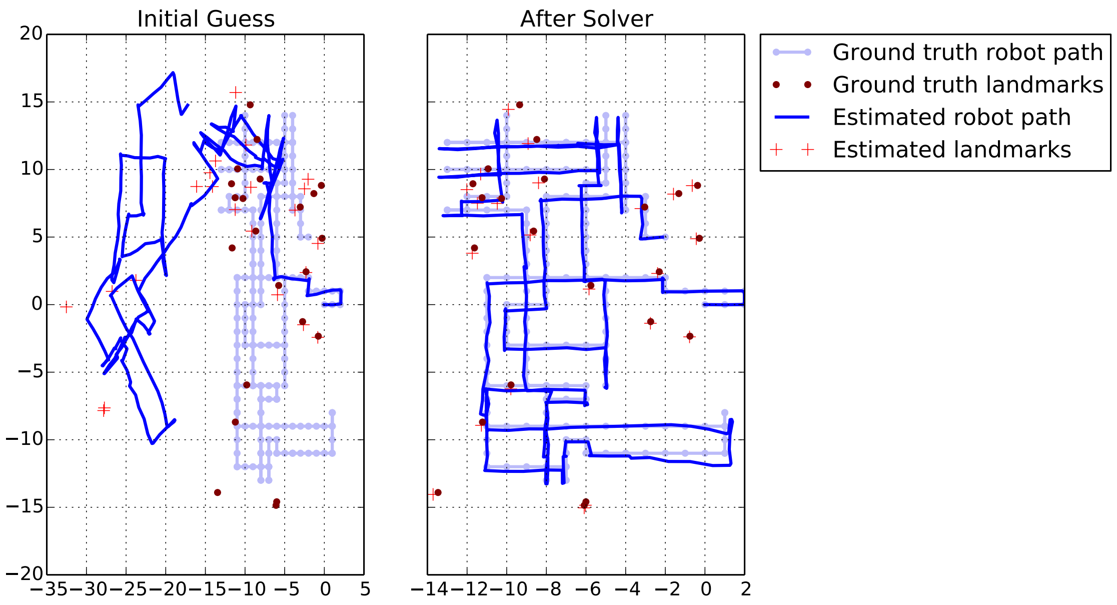
(b) Path normalized error.

Figure 4.1: Results for Test I.

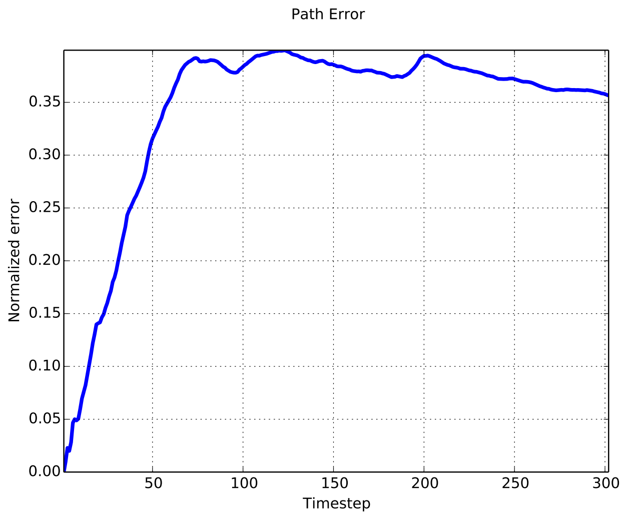
Test II

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	100	100	100	20	1

Table 4.2: Parameters for Test II.



(a) Initial guess and solver estimation.



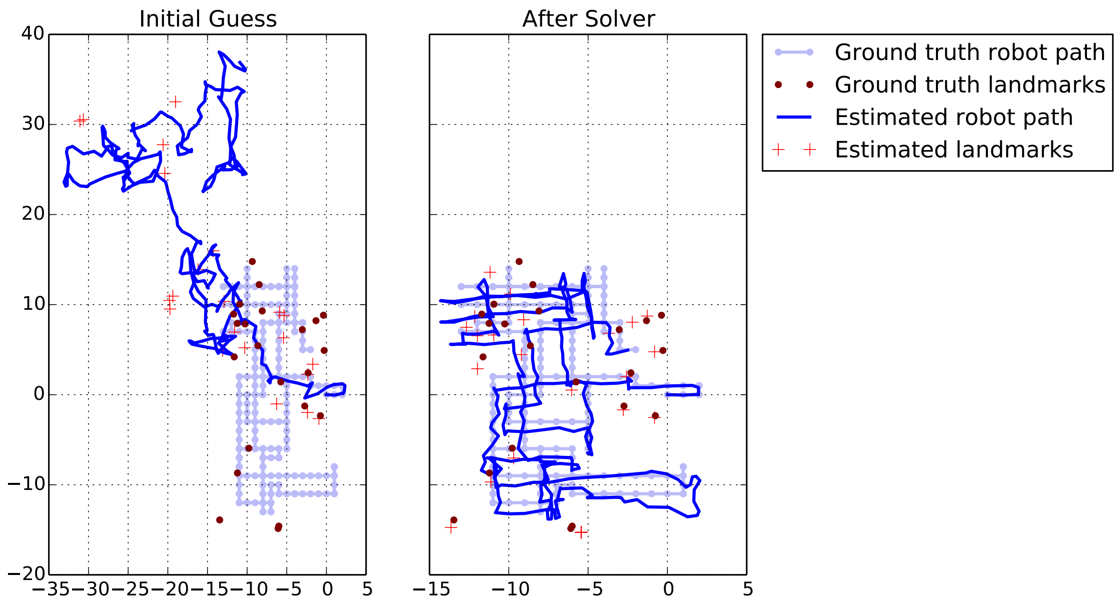
(b) Path normalized error.

Figure 4.2: Results for Test II.

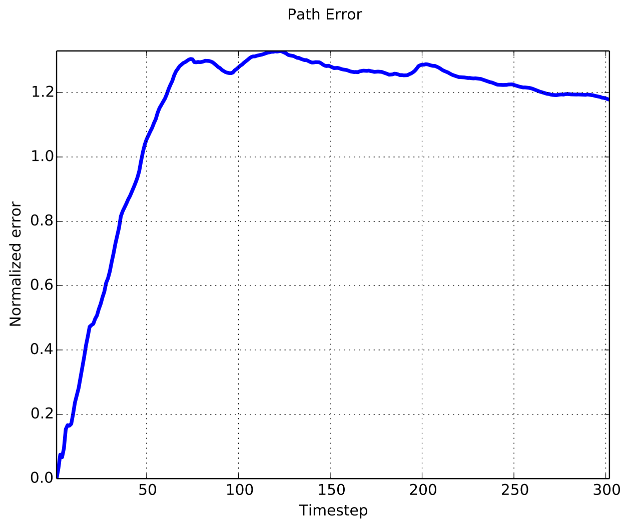
Test III

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	10	10	10	20	1

Table 4.3: Parameters for Test III.



(a) Initial guess and solver estimation.



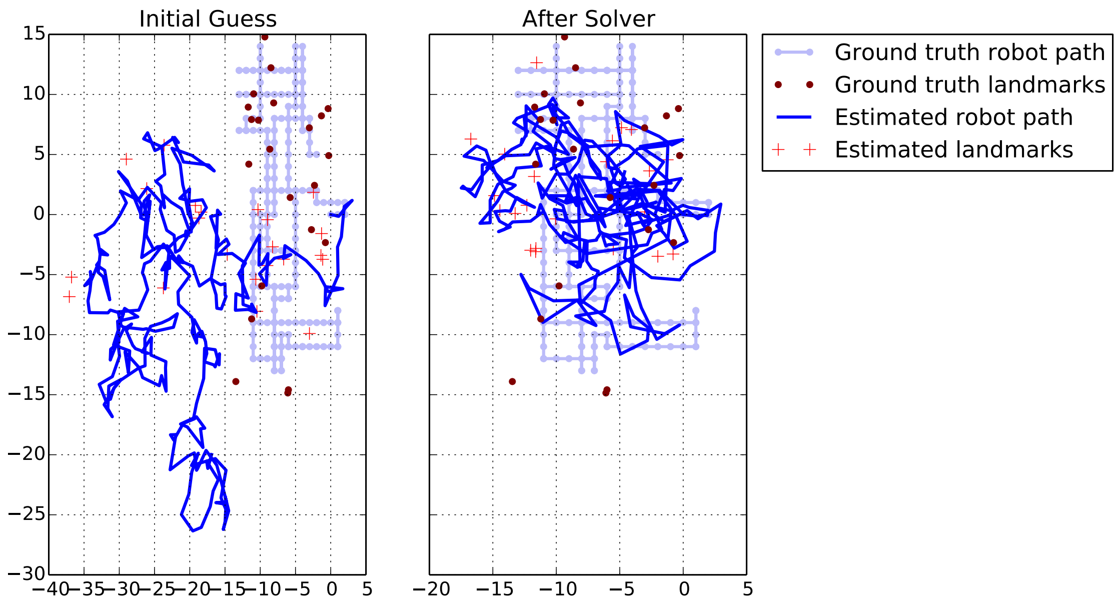
(b) Path normalized error.

Figure 4.3: Results for Test III.

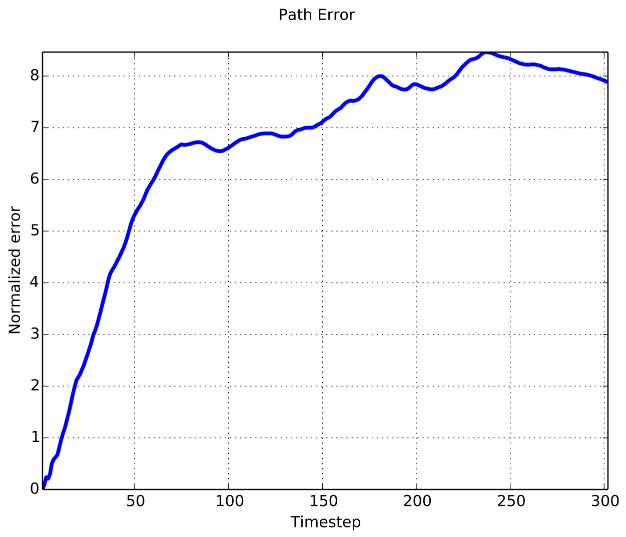
Test IV

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	1	1	1	20	1

Table 4.4: Parameters for Test IV.



(a) Initial guess and solver estimation.



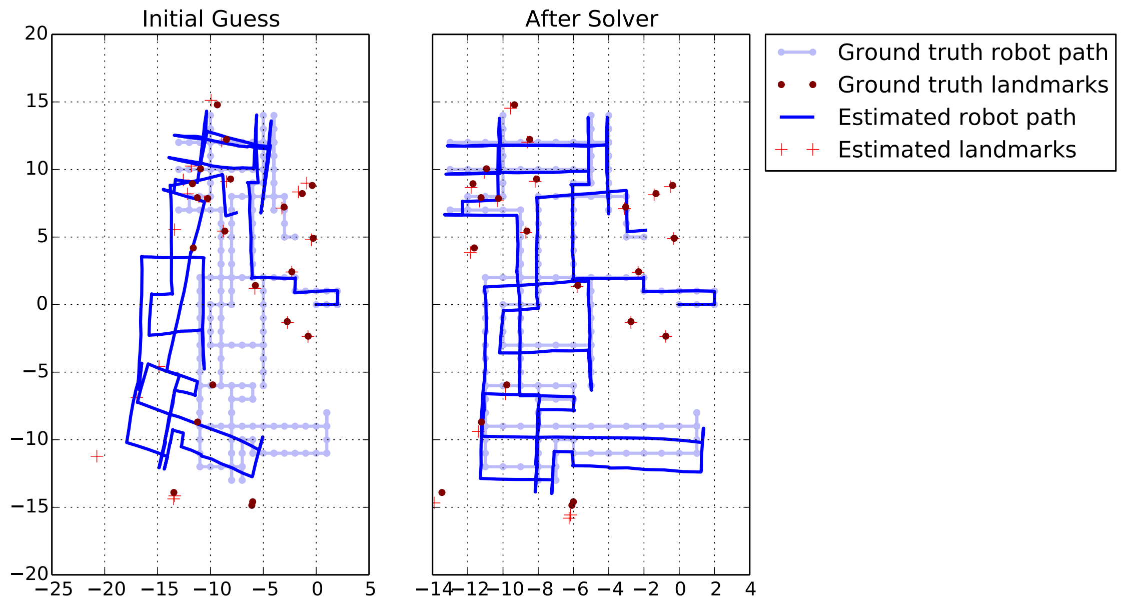
(b) Path normalized error.

Figure 4.4: Results for Test IV.

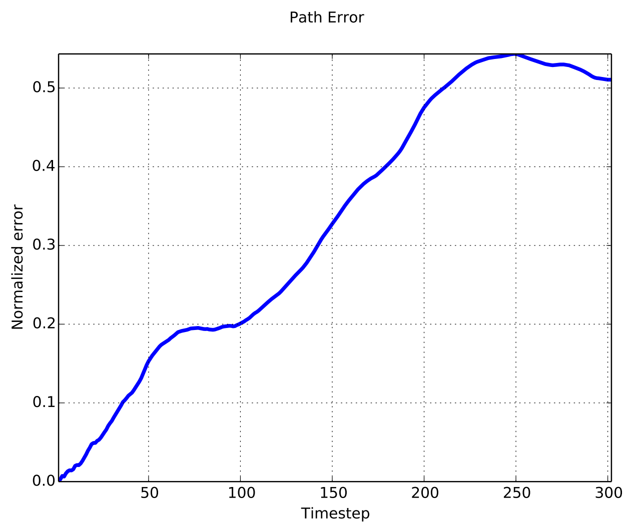
Test V.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	1000	1000	1000	1	1

Table 4.5: Parameters for Test V.



(a) Initial guess and solver estimation.



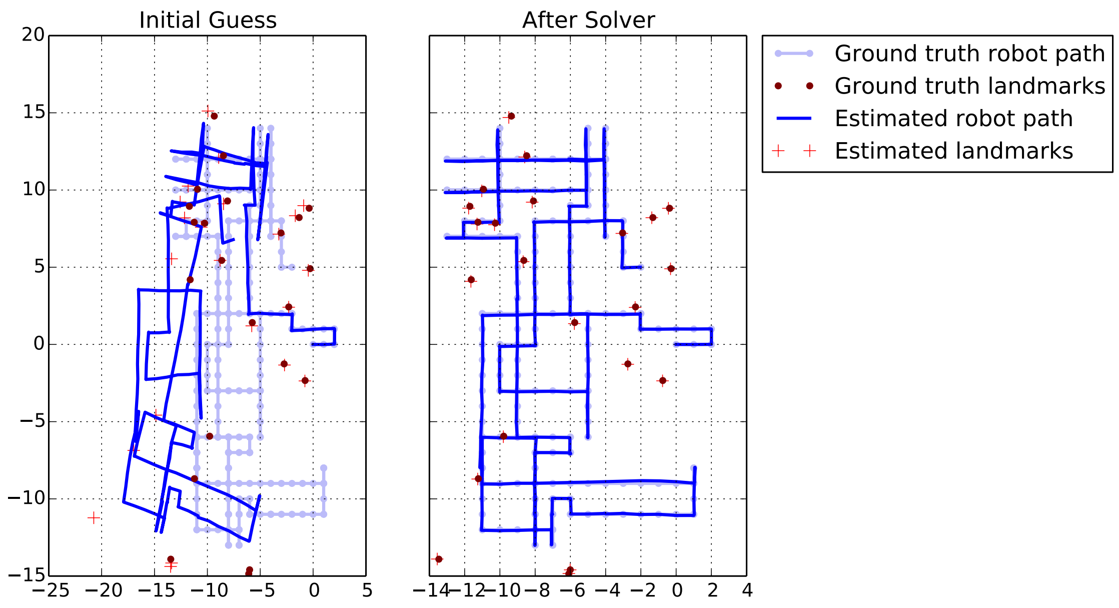
(b) Path normalized error.

Figure 4.5: Results for Test V.

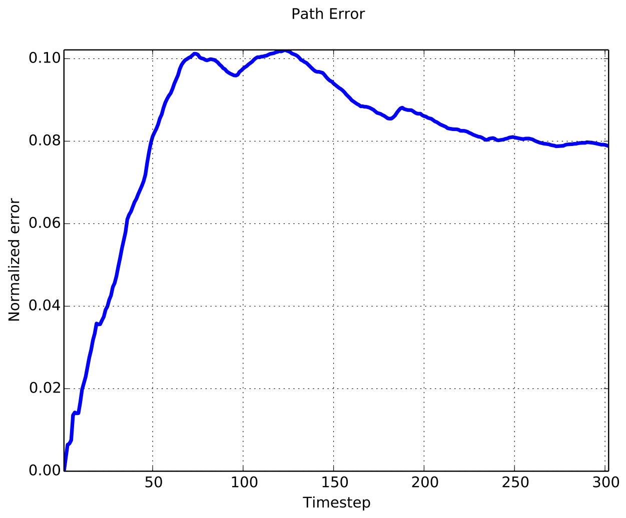
Test VI.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w
300	40	1000	1000	1000	20	0.1

Table 4.6: Parameters for Test VI.



(a) Initial guess and solver estimation.



(b) Path normalized error.

Figure 4.6: Results for Test VI.

4.2 Unknown Data Association

In this section, similar tests are made, but in this case, it is assumed unknown data association of the landmarks. The same simulator is used to generate the data, and Algorithm 3 to compute the estimate.

Along with the parameters used in the known correspondence case, the following new parameters must be set in these tests:

- χ: likelihood threshold for data association.
- dt: maximum distance for distance test.
- io: inter full optimization frequency.
- ps: Pose skipping.

Where χ and dt are the thresholds used in Algorithm 4. io is the number of incremental optimizations between two full optimizations, and ps is the number of poses between optimizations, both from Algorithm 3.

Test VII show the results for a successful test with unknown data association. Notice that in the initial guess there are several more landmarks than in the ground truth. That is because the initial guess considers every single measurement as an independent landmark. However in the result of the solver, the algorithm is able to associate the measurements with the corresponding landmark, and correct the path of the robot. Nevertheless, there are 4 cases that are unable to be associated correctly.

The parameter dt is set to ∞ so no distant test is performed. In Table 4.7, variable t correspond to the total time of the algorithm.

In the next tests, the threshold χ is modified. In Test VIII the threshold is set to a very low value $\chi = 10^{-100}$. It can be seen that the algorithm merges nonequivalent landmarks, thus causes estimate divergence. In Test IX and Test X χ is set to 1 and 3 respectively. In these cases, the thresholds are so high that the algorithm is unable to associate all the equivalent landmarks. This cause that the estimated path gradually drift.

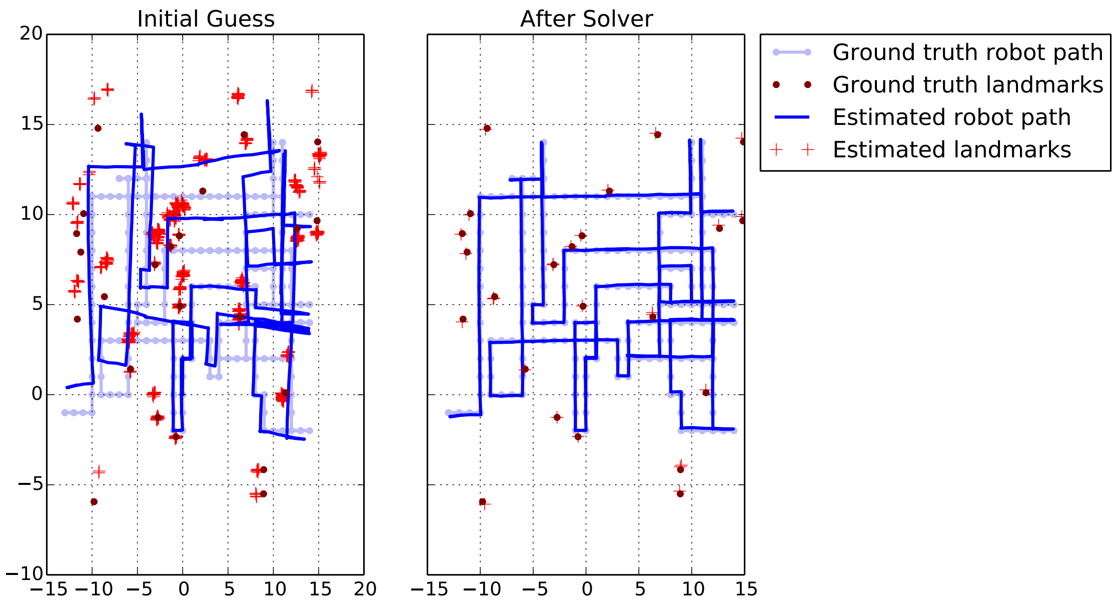
In Test XI $\chi = 0$ and $dt = 1$, so that in this case the algorithm associates landmarks using the distant test instead of the correspondence test. Every landmark at a distance of 1 unit will be automatically associated. Looking at Figure 4.11a it can be seen that now all landmarks are correctly associated. However, Figure 4.11b shows that the estimation has a greater cumulative error. Also, the distant test is faster than the correspondence test, taking only $5[s]$. Therefore, a trade-off exists between using the distant test and the correspondence test as a means to solve unknown association.

In Test XII the pose skip parameter is increased to $ps = 100$. the idea is to do as least optimization as possible to increase the algorithm speed. It can be seen however that, due to the lack of optimization, the algorithm is unable to correct the path and to associate the landmarks. This errors also cause that the algorithm actually takes longer than in Test VII.

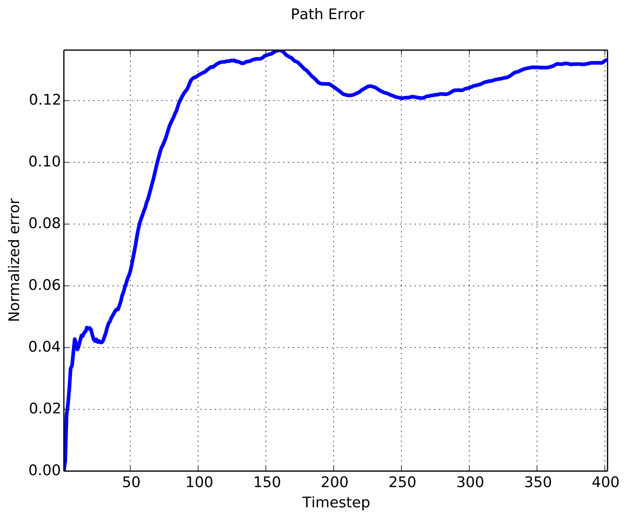
Test VII.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	0.1	∞	400	10	$13[s]$

Table 4.7: Parameters for Test VII.



(a) Initial guess and solver estimation.



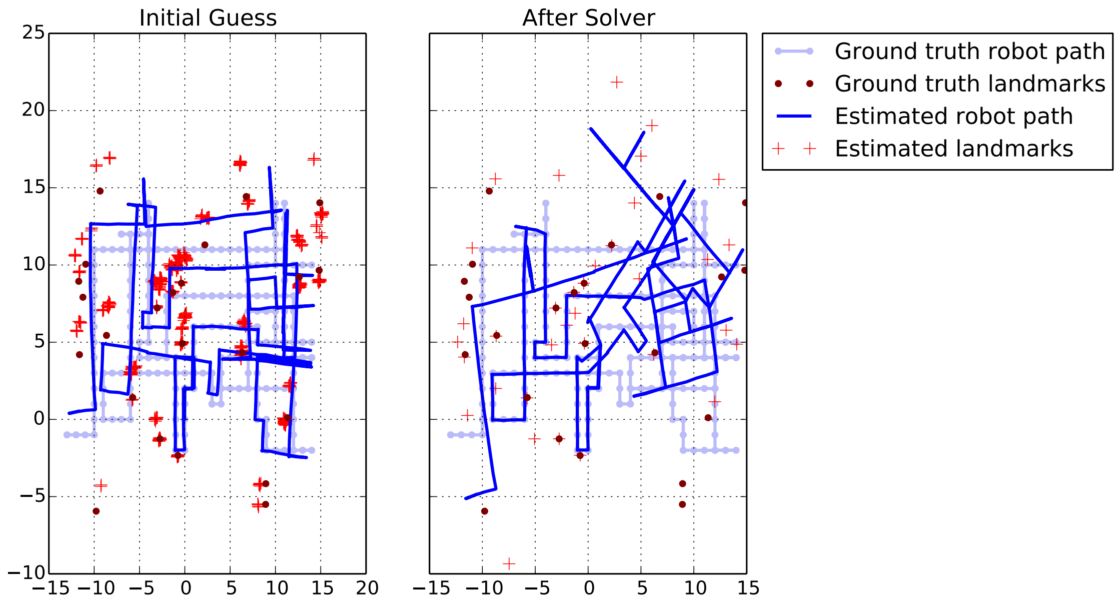
(b) Path normalized error.

Figure 4.7: Results for Test VII.

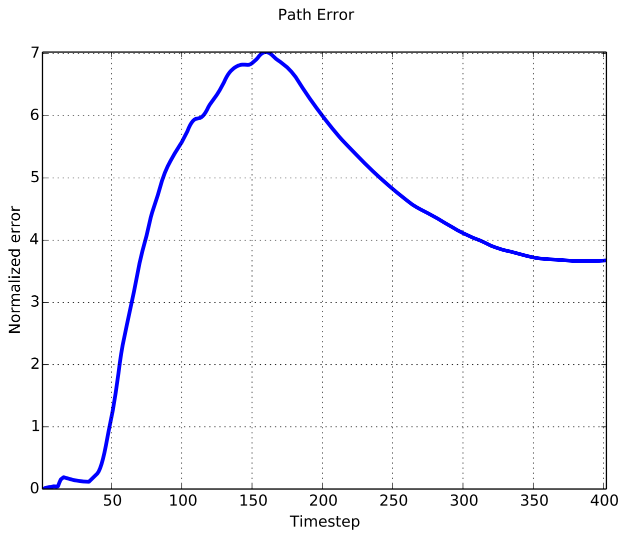
Test VIII.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	10^{-100}	∞	400	10	$94[s]$

Table 4.8: Parameters for Test VIII.



(a) Initial guess and solver estimation.



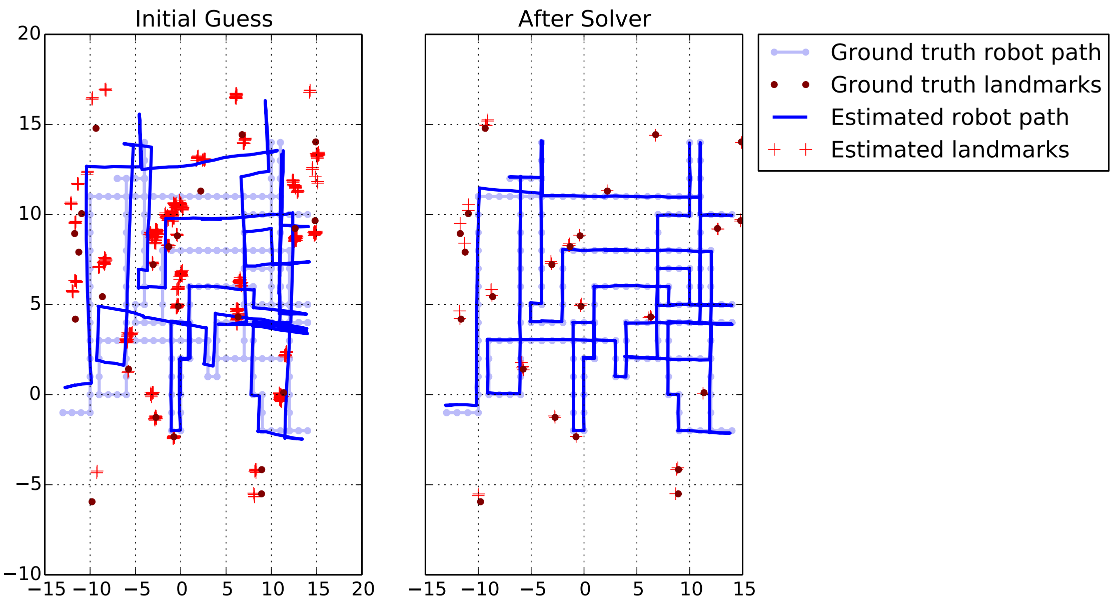
(b) Path normalized error.

Figure 4.8: Results for Test VIII.

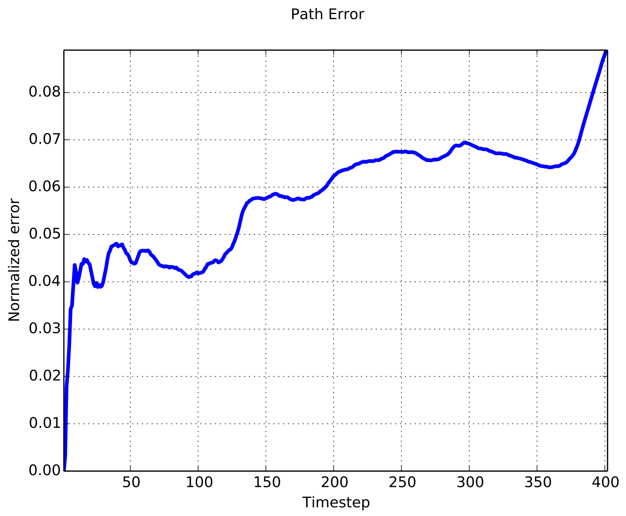
Test IX.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	1	∞	400	10	$10[s]$

Table 4.9: Parameters for Test IX.



(a) Initial guess and solver estimation.



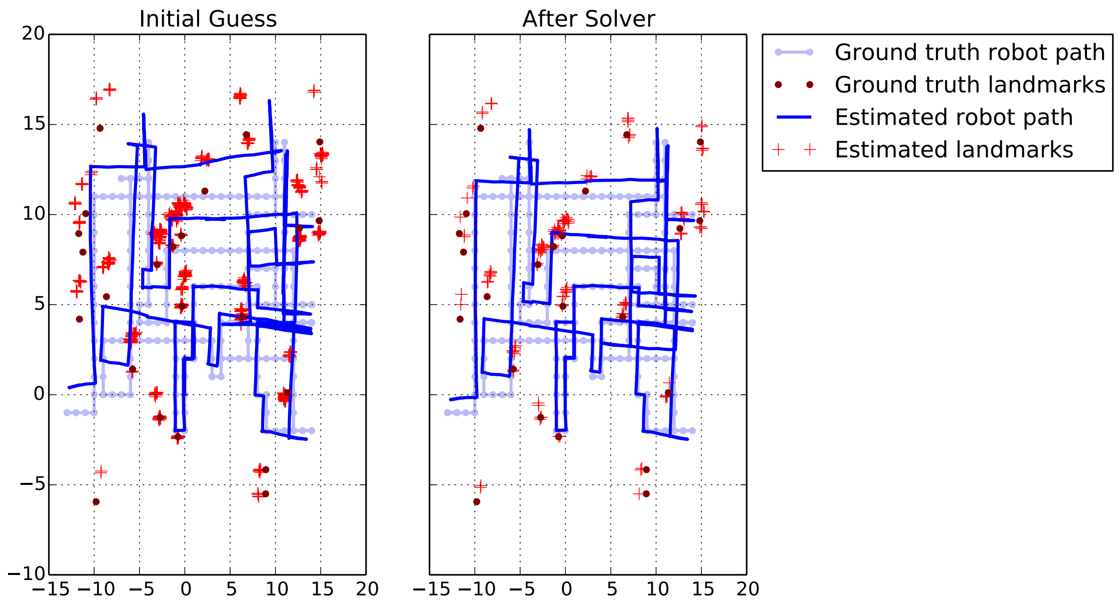
(b) Path normalized error.

Figure 4.9: Results for Test IX.

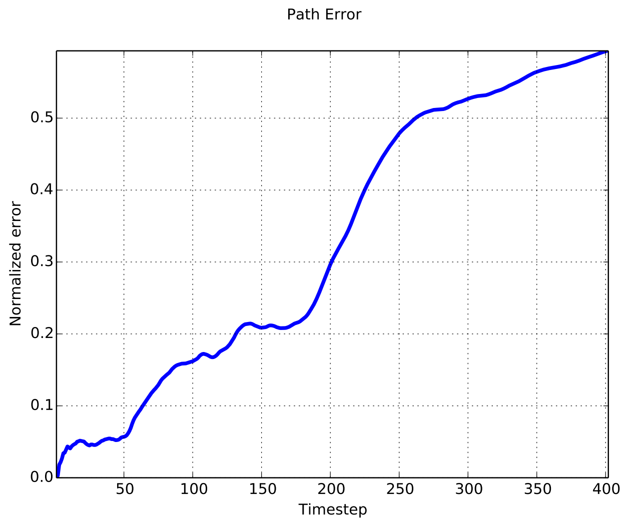
Test X.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	3	∞	400	10	17$[s]$

Table 4.10: Parameters for Test X.



(a) Initial guess and solver estimation.



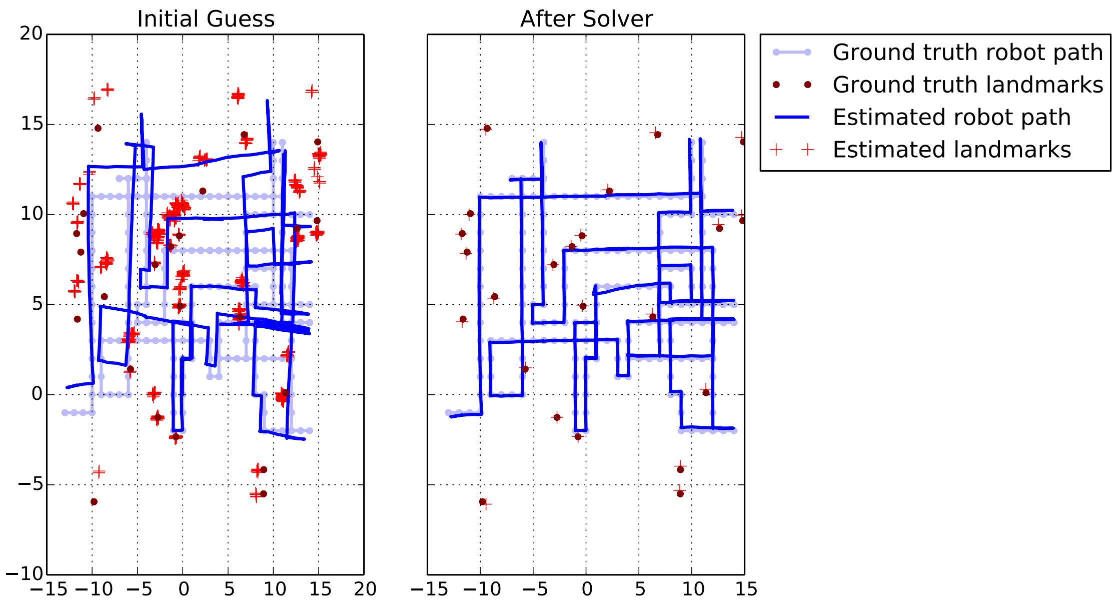
(b) Path normalized error.

Figure 4.10: Results for Test X.

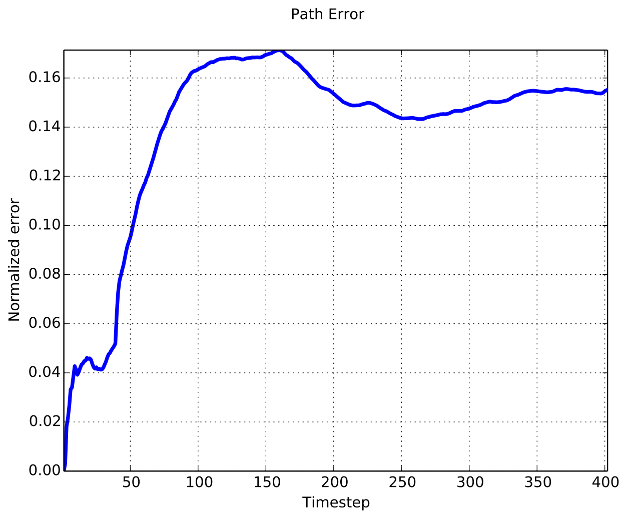
Test XI.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	0	1	400	10	$5[s]$

Table 4.11: Parameters for Test XI.



(a) Initial guess and solver estimation.



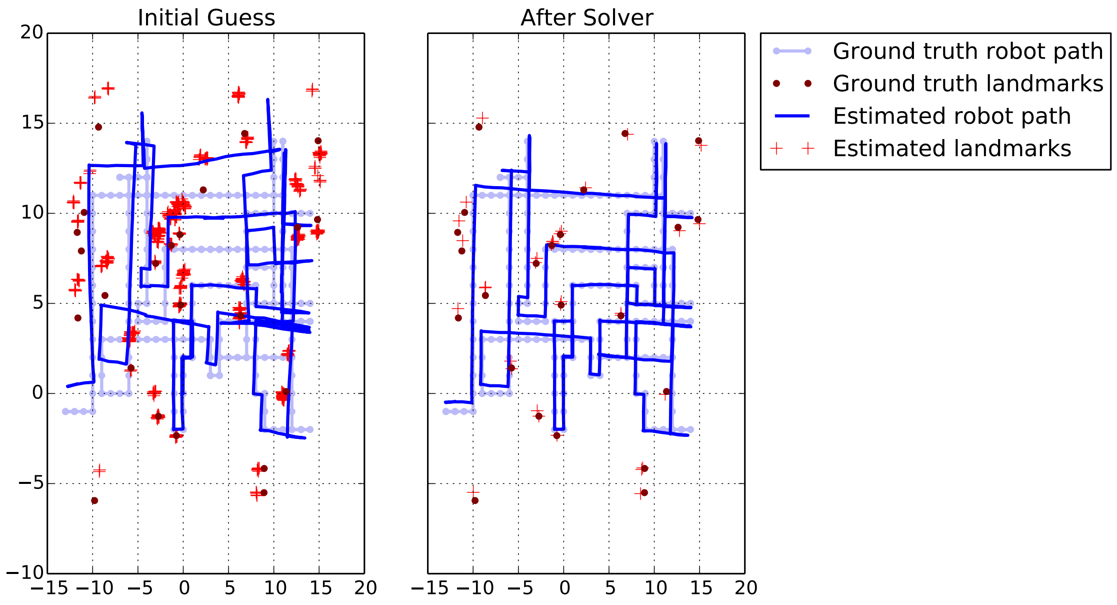
(b) Path normalized error.

Figure 4.11: Results for Test XI.

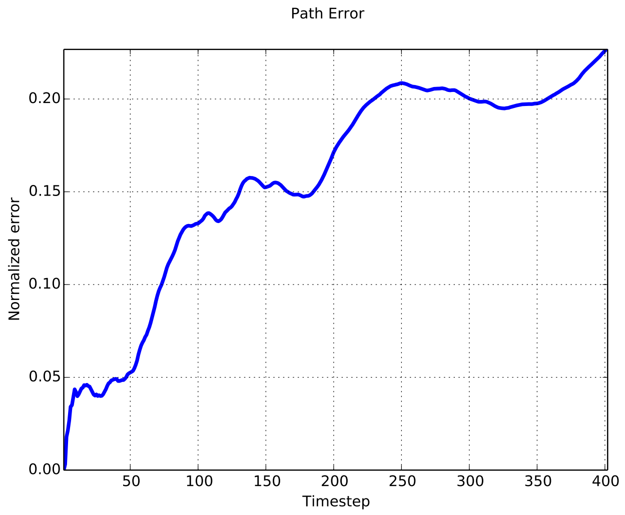
Test XII.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
400	30	1000	10000	1000	20	1	0.1	∞	400	100	$21[s]$

Table 4.12: Parameters for Test XII.



(a) Initial guess and solver estimation.



(b) Path normalized error.

Figure 4.12: Results for Test XII.

4.3 Real Data

In this section, the correctness of the algorithm is proven for more realistic data. The size of the data is much larger than the previous cases, the number of poses in a path are in the order of thousands and tens of thousands. Unfortunately, the amount of data is so large, that it becomes infeasible to apply the correspondence test because it so computationally expensive it takes several hours to run the algorithm. For this reason, only the distant test is used. Moreover, the data presents undesired properties like non-Gaussian noise and false alarms, nevertheless, it'll be shown that it's still possible to apply the algorithm and get good results.

4.3.1 Husky a200 Robot

The first of these test is a Husky a200 Robot simulated in Gazebo. Since this test is still a simulation the ground truth and the path error can still be retrieved. However, since Gazebo is a more realistic simulation, the nonlinear models of the robot, miss in the detections, and false alarms are also simulated.

The parameters of the algorithm are presented it Table 4.13. In this case the information parameters i_{op}, i_{oa}, and i_{lp} are the assumed uncertainty that better fit the robot model. Since no knowledge was acquired about the robot, these parameters were chosen by trial and error.

The results are shown in Figure 4.13. It can be seen that the algorithm can correctly predict the robot path. Most of the landmarks are also found, but there a lot of false positives, possibly generated by sensor malfunctions, spurious measurements, or moving objects. A possible way to get rid of these false alarms is to apply a policy of elimination of landmarks if they are not measured a certain number of times. Nevertheless, the false alarms do not affect greatly the results of the estimation.

4.3.2 Parque O'Higgins

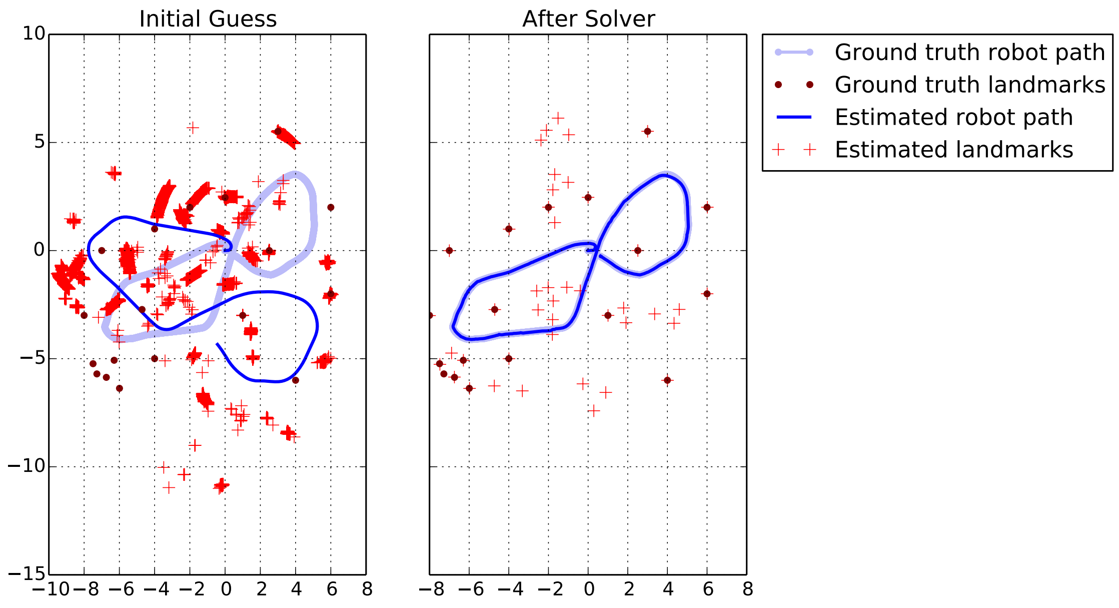
This dataset corresponds to data taken in Parque O'Higgins by a robot from Universidad de Chile. Since this is real data there is no ground truth with which compare the robot path. However, ground truth of landmarks (trees in this case) was made by hand using GPS image. Also, the robot was controlled so that it return to the starting position at the end of the path, so it is possible to check the correctness of the path by looking at the robot's final position.

The results are shown in Test XIV, along with the parameters used in the test. It can be seen that in the initial guess the robot doesn't return to the starting position (0,0), however, it does when the algorithm is applied, thus proving the correctness of the estimate. Again, the robot is able to detect the landmarks, but a lot of false positive are generated in the process. Due to the large size of the data, the algorithm took 4 hours to finish.

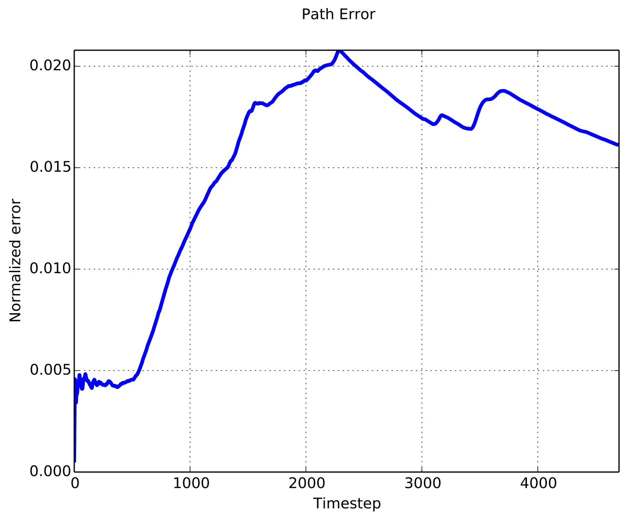
Test XIII.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
4700	18	10000	10000	1000	10	1	∞	0.5	500	10	$3[min]$

Table 4.13: Parameters for Test XIII.



(a) Initial guess and solver estimation.



(b) Path normalized error.

Figure 4.13: Results for Test XIII.

Test XIV.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
8130	174	100000	100000	100	10	1	∞	3	500	10	$4[hrs]$

Table 4.14: Parameters for Test XIV.

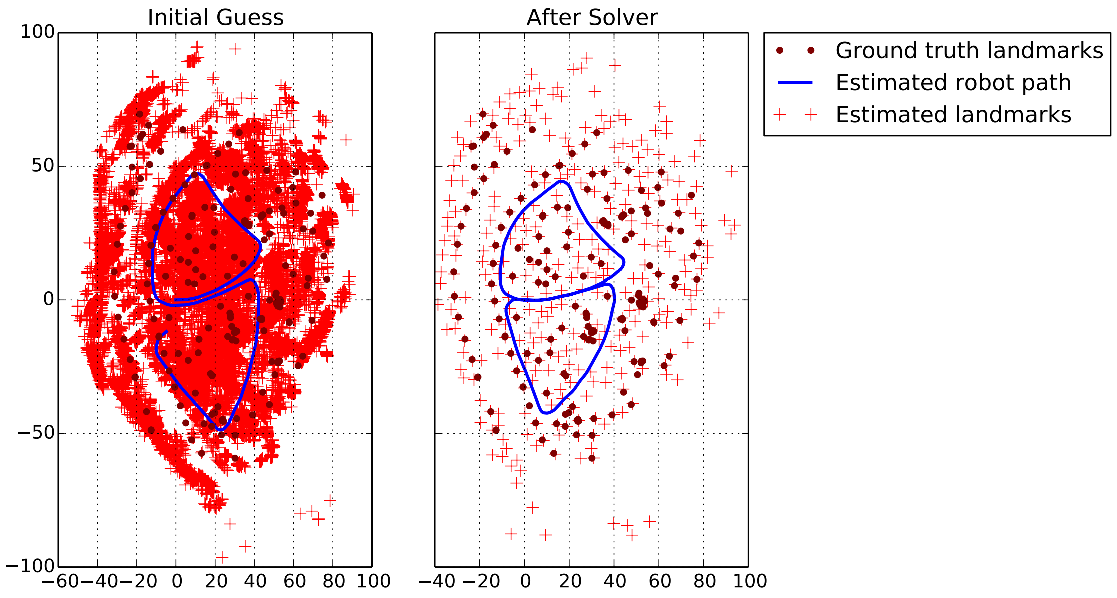


Figure 4.14: Results for Test XIV.

4.3.3 Victoria Park

The Victoria Park[2] is a commonly used dataset to test algorithms related to localization, mapping, and SLAM. No real ground truth of the map exists, however in this case the results can be compared with ones obtained by similar algorithms.

Test XV show the results of the algorithm in the Victoria Park dataset. In Figure 4.16 the results are compared with the iSAM algorithm [5] for the same dataset. It can be appreciated the resemblance between the two results. The main difference is the number of landmarks, since this implementation of GraphSLAM does not have a way to eliminate false alarms, they will stay forever in the map. The main disadvantage of the GraphSLAM algorithm is the computation time, due to the size of the dataset, it took 10 hours to finish.

[2]Avaliable in `http://www.mrpt.org/Dataset_The_Victoria_Park`

Test XV.

n_p	n_l	i_{op}	i_{oa}	i_{lp}	it	k_w	χ	dt	io	ps	t
61763	-	8000	100000	5	15	1	∞	5	500	10	$10[hrs]$

Table 4.15: Parameters for Test XV.

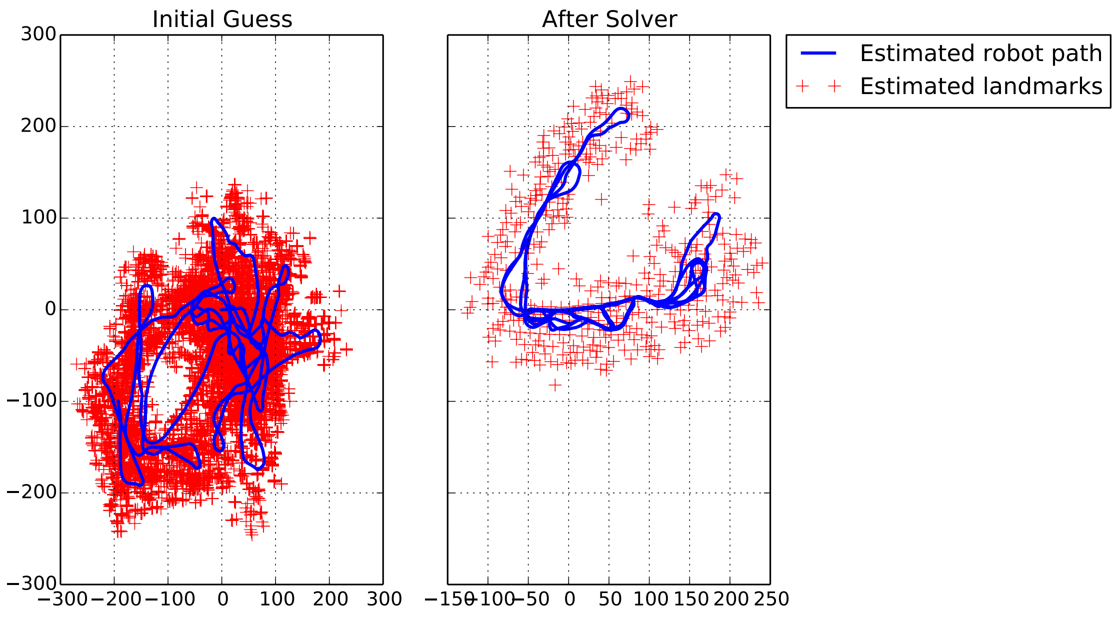
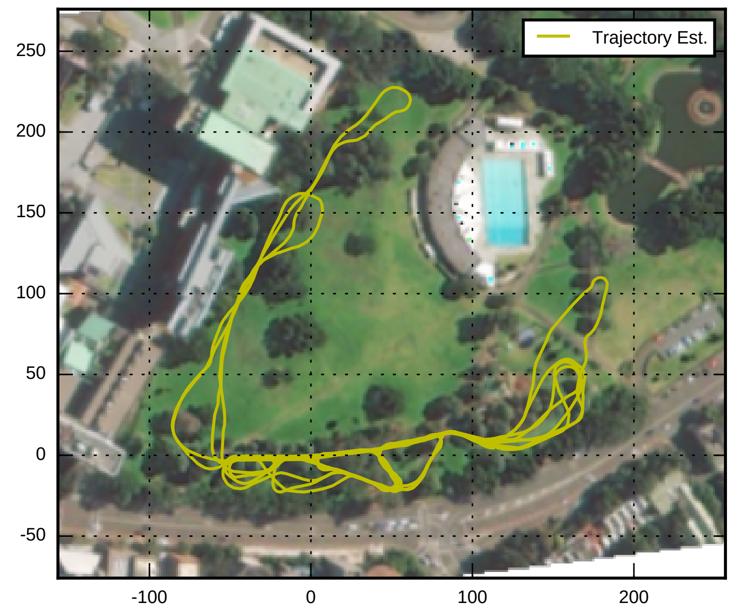
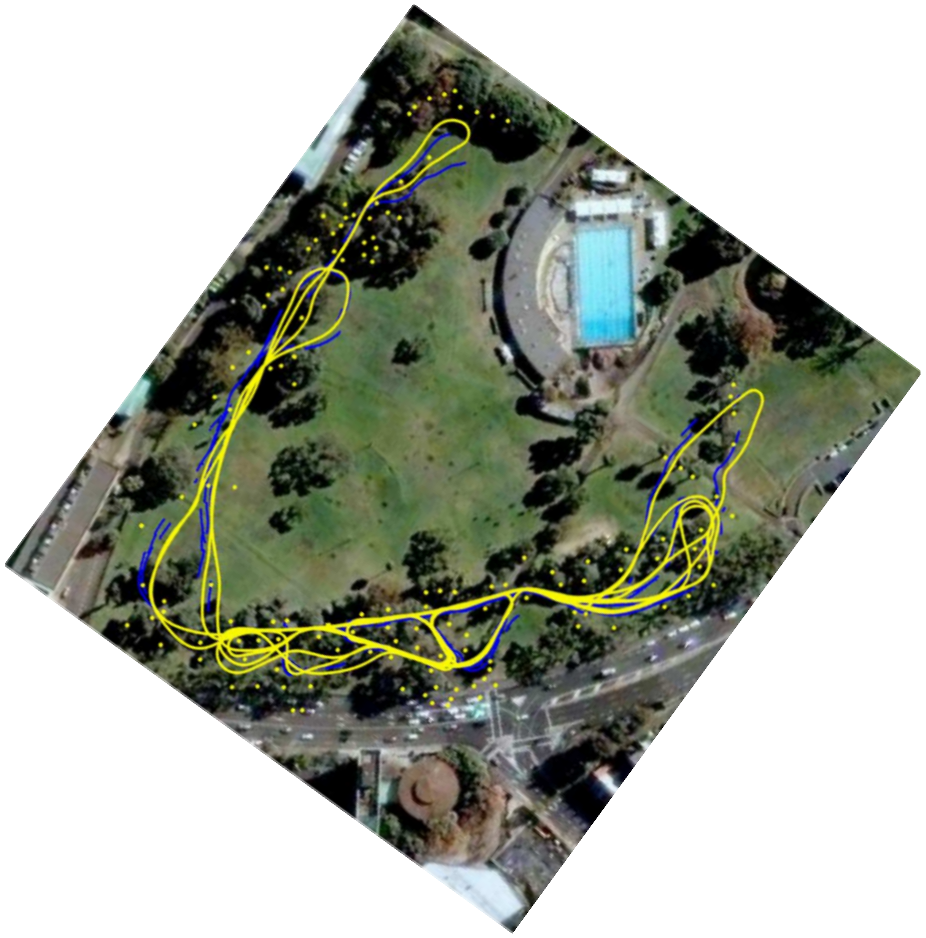


Figure 4.15: Results for Test XV.



(a) Results GraphSLAM in the Victoria Park dataset (trajectory only).



(b) Results iSAM in the Victoria Park dataset [5].

Figure 4.16: Comparison of results between GraphSLAM and iSAM.

Chapter 5

Conclusions

In the present work, the GraphSLAM algorithm was implemented for solving the SLAM problem in the 2D scenario. The g^{2}o framework provided to be a good tool for least squares nonlinear optimization, and it includes kernels methods for robust estimation. The algorithm makes some assumptions about the robot models, in particular, it assumes zero-mean white Gaussian noise and the Markov condition. However, GraphSLAM proved to be robust to outliers and non-Gaussian noise.

The GraphSLAM implementation is able to deal with known and unknown data association of the landmarks. The known data association case is a simple application of the g^{2}o framework with the right parameters. The unknown data association is a more complicated case since there is no a straightforward way to deal with it. A method of maximum likelihood was developed using the uncertainty information that can be retrieved from g^{2}o to create a correspondence test. Then this test was applied iteratively to give a solution to the unknown correspondence. To speed up the algorithm several strategies were implemented: incremental optimization, pose skipping and distant test, all were able to decrease significantly the computational time.

The GraphSLAM algorithm was also designed to be fine-tuned, with several parameters for the user to set. these parameters include the information in the robot model, the number of iterations of the optimization algorithm, the kernel width, the number of full optimizations, and the threshold of the correspondence and distant tests.

The algorithm was tested with simulated and real data, for known and unknown data association. The algorithm performed satisfactorily, being able to correct the path in all the tested cases, and achieved a low error when the ground truth was available. It was even able to work correctly with real data, where several assumption about the robot model and the data no longer hold. The implementation was also able to get results comparables with popular algorithms like iSAM.

A parameter analysis was made for several test scenarios, from which is concluded that a thorough choice of some parameters must be made for the algorithm to work correctly, while others parameters affect the convergence speed. The biggest drawbacks of the algorithm are the computation time, that can take in the order of hours for large datasets, and the trial-and-error tuning of the parameters that has to be made if no prior information of the robot model is known.

As future work it is suggested to improve the algorithm speed, for example, by incrementally constructing the graph to optimize in g^{2}o. Another unsolved issue in this work is the management of false positives. It is suggested to implement a policy to discard landmarks that are observed only a limited number of times, especially when future observations doesn't find landmarks where they should. The algorithm could also be extended to work in the 3 dimensional case. This extension shouldn't be difficult since g^{2}o already support nodes and edges of three dimensions.

Chapter 6

Bibliography

[1] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[2] MWMG Dissanayake, Paul Newman, Steven Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241, 2001.

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

[4] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, page 0278364911430419, 2011.

[5] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. isam: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, 2008.

[6] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g^{2}o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[7] Yasir Latif, César Cadena, and José Neira. Robust loop closing over time for pose graph slam. *The International Journal of Robotics Research*, page 0278364913498910, 2013.

[8] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[9] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[11] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.