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

IMPLEMENTACIÓN DEL ALGORITMO GRAPHSLAM PARA LA RESOLUCIÓN DEL PROBLEMA DE SLAM

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
DICIEMBRE 2015

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FRANCO ANDREAS CUROTTO MOLINA
FECHA: DICIEMBRE 2015
PROF. GUÍA: SR. MARTIN ADAMS

IMPLEMENTACIÓN DEL ALGORITMO GRAPHSLAM PARA LA RESOLUCIÓN DEL PROBLEMA DE SLAM

Una dedicatoria corta. Por ejemplo, A los creadores de U-Campus

Agradecimientos

Tabla de Contenidos

Índice de Tablas

Índice de Ilustraciones

Capítulo 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 like 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 move in an environment. Their 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 positions to continue working properly. In robotics, the problem of estimating the robot 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 can't be retrieved with absolute certainty, but it must be estimated by means of probabilistic methods.

If the environment in which the robot is immerse is also unknown, it must be estimated alongside with the position. The problem of getting an estimate of the robot 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 somewhat as the "Holy Grail" of mobile robotics, as knowing both, robot location and the environment, are crucial for every robot to work properly. 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 [10], is considered to perform better than classical filtering techniques, such as Extended Kalman Filter, in addition of having lower complexity than the Paticle 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 sparse matrix, for which efficients algorithms exist to compute the robot's path and the map.

1.1. General Objectives

The main objective of this work is to implement an offline version of the GraphSLAM algorithm for solving 2D SLAM. The implementation should be able to handle known and unknown data association, and be robust to non-Gaussian noise, outliers and false alarms. The g^{2}o (general graph optimization) framework[1] 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^{2}o 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 is 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 Conclusión the results are discussed, and possible future work is suggested.

[1]https://github.com/RainerKuemmerle/g2o

Capítulo 2

Antecedents

This chapter presents the theoretical framework in which this work is developed. It has the purpose to introduce readers unfamiliar with the topic of robotics, and give a theoretical foundation to 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. Finally, in section 2.2.2 GraphSLAM algorithm is presented and discussed in detail.

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 a 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 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 seeking a specific objective. It is useful to keep a record of all the control inputs that has been applied over time. We denote a single control input at time k 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 $\boldsymbol{u}_{k_1:k_2} = \boldsymbol{u}_{k_1}, \boldsymbol{u}_{k_1+1} \dots, \boldsymbol{u}_{k_2}$.

Sensors are used to take measurements of the environment. 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 in the environment: $\boldsymbol{m} = \boldsymbol{m}_1, \boldsymbol{m}_2, \dots, \boldsymbol{m}_N$. 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.

In this work use a *feature-based* map. In feature-based maps, each element of the map correspond to a distinct object, called *landmarks* or *features*, 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 sensors 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 feature measured, $\boldsymbol{z}_k^{\mathrm{i}}$ denotes the i-th feature detected at time k. Note that the measurement $\boldsymbol{z}_k^{\mathrm{i}}$ is a vector, since several properties can be sensed from a single landmark (e.g., distance and relative angle from robot). The list of measurements taken form 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 describes the behavior of the robot must be defined. The *pose* of the robot defines the state variables that depend only on the robot (e.g., position, bearing, etc.). The pose at time k is denoted as $\boldsymbol{x}_k$.

The robot motion model describes how the robot pose changes from one timestep to the next, given a control input. 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 amount of uncertainly in it. Therefore, it is advantageous to use probabilistic models that take in account this uncertainly. 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 $\boldsymbol{R}_k$ ($\boldsymbol{\delta}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{R}_k)$).

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 $\boldsymbol{u}_k$:

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

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

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

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

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

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

In principle there is no error-free method to associate the detection number i with the landmark number j. A good simplification is to assume that the association is automatically given by a special function $j = c_k^{\mathrm{i}}$, 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 an 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 states external to the robot, 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 certainly, 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 form 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's assumed that the location of the robot is known, and it's 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. 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 find a solution to them [9].

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 form the control input and the measurements. In this case, localization and mapping has 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, i.e. only at time k. This is called the *online SLAM* problem. In some application is useful to estimate the robot pose at every instant of time since the robot start its motion, that is, the robot trajectory. In this case the SLAM problem is stated within a sightly different way:

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

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

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

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

2.2.1. Correspondence Problem in SLAM

Until now is is being assumed that when the robot makes a measurement $\boldsymbol{z}_k$, it can successfully associate the i-th detection 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 non-trivial 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 features 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 measurement, 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.*

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

2.2.2. The GraphSLAM Algorithm

GraphSLAM is an algorithm for solving SLAM. It was first presented in [10]. It transforms the SLAM posterior (2.9) in a graphical network, called *graph*, that represents the log-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 compose of two types of entities, *nodes* and *edges*. Nodes are abstract entities that are uniquely identify 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. Figure 2.1 shows an example of a graph.

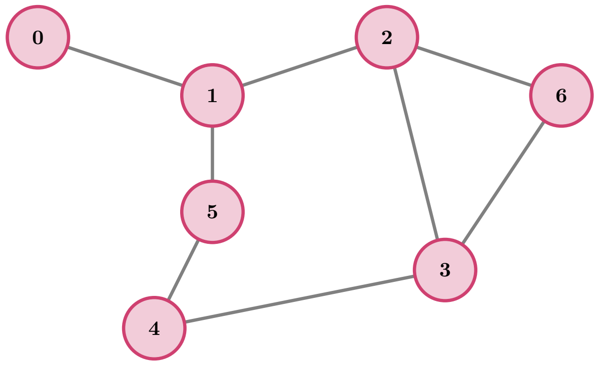


Figura 2.1: An example of a graph, with 6 nodes and 8 edges.

In the GraphSLAM context, nodes and edges represents specifics variables of the state. Nodes represents two types of state variables: it could be either the pose of the robot a certain time $\boldsymbol{x}_k$, 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 $\boldsymbol{x}_{k+1}$, this correspond to the translation that the robot realize 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 $\boldsymbol{z}_k^{\mathrm{i}}$. Figure 2.2 illustrates the graph generated by a robot, as it moves in a map and take measurement 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).

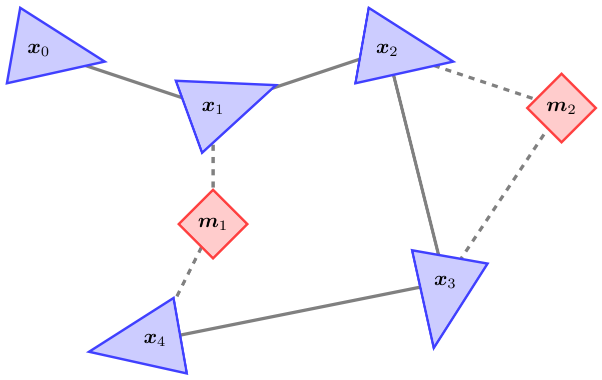


Figura 2.2: GraphSLAM illustration in 2D equivalent to Figure 2.1. The blue triangles are robot poses, and the red diamonds are landmarks positions. The solid lines represents the robot motion and the dashed lines the robot measurements.

Quadratic Form of SLAM Posterior

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

Using classical probability theory one can find a close form for (2.9), given by [10]:

$$p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \eta\, p(\boldsymbol{x}_0) \prod_k \left[p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \prod_{\mathrm{i}} p(\boldsymbol{z}_k^{\mathrm{i}}|\boldsymbol{x}_k, \boldsymbol{m}) \right] \tag{2.11}$$

Where η is a normalization factor, and $p(\boldsymbol{x}_0)$ is the initial knowledge of the robot pose. For this derivation the Markov assumption (next state is independent of past states given current state), the assumption of conditional independence between robot pose and map given control inputs and measurements, and conditional independence of measurement given robot pose and map is used. For mathematical proposes, it is convenient to work with the negative log-likelihood of the posterior:

$$\begin{aligned} &-\log(p(\boldsymbol{x}_{0:k}, \boldsymbol{m}|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \\ &\qquad -c - \log(p(\boldsymbol{x}_0)) - \sum_k \left[\log(p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)) \sum_{\mathrm{i}} \log(p(\boldsymbol{z}_k^{\mathrm{i}}|\boldsymbol{x}_k, \boldsymbol{m})) \right] \end{aligned} \tag{2.12}$$

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 $p(\boldsymbol{z}_k^{\mathrm{i}}|\boldsymbol{x}_k, \boldsymbol{m})$ in (2.5). For the initial belief, as usual, it is assumed a zero-mean Gaussian distribution with covariance $\boldsymbol{\Omega}_0$ ($p(\boldsymbol{x}_0) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Omega}_0)$). In virtue of the assumption of independent Gaussian noise, replacing all these expression into (2.12), gives a quadratic form for the negative log-SLAM posterior:

$$
\begin{aligned}
&-\log(p(\boldsymbol{x}_{0:k}, \boldsymbol{m} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \\
&\qquad c + \boldsymbol{x}_0^T \boldsymbol{\Omega}_0 \boldsymbol{x}_0 + \\
&\qquad \sum_k (\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1}))^T \boldsymbol{R}_k^{-1} (\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{u}_k, \boldsymbol{x}_{k-1})) + \\
&\qquad \sum_k \sum_{\mathrm{i}} (\boldsymbol{z}_k^{\mathrm{i}} - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_{\mathrm{i}}))^T \boldsymbol{Q}_k^{-1} (\boldsymbol{z}_k^{\mathrm{i}} - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_{\mathrm{i}}))
\end{aligned}
\tag{2.13}
$$

Notice that every term of the sum in (2.13) 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:

$$
\boldsymbol{y} = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_k \\ \boldsymbol{m} \end{bmatrix} \tag{2.14}
$$

Furthermore every term in (2.13) can be encapsulated into a single notation called the *error function*: $\mathbf{e}_{\mathrm{i}j}(\boldsymbol{y})$. Every pair of indexes i, j in the error function corresponds to a node in the graph, that is, a robot pose or a landmark position. Then $\mathbf{e}_{\mathrm{i}j}(\boldsymbol{y})$ 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 $\boldsymbol{z}_k^{\mathrm{i}} - \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{m}_{\mathrm{i}})$, if indexes correspond to one pose and one landmark, or $\boldsymbol{x}_0$ if $\mathrm{i} = j = 0$. The error function can be seen as difference between the expected and actual odometry or measurement.

Similarly, the information matrix $\boldsymbol{\Omega}_{\mathrm{i}j}$ 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,(2.13) can be written as:

$$
F(\boldsymbol{y}) := -\log(p(\boldsymbol{y} | \boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k})) = \sum_{\langle \mathrm{i}, j \rangle \in \mathcal{C}} \mathbf{e}_{\mathrm{i}j}(\boldsymbol{y})^T \boldsymbol{\Omega}_{\mathrm{i}j} \mathbf{e}_{\mathrm{i}j}(\boldsymbol{y}) \tag{2.15}
$$

Where the constant c is removes, as it is unnecessary. Finally, the most probable state for the map and the poses is determined by solving the following minimization problem:

$$
\boldsymbol{y}^* = \arg \underset{\boldsymbol{y}}{\mathrm{mín}}\, F(\boldsymbol{y}) \tag{2.16}
$$

Taylor Expansion

The terms in (2.13) are quadratic in the functions $\boldsymbol{g}$ and $\boldsymbol{h}$, which are usually nonlinear functions. Having nonlinear dependency of F over $\boldsymbol{y}$ makes (2.16) difficult to solve. A way to simplify the problem is to linearize those functions using a first order Taylor approximation over $\mathbf{e}_{\mathrm{i}j}$:

$$\mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y}) \approx \mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{\mathrm{i}j}\boldsymbol{\Delta y} \tag{2.17}$$

Where $\breve{\boldsymbol{y}}$ an initial estimate of the state, and $\boldsymbol{J}_{\mathrm{i}j}$ is the Jacobian of $\mathbf{e}_{\mathrm{i}j}$ computed at $\breve{\boldsymbol{y}}$. Then, a local approximation of F can be obtained:

$$\begin{aligned}
F(\breve{\boldsymbol{y}} + \boldsymbol{\Delta y}) &= \sum_{\langle \mathrm{i},j\rangle \in \mathbb{C}} \mathbf{e}_{\mathrm{i}j}(\boldsymbol{y} + \boldsymbol{\Delta y})^T \boldsymbol{\Omega}_{\mathrm{i}j} \mathbf{e}_{\mathrm{i}j}(\boldsymbol{y} + \boldsymbol{\Delta y}) \\
&\approx \sum_{\langle \mathrm{i},j\rangle \in \mathbb{C}} (\mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{\mathrm{i}j}\boldsymbol{\Delta y})^T \boldsymbol{\Omega}_{\mathrm{i}j} (\mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}}) + \boldsymbol{J}_{\mathrm{i}j}\boldsymbol{\Delta y}) \\
&= \sum_{\langle \mathrm{i},j\rangle \in \mathbb{C}} \underbrace{\mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}})^T \boldsymbol{\Omega}_{\mathrm{i}j} \mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}})}_{:=k_{\mathrm{i}j}} + 2\underbrace{\mathbf{e}_{\mathrm{i}j}(\breve{\boldsymbol{y}})^T \boldsymbol{\Omega}_{\mathrm{i}j} \boldsymbol{J}_{\mathrm{i}j}}_{:=\boldsymbol{b}_{\mathrm{i}j}} \boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \underbrace{\boldsymbol{J}_{\mathrm{i}j}^T \boldsymbol{\Omega}_{\mathrm{i}j} \boldsymbol{J}_{\mathrm{i}j}}_{:=\boldsymbol{H}_{\mathrm{i}j}} \boldsymbol{\Delta y} \\
&= \sum_{\langle \mathrm{i},j\rangle \in \mathbb{C}} k_{\mathrm{i}j} + 2\boldsymbol{b}_{\mathrm{i}j}\boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \boldsymbol{H}_{\mathrm{i}j} \boldsymbol{\Delta y} \\
&= k + 2\boldsymbol{b}\boldsymbol{\Delta y} + \boldsymbol{\Delta y}^T \boldsymbol{H} \boldsymbol{\Delta y}
\end{aligned} \tag{2.18}$$

Where $\sum k_{\mathrm{i}j} = k$, $\sum \boldsymbol{b}_{\mathrm{i}j} = \boldsymbol{b}$ and $\sum \boldsymbol{H}_{\mathrm{i}j} = \boldsymbol{H}$. $\boldsymbol{H}$ and $\boldsymbol{b}$ are called the information matrix and the information vector of the linearized system, respectively. The expression (2.18) can be minimized in $\boldsymbol{\Delta y}$ by solving the linear system:

$$\boldsymbol{H}\boldsymbol{\Delta y}^* = -\boldsymbol{b} \tag{2.19}$$

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

$$\boldsymbol{y}^* = \breve{\boldsymbol{y}} + \boldsymbol{\Delta y}^* \tag{2.20}$$

The popular Gauss-Newton algorithm iterates several times between the steps of linearizing the system in (2.18), solving the linear system in (2.19), and updating the state in (2.20).

Structure of the Linearized System

Of all steps of the Gauss-Newton algorithm, the linear system solving in (2.19) is the most computational expensive, because it involves a matrix inversion (in practice or 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. We'll see that actually, the information matrix $\boldsymbol{H}$ has and underlying structure that make the solving of the system (2.19), more efficient and precise.

Looking at $\mathbf{e}_{\mathrm{i}j}(\boldsymbol{y})$ 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 $\boldsymbol{J}_{\mathrm{i}j}$ of $\mathbf{e}_{\mathrm{i}j}$ has an special structure formed by 2 blocks, and the rest is zero:

$$\boldsymbol{J}_{\mathrm{i}j} = \begin{bmatrix} \underbrace{\mathbf{0}\ldots\mathbf{0}\ \underbrace{\boldsymbol{A}_{\mathrm{i}j}}_{\mathrm{i}}\ \mathbf{0}\ldots\mathbf{0}\ \underbrace{\boldsymbol{B}_{\mathrm{i}j}}_{j}\ \mathbf{0}\ldots\mathbf{0}}_{\boldsymbol{y}} \end{bmatrix} \tag{2.21}$$

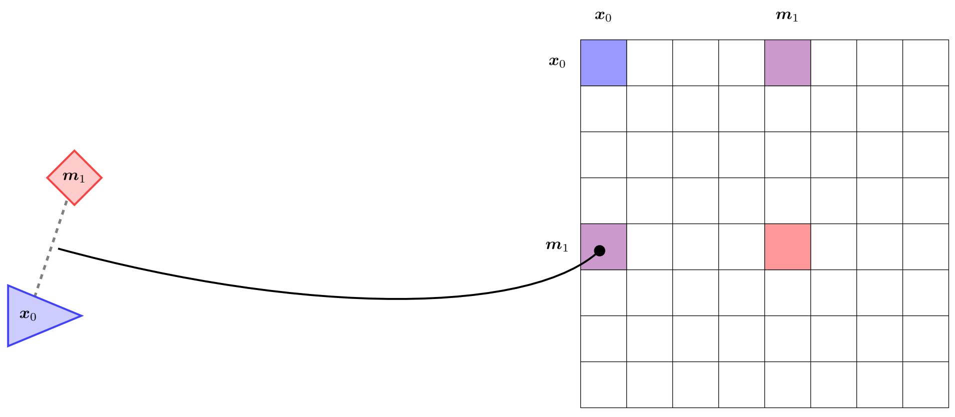
Where $\boldsymbol{A}_{\mathrm{i}j}$ and $\boldsymbol{B}_{\mathrm{i}j}$ are the matrices of the derivatives of the functions in $\mathbf{e}_{\mathrm{i}j}$ with respect the variables in nodes i and j respectively. This makes the matrix $\boldsymbol{H}_{\mathrm{i}j}$ to be formed of four blocks, and the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

$$\boldsymbol{H}_{\mathrm{i}j} = \begin{bmatrix} \ddots & & & & \\ & \boldsymbol{A}_{\mathrm{i}j}^T\boldsymbol{\Omega}_{\mathrm{i}j}\boldsymbol{A}_{\mathrm{i}j} & \ldots & \boldsymbol{A}_{\mathrm{i}j}^T\boldsymbol{\Omega}_{\mathrm{i}j}\boldsymbol{B}_{\mathrm{i}j} & \\ & \vdots & & \vdots & \\ & \boldsymbol{B}_{\mathrm{i}j}^T\boldsymbol{\Omega}_{\mathrm{i}j}\boldsymbol{A}_{\mathrm{i}j} & \ldots & \boldsymbol{B}_{\mathrm{i}j}^T\boldsymbol{\Omega}_{\mathrm{i}j}\boldsymbol{B}_{\mathrm{i}j} & \\ & & & & \ddots \end{bmatrix} \tag{2.22}$$

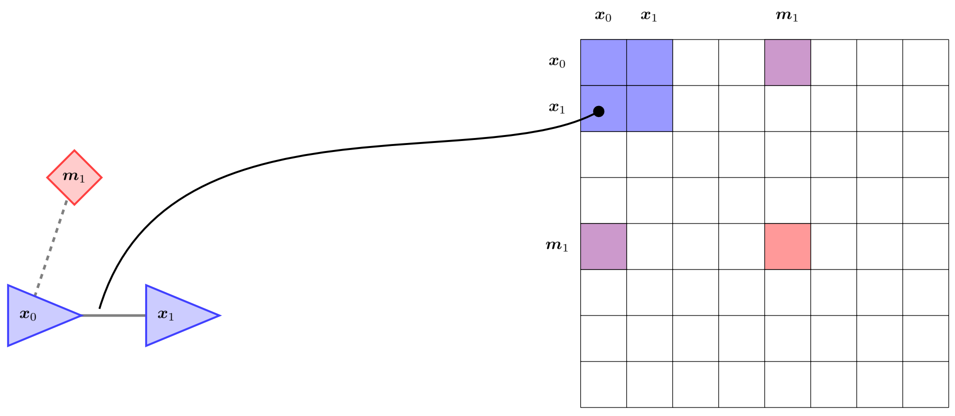
Where all the zero entries are omitted. The off-diagonal block represent the relative information given by the edge $\langle \mathrm{i}, j\rangle$. Therefore, every term $\boldsymbol{H}_{\mathrm{i}j}$ adds 4 blocks to the information matrix $\boldsymbol{H}$.

However, for the SLAM problem, not every combination of block (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 correspond to a block from (2.22).

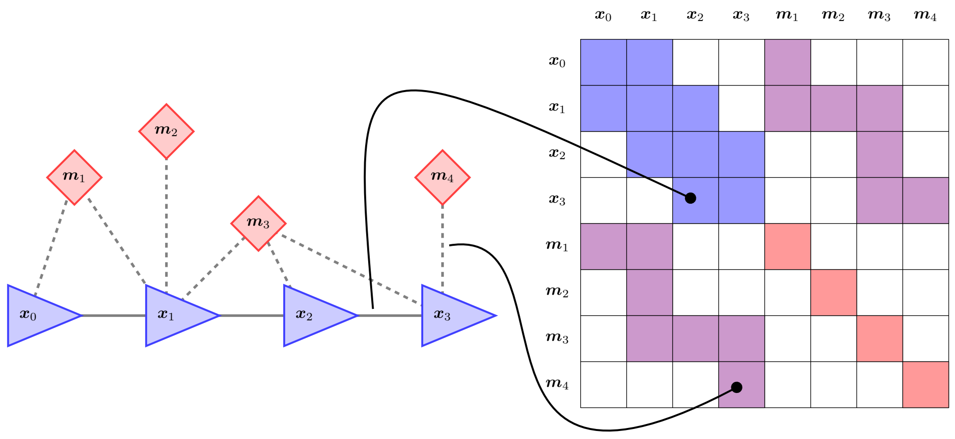
The information matrix can be divided in four submatrices. The top-left matrix (filled with blue squares) has the relative information between to pair of poses. Since the information acquired is only relative to two consecutive poses, this submatrix has a band diagonal structure, leaving all the other elements in zero. The top-right and bottom-left submatrices are equivalent, and contains the relative information between a pose and a measured land-



(a) Block addition to the information matrix after a landmark measurement.



(b) Block addition to the information matrix after a robot movement.



(c) Shows the block structure of the information matrix after 3 robot movements and 7 measurement.

Figura 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.

mark. The number of block 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 pair of landmarks. Since there are not direct measurement between two landmarks, this submatrix is leaved as a block diagonal matrix.

Therefore it's 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. This kind of matrices are called *sparse matrices*. Fast and efficient algorithm exist to solve (2.19) 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.

2.2.3. 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 [2]. One of the first papers to give a solution to SLAM was [8] using the Extended Kalman Filter technique. This solution is now known EKF SLAM.

One of the first papers to state SLAM as an optimization over a graph is [7] by Li and Milios. They were the first to treat motion and measurement in a equally as information constrains, which differed form standard EKF view 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 [10] 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 2.2.2. However, it not specify the way of solving the optimization in (2.16).

From there, several improvement have been made to the general idea of the GraphSLAM algorithm. In [1] a similar graph representation is used to solve SLAM, called $\sqrt{\text{SAM}}$, but it used a *smoothing* approach, meaning that the entire robot trajectory up to the current time is estimated. They implement a batch and incremental algorithm for SLAM solving, and use either QR or Cholesky factorization of the linear equation solving, with variable ordering for complexity improvement.

An updated version of $\sqrt{\text{SAM}}$, called iSAM, is presented in [4]. It improves the performance in the incremental version, by directly updating the square root information matrix with new measurement 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 [3], iSAM2, in which they define a new data structure, called *Bayes Tree*, that allows them to improve the performance in the online case, by updating only the necessary variables in the linearization point after arrival of new data.

In [6] a robust consistency-based loop-closure verification method, Realizing Reversing and Recovering (RRR) algorithm, is presented for de detection and correction of incorrect loop closure generated by the SLAM algorithm. Incorrect loop closure appear when a robot visit similar looking locations, and 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 [5]. 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.

Capítulo 3

Methodology and Implementation

In this chapter the implementation developed for the GraphSLAM algorithm is presented and explained. The implementation 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^{2}o framework used in this work provides of a least squares solver for the optimization of equation (2.16), as well as a protocol to define a graph in the SLAM problem. g^{2}o is well optimized and it has several options for the solver, so the known correspondence version of the GraphSLAM algorithm is relative straightforward to implement.

However, g^{2}o doesn't 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^{2}o Protocol

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

In the SLAM context, nodes itself can be of two types, pose nodes and landmark nodes. Pose nodes represent the pose of the robot, in the 2D case they consist in 3 variables: robot's x and y position, and robot's orientation θ. In g^{2}o 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 represents 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 measurement are given

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

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

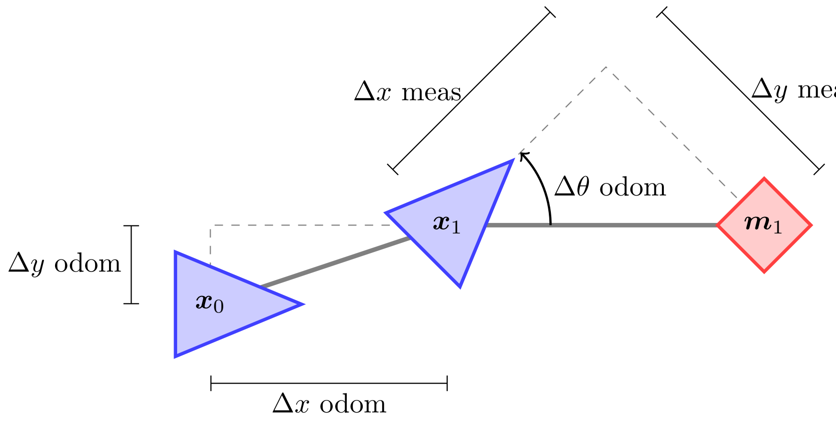


Figura 3.1: Ilustration of odometry and measurements values in g^{2}o.

For GraphSLAM to work correctly, one must also provide to the algorithm of 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) model respectively. g^{2}o works with the inverse of the covariance matrix, known as the information matrix however, but these representation are equivalent in terms of the knowledge of the system. Since covariance matrices (and their inverse) are symmetric, only the upper diagonal elements are needed. In this work it is assumed that the model's uncertainly 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:

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

Finally, nodes must be indexed so they can be distinguishable form 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 summarize the g^{2}o notation to represent nodes and edges.

Graph element	Notation
Pose node	`VERTEX_SE2 id x y t`
Landmark node	`VERTEX_SE2_XY id x y`
Odometry edge	`EDGE_SE2 id1 id2 dx dy dt ipxx ipxy ipxt ipyy ipyt ipyy`
Measurement edge	`EDGE_SE2_XY id1 id2 dx dy ilxx ilxy ilyy`

Tabla 3.1: g^{2}o protocol for node and edge definition.

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. Aside of the first pose, it is assume that only information about robot odometry and measurement is provided, so only data from edges is known.

To use the data in the framework, it can be written in a plain text, which is then uploaded to g^{2}o.

3.2. The Known Correspondence Case

In the known correspondence case each measurement incorporates the information of the landmark sensed form the map. It's equivalent to know 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 includes:

1. The sparse solver for the inversion in (2.19): Cholesky solver, PCG solver
2. The optimization algorithm for solving (2.18)-(2.20): Gauss-Newton, Levenberg-Marquardt
3. The number of iteration for the algorithm to stop.

For the sparse solver, g^{2}o uses third-parties 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's 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 make possible to test the GraphSLAM algorithm for different noise levels. The simulation also allows to compare the results with the ground truth. The ground true estimate to be achieved, and it's 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 3.2 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 follows the distribution assumed for the model. They are usually generated

[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`

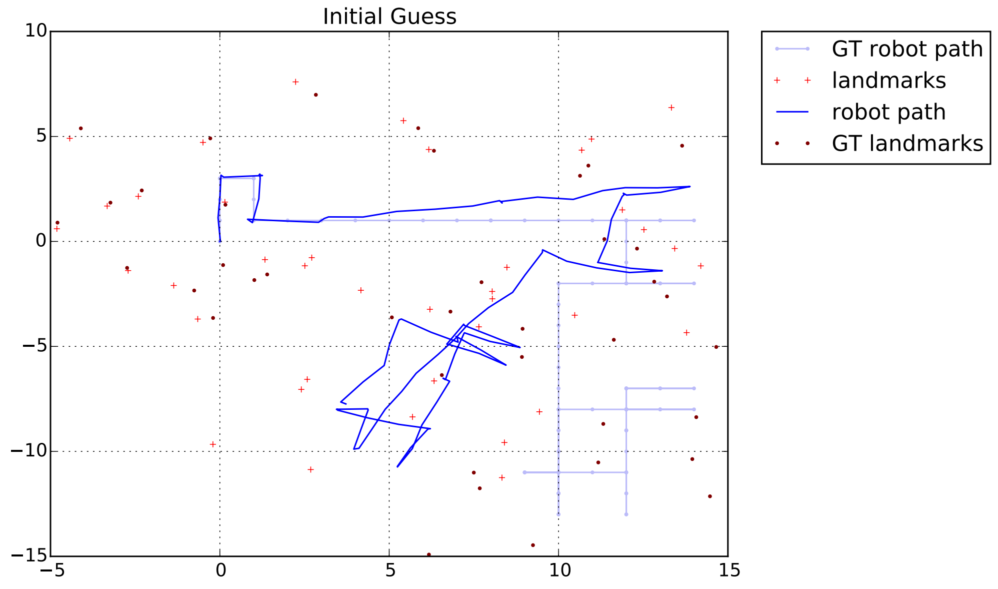


Figura 3.2: Example of an initialization of a SLAM simulation. The light blue line is ground truth path, and dark red circles are ground truth landmark. The blue line is odometry path, and red crosses are the initial guess for landmarks.

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()
```

by sensors malfunctions and tend to have extreme values, far away form the expected measurement.

From equation (2.15), 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 $\mathbf{e}_{ij}(\boldsymbol{y})$ in (2.15), so that high values of $\mathbf{e}_{ij}$ 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's width, that is the point on the function in which the kernel effect start. Figure 3.3 shows the plots of different kernels in g^{2}o.

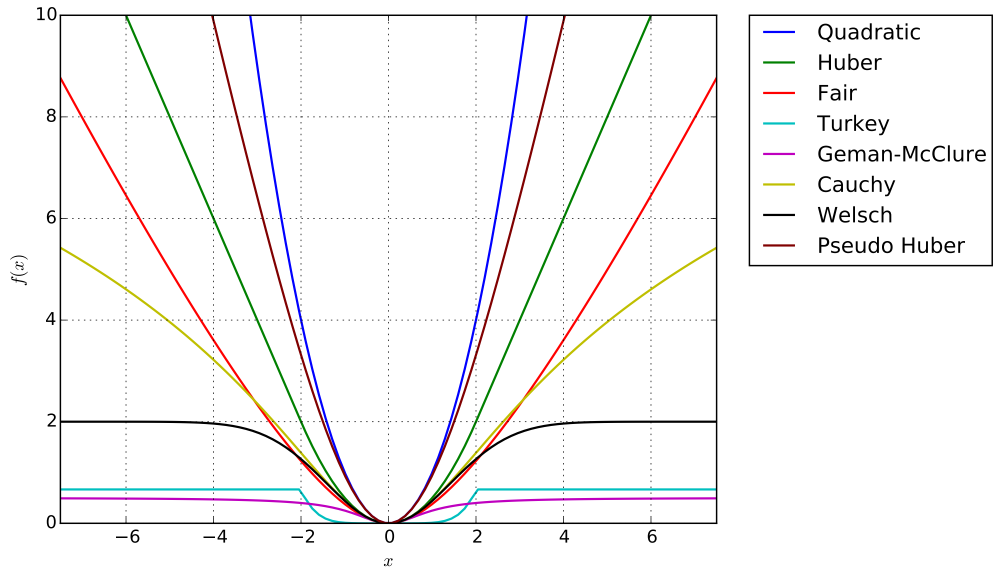


Figura 3.3: Plots of different robust kernels with equals width, compared with the quadratic function.

In this work a Huber kernel of width 1 is used.

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 in the map. The g^{2}o framework does not provides 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 in the one presented in [10], with some differences to take in account the speed, and false alarms.

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 $\boldsymbol{\Delta}_{j,k} = \boldsymbol{m}_j - \boldsymbol{m}_k$ is defined as the difference in position of landmark $\boldsymbol{m}_j$ and $\boldsymbol{m}_k$. The posterior probability of $\boldsymbol{\Delta}_{j,k}$ over the measurements and control inputs is given by:

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

Where $\boldsymbol{\mu}_{\boldsymbol{\Delta}_{j,k}}$ is the current estimate of the landmark's difference. It can be easily computed as $\boldsymbol{\mu}_{\boldsymbol{m}_j} - \boldsymbol{\mu}_{\boldsymbol{m}_k}$, where $\boldsymbol{\mu}_{\boldsymbol{m}_j}$ and $\boldsymbol{\mu}_{\boldsymbol{m}_k}$ are the current estimate of both landmarks.

Matrix $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{j,k}}$ is the covariance matrix marginalized over landmarks j and k. Since it has been assumed a normal distribution for the estimate, $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{j,k}}$ can be computed using the marginalization lemma [10]. In practice, g^{2}o provides a function to compute $\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{j,k}}$.

When two landmarks are equivalent it is expected that their position is the same, hence $\Delta_{j,k} = 0$. Evaluating this in the posterior probability (3.2) gives the likelihood of landmark equivalence:

$$\pi_{j=k} := p(\boldsymbol{\Delta}_{j,k} = 0|\boldsymbol{z}_{1:k}, \boldsymbol{u}_{1:k}) = \det(2\pi\boldsymbol{\Sigma}_{\boldsymbol{\Delta}_{j,k}})^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\boldsymbol{\mu}^T_{\boldsymbol{\Delta}_{j,k}} \boldsymbol{\Sigma}^{-1}_{\boldsymbol{\Delta}_{j,k}} \boldsymbol{\mu}_{\boldsymbol{\Delta}_{j,k}}\right\} \tag{3.3}$$

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

3.3.2. The Unknown Correspondence Algorithm

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

The algorithm works as follows: first all landmarks are initialized as if each measurement correspond 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
optimizer.setParameters(parameters)
optimizer.loadData(data)
optimizer.genInitialGuess()

while association found do
    for all pair of landmark ($j$,$k$) do
        if correspondenceTest($j$,$k$) $\geq \chi$ then
            optimizer.merge($j$,$k$)
        end if
    end for
    optimizer.solve(numberIterations)
end while

optimizer.writeData()
```

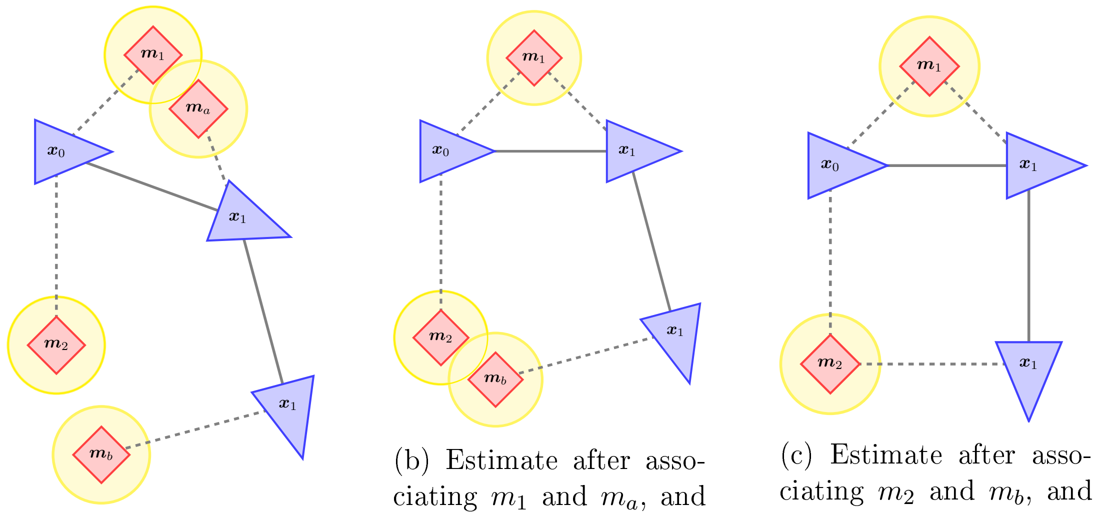
3.3.3. Speeding up the Unknown Correspondence Algorithm

Algorithm 2 is inefficient. In particular the correspondence test is run over every pair of landmarks at each iteration. The possible 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 have shown that the bottleneck of the algorithm is the correspondence test, in particular, the computation of the marginalized covariance $\mathbf{\Sigma}_{\mathbf{\Delta}_{j,k}}$, so is necessary call this function as less often as possible. The optimization of the algorithm is essential to run GraphSLAM in large datasets. The next subsections presents the strategies adopted to improve the algorithm performance.

Incremental Optimization

Incremental optimization is based in the following principle: landmarks measured late in the robot's path are subject to the accumulated uncertainty of all past robot's poses. This fact is can be visualized in the simulation in Figure 3.2. This has as consequence that equivalent landmarks found late in the trajectory will be merged only when all the previous estimates of the pose has already been corrected by the algorithm. Testing correspondence between these landmarks earlier is pointless, simply because their uncertainty is to 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 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 3.4 illustrate 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.

Figura 3.4: 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 uncertainty 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 landmark, it's never tested again, since only current measurements are tested. If a test failed at some point, but posterior correction make it possible for the test to pass, this association will be miss. 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 pair 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 has the is able to choose the frequency with which the full optimization are run. It is done by setting the number of incremental optimization performed between two full optimizations.

Distant Test

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

Defining a value for the threshold is non trivial. Two method 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 exist 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 $\boldsymbol{\delta}_{j,k}$ between two landmarks. Its pdf is given by:

$$p(\boldsymbol{\delta}_{j,k}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k}) = \frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{j,k}}}} \exp\left\{-\frac{(\boldsymbol{\delta}_{j,k}-\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}})^2}{2\sigma^2_{\boldsymbol{\delta}_{j,k}}}\right\} \tag{3.4}$$

$$\mathrm{d}_{j=k} := p(\boldsymbol{\delta}_{j,k}=0|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k}) = \frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{j,k}}}} \exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{j,k}}}{2\sigma^2_{\boldsymbol{\delta}_{j,k}}}\right\} \tag{3.5}$$

Which is equivalent to the pdf $p(\boldsymbol{\Delta}_{j,k}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k})$ in (3.2), for the one dimensional case. $\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{j,k}}$ is the euclidean distance between the mean of the landmarks, and $\sigma^2_{\boldsymbol{\delta}_{j,k}}$ is the variance of $p(\boldsymbol{\Delta}_{j,k}|\boldsymbol{z}_{1:k},\boldsymbol{u}_{1:k})$ projected in the line between $\boldsymbol{\mu}^2_{\boldsymbol{\Delta}_{j,k}}$ and $(0,0)$. For this analysis, the exact value of $\sigma^2_{\boldsymbol{\delta}_{j,k}}$ is not important.

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

$$\max_{\mathrm{d}_{j=k}\geq\chi} \quad |\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}}| \tag{3.6}$$

Considering only the positive values of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}}$, $\mathrm{d}_{j=k}$ becomes a decreasing function. By this fact, it can be seen that maximum at (3.6) is achieved when $\mathrm{d}_{j=k}=\chi$. A visual demonstration of this is shown in Figure 3.5. Expanding the equality found:

$$\mathrm{d}_{j=k} = \chi \tag{3.7}$$

$$\frac{1}{\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{j,k}}}} \exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{j,k}}}{2\sigma^2_{\boldsymbol{\delta}_{j,k}}}\right\} = \chi \tag{3.8}$$

$$\exp\left\{-\frac{\boldsymbol{\mu}^2_{\boldsymbol{\delta}_{j,k}}}{2\sigma^2_{\boldsymbol{\delta}_{j,k}}}\right\} = \sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{j,k}}}\chi \tag{3.9}$$

$$\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}} = \sqrt{-2\sigma^2_{\boldsymbol{\delta}_{j,k}}\log(\sqrt{2\pi\sigma^2_{\boldsymbol{\delta}_{j,k}}}\chi)} \tag{3.10}$$

So the maximum distance for an association is given by (3.10). Unfortunately the distance is a function of $\sigma^2_{\boldsymbol{\delta}_{j,k}}$, which is computationally expensive to get. However expression (3.10) is concave in $\sigma^2_{\boldsymbol{\delta}_{j,k}}$, so a global maximum can be found (see Figure 3.6). Intuitively this means that neither a large nor a low value of $\sigma^2_{\boldsymbol{\delta}_{j,k}}$ is good for associating landmarks at great distances.

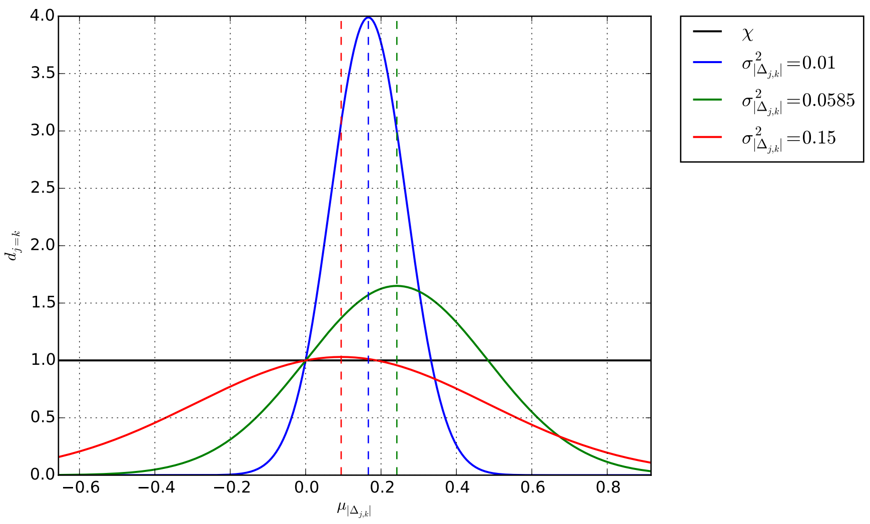


Figura 3.5: Plots of the pdf of the distance between two landmarks for different values of $\sigma^2_{\boldsymbol{\delta}_{j,k}}$. The plots shows that maximum distance from zero is achieve when, $\mathrm{d}_{j=k} = \chi$. Note that the function that achieve more distance is neither the one with more nor with less variance.

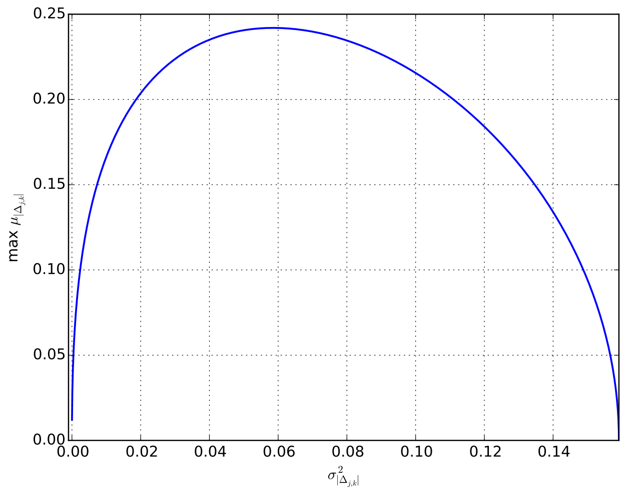


Figura 3.6: Plot of the maximum distance $\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}}$ in function of $\sigma^2_{\boldsymbol{\delta}_{j,k}}$. It can be seen that is a concave function.

Using differential calculus it can be found maximum of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}}$ for $\sigma^2_{\boldsymbol{\delta}_{j,k}}$ is achieved in $\sigma^2_{\boldsymbol{\delta}_{j,k}} = \frac{1}{2\pi e\chi}$, which yields a value of $\boldsymbol{\mu}_{\boldsymbol{\delta}_{j,k}} = \frac{1}{\sqrt{2\pi e}\chi}$. This value can be used as a distant 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.

Capítulo 4

Results

Conclusión

Bibliografía

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

[3] 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.

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

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

[6] 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.

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

[8] 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.

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

[10] 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.